数据结构

树状数组

普通

```
template<typename T>
struct Fenwick {
    int n;
    std::vector <T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n + 1, T{});
    }

    void add(int x, const T &v) {
        if (x <= 0 || x > n) return;
        for (int i = x; i <= n; i += i & -i) {
            a[i] = a[i] + v;
        }
    }

    T Query(int x) {
        if (x <= 0) return T{};
        if (x > n) x = n;
        T ans{};
        for (int i = x; i != 0; i -= i & -i) {
            ans = ans + a[i];
        }
        return ans;
    }

    T range_Query(int l, int r) {
        if (l > r) return 0;
        return Query(r) - Query(l - 1);
    }

    int kth(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i] < k) {
                x += i;
```

```
                cur = cur + a[x];
            }
        }
        return x + 1;
    }
};
```

区间加

```
template<typename T>
struct Range_Fenwick {
    int n;
    Fenwick <T> a, b;

    Range_Fenwick (int _n = 0) {
        init (_n);
    }

    void init (int _n) {
        n = _n;
        a.init(n); b.init(n);
    }

    void range_Change (int l, int r, const T& k) {
        a.add(l, k); a.add(r + 1, -k);
        b.add(l, k * l); b.add(r + 1, -k * (r + 1)) ;
    }

    T range_Query (int l, int r) {
        return (r + 1) * a.Query(r) - l * a.Query(l - 1) - b.range_Query(l, r);
    }

    int kth(const T &k) {
        int x = 0;
        T cur0{}, cur1{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && (cur0 + a.a[x + i]) * (x + i + 1) - (cur1 + b.a[x +
i]) < k) {
                x += i;
                cur0 = cur0 + a.a[x];
                cur1 = cur1 + b.a[x];
            }
        }
        return x + 1;
    }
};
```

二维

```
template<typename T>
struct Two_dimensional_Fenwick {
    struct Base_Fenwick {
        int n, m;
        std::vector <std::vector<T>> s;
```

```
    Base_Fenwick(int _n = 0, int _m = 0) {
        init(_n, _m);
    }

    void init(int _n, int _m) {
        n = _n, m = _m;
        s.assign(n + 1, std::vector<T>(m + 1, T()));
    }

    void change(int x, int y, const T &v) {
        if (x <= 0 || y <= 0) return;
        if (x > n) x = n;
        if (y > m) y = m;
        for (int i = x; i <= n; i += i & (-i))
            for (int j = y; j <= m; j += j & (-j))
                s[i][j] += v;
    }

    T Query(int x, int y) {
        if (x <= 0 || y <= 0) return T();
        if (x > n) x = n;
        if (y > m) y = m;
        T ans = 0;
        for (int i = x; i != 0; i -= i & (-i))
            for (int j = y; j != 0; j -= j & (-j))
                ans += s[i][j];
        return ans;
    }
};

int n, m;
Base_Fenwick A, B, C, D;

Two_dimensional_Fenwick(int _n = 0, int _m = 0) {
    init(_n, _m);
}

void init(int _n, int _m) {
    n = _n, m = _m;
    A.init(n, m);
    B.init(n, m);
    C.init(n, m);
    D.init(n, m);
}

void Base_add(int x, int y, int v) {
    A.change(x, y, v);
    B.change(x, y, v * x);
    C.change(x, y, v * y);
    D.change(x, y, v * x * y);
}

T Base_Query(int x, int y) {
    return A.Query(x, y) * (x * y + x + y + 1)
           - B.Query(x, y) * (y + 1)
```

```
                - C.Query(x, y) * (x + 1)
                + D.Query(x, y);
    }

    void add(int x0, int y0, int x1, int y1, int v) {
        Base_add(x0, y0, v);
        Base_add(x0, y1 + 1, -v);
        Base_add(x1 + 1, y0, -v);
        Base_add(x1 + 1, y1 + 1, v);
    }

    T Query(int x0, int y0, int x1, int y1) {
        return Base_Query(x1, y1) - Base_Query(x0 - 1, y1)
               - Base_Query(x1, y0 - 1) + Base_Query(x0 - 1, y0 - 1);
    }
};
```

RMQ

```
/**
 * author:jiangly
 * pretreatment:O(n)
 * Inquire:O(1)
*/
template<class T,
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
```

```
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

线段树

单点

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
```

```
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
```

```
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
```

区间

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
```

```
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
```

```
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
```

```
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Tag {
    i64 a = 0, b = 0;
    void apply(Tag t) {
        a = std::min(a, b + t.a);
        b += t.b;
    }
};

int k;

struct Info {
    i64 x = 0;
    void apply(Tag t) {
        x += t.a;
        if (x < 0) {
            x = (x % k + k) % k;
        }
        x += t.b - t.a;
    }
};
Info operator+(Info a, Info b) {
    return {a.x + b.x};
}
```

tourist zkw 线段树（精简版）区间最大值

```
struct SegmTree {
  vector<int> T; int n;
  SegmTree(int n) : T(2 * n, (int)-2e9), n(n) {}

  void Update(int pos, int val) {
    for (T[pos += n] = val; pos > 1; pos /= 2)
      T[pos / 2] = max(T[pos], T[pos ^ 1]);
  }

  int Query(int b, int e) {
    int res = -2e9;
    for (b += n, e += n; b < e; b /= 2, e /= 2) {
      if (b % 2) res = max(res, T[b++]);
      if (e % 2) res = max(res, T[--e]);
    }
    return res;
  }
```

```
};
```

动态开点线段树

```
uint8_t _buf[262144000];
uint8_t *head = _buf;

template<typename value_type, typename modify_type>
struct segment_tree {
    int n;
    using size_type = long long;
    using idx_type = uint32_t;
    struct node {
        value_type m_value;
        modify_type m_modify;
        idx_type ls, rs;
        node () :m_value(), m_modify(), ls(), rs() {}
        void reset() {
            *this = node();
        }
    };
    using ptr = node *;
    idx_type root{0};
    idx_type _new() {
        idx_type cur = (head += sizeof(node)) - _buf;
        // ptr p = ptr(_buf + cur);
        // p->reset();
        return cur;
    }
    void _apply(idx_type &cur, const modify_type &v, size_type l, size_type r) {
        if (!cur) {
            cur = _new();
        }
        ptr p = ptr(_buf + cur);
        p->m_value.apply(v, l, r);
        p->m_modify.apply(v);
    }
    void _push(idx_type &cur, size_type l, size_type r, size_type m) {
        ptr p = ptr(_buf + cur);
        // assert(l < r);
        if (!bool(p->m_modify)) {
            return;
        }
        _apply(p->ls, p->m_modify, l, m);
        _apply(p->rs, p->m_modify, m, r);
        p->m_modify.clear();
    }
    void _pull(idx_type &cur, size_type l, size_type r, size_type m) {
        ptr p = ptr(_buf + cur);
        ptr lc = ptr(_buf + p->ls);
        ptr rc = ptr(_buf + p->rs);
        p->m_value.set(value_type::op(lc->m_value, rc->m_value, l, r, m));
    }
    size_type floor, ceil;
    segment_tree(size_type floor, size_type ceil) : floor(floor) , ceil(ceil) {}
```

```
    void _range_modify(idx_type &cur, const modify_type &v, size_type l,
size_type r, size_type x, size_type y) {
        if (!cur)
            cur = _new();
        ptr p = ptr(_buf + cur);
        size_type m = (l + r) >> 1;
        if (x <= l && r <= y) {
            _apply(cur, v, l, r);
            return;
        }
        _push(cur, l, r, m);
        if (m > x)
            _range_modify(p->ls, v, l, m, x, y);
        if (m < y)
            _range_modify(p->rs, v, m, r, x, y);
        _pull(cur, l, r, m);
    }
    void range_modify(size_type x, size_type y, const modify_type &v) {
        _range_modify(root, v, floor, ceil, x, y);
    }
    value_type::op_type _range_query(idx_type &cur, size_type l, size_type r,
size_type x, size_type y) {
        if (!cur)
            return value_type::op(l, r);
        ptr p = ptr(_buf + cur);
        size_type m = (l + r) >> 1;
        if (x <= l && r <= y) {
            return value_type::op(p->m_value);
        }
        _push(cur, l, r, m);
        if (m >= y) {
            return _range_query(p->ls, l, m, x, y);
        } else if (m <= x) {
            return _range_query(p->rs, m, r, x, y);
        } else {
            return value_type::op(_range_query(p->ls, l, m, x, y),
_range_query(p->rs, m, r, x, y));
        }
    }
    value_type::op_type range_query(size_type x, size_type y) {
        return _range_query(root, floor, ceil, x, y);
    }
    void _show(idx_type &cur, size_type l, size_type r) {
        if (!cur)
            return;
        ptr p = ptr(_buf + cur);
        size_type m = (l + r) >> 1;
        if (r - l == 1) {
            debug(p->m_value.x);
            return;
        }
        _push(cur, l, r, m);
            _show(p->ls, l, m);
            _show(p->rs, m, r);
        _pull(cur, l, r, m);
    }
```

```
    void show() {
        _show(root, floor, ceil);
    }
    ptr ptr_root() { return ptr(_buf + root); }
};

struct Tag {
    i64 x = 0;
    operator bool() {
        return x != 0;
    }
    void apply(const Tag &rhs) {
        x += rhs.x;
    }
    void clear() {
        x = 0;
    }
};

struct Info {
    i64 x;
    void apply(const Tag &rhs, i64 l, i64 r) {
        x += (r - l) * rhs.x;
    }
    using op_type = Info;
    static Info op(Info lhs, Info rhs, i64 l, i64 r, i64 m) {
        return {lhs.x + rhs.x};
    }
    static Info op(Info lhs, Info rhs) {
        return {lhs.x + rhs.x};
    }
    static Info op(i64 l, i64 r) {
        // assert(false);
        return {0};
    }
    static Info op(Info rhs) {
        return rhs;
    }
    void set(Info rhs) {
        x = rhs.x;
    }
};

using SegmentTree = segment_tree<Info, Tag>;
```

线段树分治

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
```

```
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
        };
        build(1, 0, n);
    }
    void rangeChange(int x, int y, const Info &tag) {
        std::function<void(int, int, int, int, int, const Info&)>
            rangeChange = [&] (int p, int l, int r, int x, int y, const Info
&tag) {
            if (l >= y || r <= x) {
                return;
            }
            if (l >= x && r <= y) {
                info[p].apply(tag);
                return;
            }
            int m = (l + r) / 2;
            rangeChange(p << 1, l, m, x, y, tag);
            rangeChange(p << 1 | 1, m, r, x, y, tag);
        };
        rangeChange(1, 0, n, x, y, tag);
    }
};

struct Info {
    vector<array<ll, 2>> x;
    void apply(const Info& tag) {
        for (auto u : tag.x) {
            x.push_back(u);
        }
    }
};

using Segmenttree = SegmentTree<Info>;
```

可持久化线段树

```
constexpr int max_size = 262144000;
uint8_t _buf[max_size];
uint8_t *head = _buf;

template<typename Info>
struct persistent_segment_tree {
    int n;
    struct node {
        Info m_info;
        int ls, rs;
        node () : m_info(), ls(), rs() {}
        void reset () {
            *this = node();
        }
    };
    using pointer = node *;
    int _new() {
        assert(head < _buf + max_size);
        return (head += sizeof(node)) - _buf;
    }
    vector<int> root;
    persistent_segment_tree(): n(0) {}
    persistent_segment_tree(int _n, Info _v = Info()) {
        _init(std::vector(_n, _v));
    }
    template<typename T>
    persistent_segment_tree(std::vector<T> _init) {
        _init(_init);
    }
    void _pull(int cur1) {
        pointer p1 = pointer(_buf + cur1);
        pointer lc = pointer(_buf + p1->ls);
        pointer rc = pointer(_buf + p1->rs);
        p1->m_info.set(Info::op(lc->m_info, rc->m_info));
    }
    template<typename T>
    void _init(std::vector<T> _init) {
        n = _init.size();
        root.push_back(_new());
        std::function<void(int, int, int)>
        build = [&] (int cur, int l, int r) {
            pointer p = pointer(_buf + cur);
            if (r - l == 1) {
                p->m_info = _init[l];
                return;
            }
            int m = (l + r) / 2;
            p->ls = _new(), p->rs = _new();
            build(p->ls, l, m), build(p->rs, m, r);
            _pull(cur);
        };
        build(root.back(), 0, n);
    }
```

```
    template<typename Tag>
    void _modify(int cur0, int cur1, const Tag &v, int l, int r, int x) {
        pointer p0 = pointer(_buf + cur0), p1 = pointer(_buf + cur1);
        if (r - l == 1) {
            p1->m_info = p0->m_info;
            p1->m_info.apply(v);
            return;
        }
        int m = (l + r) >> 1;
        if (m > x) {
            p1->ls = _new();
            p1->rs = p0->rs;
            _modify(p0->ls, p1->ls, v, l, m, x);
        } else {
            p1->ls = p0->ls;
            p1->rs = _new();
            _modify(p0->rs, p1->rs, v, m, r, x);
        }
        _pull(cur1);
    }
    template<typename Tag>
    void modify(int x, const Tag &v, int from = -1) {
        int cur0 = (from == -1 ? root.back() : root[from]);
        int cur1 = _new();
        root.push_back(cur1);
        _modify(cur0, cur1, v, 0, n, x);
    }
    typename Info::op_t _range_query(int cur0, int cur1, int l, int r, int x, int
y) {
        pointer p0 = pointer(_buf + cur0), p1 = pointer(_buf + cur1);
        if (x <= l && r <= y) {
            return Info::del(p1->m_info, p0->m_info);
        }
        int m = (l + r) >> 1;
        if (m >= y) {
            return _range_query(p0->ls, p1->ls, l, m, x, y);
        } else if (m <= x) {
            return _range_query(p0->rs, p1->rs, m, r, x, y);
        } else {
            return Info::op(_range_query(p0->ls, p1->ls, l, m, x, y),
_range_query(p0->rs, p1->rs, m, r, x, y));
        }
    }
    typename Info::op_t range_query(int from, int to, int x, int y) {
        return _range_query(root[from], root[to], 0, n, x, y);
    }
    typename Info::op1_t _kth(int cur0, int cur1, int l, int r, i64 k) {
        pointer p0 = pointer(_buf + cur0), p1 = pointer(_buf + cur1);
        pointer ls0 = pointer(_buf + p0->ls), ls1 = pointer(_buf + p1->ls);
        if (r - l == 1) {
            return Info::op1(l, Info::op1(k));
        }
        int m = (l + r) >> 1;
        typename Info::op1_t lhs = Info::del1(ls1->m_info, ls0->m_info);
        if (int(lhs) >= k) {
            return _kth(p0->ls, p1->ls, l, m, k);
```

```
        } else {
            return Info::op1(lhs, _kth(p0->rs, p1->rs, m, r, k - int(lhs)));
        }
    }
    typename Info::op1_t kth(int from, int to, i64 k) {
        return _kth(root[from], root[to], 0, n, k);
    }
    void _show(int cur, int l, int r) {
        pointer p = pointer(_buf + cur);
        if (r - l == 1) {
            p->m_info.show();
            return;
        }
        int m = (l + r) >> 1;
        _show(p->ls, l, m);
        _show(p->rs, m, r);
    }
    void show(int time) {
        _show(root[time],0, n);
    }
};

struct Info {
    i64 cnt = 0;
    using op_t = int;
    using op1_t = int;
    operator op_t() {
        return cnt;
    }
    void set(op_t rhs) {
        cnt = rhs;
    }
    static op_t op(op_t lhs, op_t rhs) {
        return lhs + rhs;
    }
    static op_t del(op_t lhs, op_t rhs) {
        return lhs - rhs;
    }
    static array<ll, 1> op1 (i64 k) {
        return array<ll, 1>{0};
    }
    static op1_t op1(int x, array<ll, 1> mul) {
        return x;
    }
    static op1_t op1(op1_t lhs, op1_t rhs) {
        return rhs;
    }
    static op1_t del1(op1_t lhs, op1_t rhs) {
        return lhs - rhs;
    }
    void apply(Info x) {
        cnt += x.cnt;
    }
    void show() {
        cerr << cnt << ' ';
    }
```

```
};

using SegmentTree = persistent_segment_tree<Info>;
```

李超线段树

```
template<typename T, class Line, class Cmp>
struct Li_Chao_SegmentTree {
    int n;
    std::vector<int> id;
    std::vector<T> real;
    std::vector<Line> line;
    Cmp cmp;
    Li_Chao_SegmentTree() {}
    Li_Chao_SegmentTree(int _n) {
        init(_n);
    }
    Li_Chao_SegmentTree(const std::vector<T> &_init) {
        init(_init);
    }
    void init(int _n) {
        std::vector<int> _init(_n);
        iota(_init.begin(), _init.end(), 0);
        init(_init);
    }
    void init(const std::vector<T> &_init) {
        n = _init.size();
        id.assign(4 << std::__lg(n), 0);
        line.push_back(Line());
        real = _init;
        sort(real.begin(), real.end());
        real.erase(std::unique(real.begin(), real.end()), real.end());
        real.push_back(real.back() + 1);
    }
    void rangeChange (int x, int y, Line add) {
        int u = line.size();
        line.push_back(add);
        std::function<void(int, int, int, int)>
        range_Change = [&] (int l, int r, int p, int u) {
            int &v = id[p], m = (l + r) / 2;
            if (cmp(line, u, v, real[m])) {
                swap(u, v);
            }
            if (cmp(line, u, v, real[l])) {
                range_Change(l, m, p * 2, u);
            }
            if (cmp(line, u, v, real[r - 1])) {
                range_Change(m, r, p * 2 + 1, u);
            }
        };
        std::function<void(int, int, int)>
        range_find = [&] (int l, int r, int p) {
            if (real[l] >= y || real[r] <= x) {
                return;
            }
```

```
            if (x <= real[l] && real[r] <= y) {
                range_Change(l, r, p, u);
                return;
            }
            int m = (l + r) / 2;
            range_find(l, m, p * 2);
            range_find(m, r, p * 2 + 1);
        };
        range_find(0, n, 1);
    }
    void insert(Line add) {
        rangeChange(real[0], real.back(), add);
    }
    int Query(int x) {
        std::function<int(int, int, int)>
        Query = [&] (int l, int r, int p) {
            if (r - l == 1) {
                return id[p];
            }
            int m = (l + r) / 2;
            int u = id[p], v = -1;
            if (x < real[m]) {
                v = Query(l, m, p * 2);
            } else {
                v = Query(m, r, p * 2 + 1);
            }
            return cmp(line, u, v, x) ? u : v;
        };
        return Query(0, n, 1);
    }
    T slope_dp_Query(int x) {
        return line[Query(x)](x);
    }
};

template<typename T>
struct Line {
    T k, b;
    Line(T k = 0, T b = 0) : k(k), b(b){}
    T operator()(T x) {
        return __int128(k) * x + b;
    }
};
template<>
struct Line<double> {
    double k, b;
    Line(double k = 0, double b = 0) : k(k), b(b){}
    template<typename T>
    Line(T x0, T y0, T x1, T y1) {
        if (x0 == x1) {
            k = 0;
            b = std::max(y0, y1);
        } else {
            k = (y0 - y1) / (0. + x0 - x1);
            b = y0 - k * x0;
        }
```

```
    }
    double operator()(double x) {
        return k * x + b;
    }
};

template<typename T>
struct Cmp {
    bool operator() (vector<Line<T>> &line, int u, int v, T x) {
        return line[u](x) < line[v](x) || (line[u](x) == line[v](x) && u < v);
    }
};
template<>
struct Cmp<double> {
    bool operator() (vector<Line<double>> &line, int u, int v, double x) {
        constexpr double exp = 1e-9;
        return line[u](x) - line[v](x) > exp || (std::abs(line[u](x) - line[v]
(x)) <= exp && u < v);
    }
};

template<typename T, typename T1 = int>
using SegmentTree =
    Li_Chao_SegmentTree<T1, Line<T>, Cmp<T>>;
```

扫描线

```
struct ScanLine {
    int n;
    struct Line {
        int x1, x2, y;
        int type;
        bool operator<(Line another) const {
            return y < another.y;
        }
    };

    struct Info {
        int l, r;
        int len = 0, cnt = 0;
    };
    vector<Info> info;
    vector<Line> line;
    vector<int> X;

    void add(int x1, int y1, int x2, int y2) {
        line.push_back({x1, x2, y1, 1});
        line.push_back({x1, x2, y2, -1});
        X.push_back(x1);
        X.push_back(x2);
    }

    int work(int n) {
        sort(line.begin(), line.end());
        sort(X.begin(), X.end());
```

```
        int tot = unique(X.begin(), X.end()) - X.begin();
        vector<Info> init_;
        for (int i = 0; i < tot - 1; i++) {
            init_.push_back({i + 1, i + 1, 0, 0});
        }
        init(init_);
        int ans = 0;
        for (int i = 0; i < 2 * n - 1; i++) {
            modify(1, line[i].x1, line[i].x2, line[i].type);
            ans += info[1].len * (line[i + 1].y - line[i].y);
        }
        return ans;
    }

    ScanLine() : n(0) {};

    void init(const vector<Info> &_init) {
        n = (int)_init.size();
        info.assign(n * 8, Info());
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            info[p].l = l;
            info[p].r = r;
            if (l == r) {
                info[p] = _init[l - 1];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m + 1, r);
            pull(p);
        };
        build(1, 1, n);
    }

    void pull(int p) {
        if (info[p].cnt) {
            info[p].len = X[info[p].r] - X[info[p].l - 1];
        } else {
            info[p].len = info[2 * p].len + info[2 * p + 1].len;
        }
    }

    void modify(int p, int L, int R, int val) {
        int l = info[p].l;
        int r = info[p].r;
        if (X[r] <= L || R <= X[l - 1]) {
            return;
        }
        if (L <= X[l - 1] && X[r] <= R) {
            info[p].cnt += val;
            pull(p);
            return;
        }
        modify(2 * p, L, R, val);
        modify(2 * p + 1, L, R, val);
        pull(p);
```

```
    }
};
```

link-cut-tree

单点

```
template<class Info>
struct LinkCutTree {
    struct node {
        int s[2], p, tag;
        Info mval;
    };
    int n;
    vector<node> tree;

    int &fa(int x) { return tree[x].p; }
    int &lc(int x) { return tree[x].s[0]; }
    int &rc(int x) { return tree[x].s[1]; }
    bool notroot(int x) {
        return tree[tree[x].p].s[0] == x || tree[tree[x].p].s[1] == x;
    }
    // 不能以0开头
    LinkCutTree(int n) : n(n) { tree.resize(n + 1); tree[0].mval.defaultclear();
}

private:
    void pull(int x) {
        push(rc(x)), push(lc(x));
        tree[x].mval.update(tree[lc(x)].mval, tree[rc(x)].mval);
    }

    void push(int x) {
        if (tree[x].tag) {
            swap(lc(x), rc(x));
            tree[lc(x)].mval.reverse();
            tree[rc(x)].mval.reverse();
            tree[rc(x)].tag ^= 1;
            tree[lc(x)].tag ^= 1;
            tree[x].tag = 0;
        }
    }

    void maintain(int x) {
        if (notroot(x)) maintain(fa(x));
        push(x);
    }

    void rotate(int x) {
        int y = fa(x), z = fa(y);
        int k = rc(y) == x;
        if (notroot(y))
            tree[z].s[rc(z) == y] = x;
        fa(x) = z;
```

```
        tree[y].s[k] = tree[x].s[k ^ 1];
        fa(tree[x].s[k ^ 1]) = y;
        tree[x].s[k ^ 1] = y;
        fa(y) = x;
        pull(y), pull(x);
    }

public:
    void splay(int x) {
        maintain(x);
        while (notroot(x)) {
            int y = fa(x), z = fa(y);
            if (notroot(y))
                ((rc(z) == y) ^ (rc(y) == x))
                ? rotate(x) : rotate(y);
            rotate(x);
        }
    }

    void access(int x) {
        for (int y = 0; x;) {
            splay(x);
            rc(x) = y;
            pull(x);
            y = x;
            x = fa(x);
        }
    }

    void makeroot(int x) {
        access(x);
        splay(x);
        tree[x].tag ^= 1;
    }

    //y变成原树和辅助树的根
    const Info &split(int x, int y) {
        makeroot(x);
        access(y);
        splay(y);
        return tree[y].mval;
    }

    int findroot(int x) {
        access(x);
        splay(x);
        while (lc (x))
            push(x), x = lc(x);
        splay(x);
        return x;
    }

    void link(int x, int y) {
        makeroot(x);
        if (findroot(y) != x) fa(x) = y;
    }
```

```
    void cut(int x, int y) {
        makeroot(x);
        if (findroot(y) == x
            && fa(y) == x && !lc(y)) {
            rc(x) = fa(y) = 0;
            pull(x);
        }
    }

    void modify(int x, const Info &val) {
        splay(x);
        tree[x].mval.modify(val);
        pull(x);
    }

    bool same(int x, int y) {
        makeroot(x);
        return findroot(y) == x;
    }
    node &operator[](int x) {
        return tree[x];
    }
};

struct Info {
    int v = 1; int id = -1; int sum = 0; int max = 0;
    void reverse() {}
    void modify(const Info& rhs) {
        v = rhs.v;
    }
    void update(const Info &lhs, const Info &rhs) {
        sum = lhs.sum + v + rhs.sum;
        max = std::max({lhs.max, id, rhs.max});
    }
    void defaultclear() {
        v = 0;
    }
};

using Tree = LinkCutTree<Info>;
```

区间

```
template<class Info, class Tag>
struct LazyLinkCutTree {
    struct node {
        int s[2], p, tag;
        Info mval;
        Tag mtag;
    };
    int n;
    vector<node> tree;

    int &fa(int x) { return tree[x].p; }
```

```
    int &lc(int x) { return tree[x].s[0]; }
    int &rc(int x) { return tree[x].s[1]; }
    bool notroot(int x) {
        return tree[tree[x].p].s[0] == x || tree[tree[x].p].s[1] == x;
    }
    // 不能以0开头
    LazyLinkCutTree(int n) : n(n) {
        tree.resize(n + 1);
        tree[0].mtag.default_clear();
        tree[0].mval.default_clear();
    }

private:
    void pull(int x) {
        push(rc(x)), push(lc(x));
        tree[x].mval.update(tree[lc(x)].mval, tree[rc(x)].mval);
    }

    void apply(int x, const Tag &rhs) {
        if (x) {
            tree[x].mval.apply(rhs);
            tree[x].mtag.apply(rhs);
        }
    }

    void push(int x) {
        if (tree[x].tag) {
            swap(lc(x), rc(x));
            tree[lc(x)].mval.reverse();
            tree[rc(x)].mval.reverse();
            tree[rc(x)].tag ^= 1;
            tree[lc(x)].tag ^= 1;
            tree[x].tag = 0;
        }
        if (bool(tree[x].mtag)) {
            apply(lc(x), tree[x].mtag);
            apply(rc(x), tree[x].mtag);
            tree[x].mtag.clear();
        }
    }

    void maintain(int x) {
        if (notroot(x)) maintain(fa(x));
        push(x);
    }

    void rotate(int x) {
        int y = fa(x), z = fa(y);
        int k = rc(y) == x;
        if (notroot(y))
            tree[z].s[rc(z) == y] = x;
        fa(x) = z;
        tree[y].s[k] = tree[x].s[k ^ 1];
        fa(tree[x].s[k ^ 1]) = y;
        tree[x].s[k ^ 1] = y;
        fa(y) = x;
```

```
        pull(y), pull(x);
    }

public:
    void splay(int x) {
        maintain(x);
        while (notroot(x)) {
            int y = fa(x), z = fa(y);
            if (notroot(y))
                ((rc(z) == y) ^ (rc(y) == x))
                ? rotate(x) : rotate(y);
            rotate(x);
        }
    }

    void access(int x) {
        for (int y = 0; x;) {
            splay(x);
            rc(x) = y;
            pull(x);
            y = x;
            x = fa(x);
        }
    }

    void makeroot(int x) {
        access(x);
        splay(x);
        tree[x].tag ^= 1;
    }

    //y变成原树和辅助树的根
    const Info &split(int x, int y) {
        makeroot(x);
        access(y);
        splay(y);
        return tree[y].mval;
    }

    int findroot(int x) {
        access(x);
        splay(x);
        while (lc (x))
            push(x), x = lc(x);
        splay(x);
        return x;
    }

    void link(int x, int y) {
        makeroot(x);
        if (findroot(y) != x) fa(x) = y;
    }

    void cut(int x, int y) {
        makeroot(x);
        if (findroot(y) == x
```

```
            && fa(y) == x && !lc(y)) {
            rc(x) = fa(y) = 0;
            pull(x);
        }
    }

    void modify(int x, const Info &val) {
        splay(x);
        tree[x].mval.modify(val);
        pull(x);
    }

    void line_modify(int u, int v, const Tag &rhs) {
        split(u, v);
        apply(v, rhs);
    }

    bool same(int x, int y) {
        makeroot(x);
        return findroot(y) == x;
    }

    node &operator[](int x) {
        return tree[x];
    }
};

struct Tag {
    int set = 0;
    void apply(const Tag &rhs) {
        set = rhs.set;
    }
    void clear() {
        set = 0;
    }
    operator bool() {
        return set != 0;
    }
    void default_clear() {}
};

struct Info {
    int c = 0; int sum = 0, l = 0, r = 0, id = 0;
    void reverse() {
        swap(l, r);
    }
    void modify(const Info& rhs) {
        l = r = c = rhs.c;
    }
    void update(const Info &lhs, const Info &rhs) {
        sum = lhs.sum + (c != lhs.r && lhs.r != 0) + (c != rhs.l && rhs.l != 0) +
rhs.sum;
        l = (lhs.r == 0 ? c : lhs.l);
        r = (rhs.l == 0 ? c : rhs.r);
    }
    void apply(const Tag &rhs) {
```

```
        l = r = c = rhs.set; sum = 0;
    }
    void show() const {
        debug(id);
        cerr << l << ' ' << c << ' ' << r << ' ' << sum << endl;
    }
    void default_clear() {}
};

using Tree = LazyLinkCutTree<Info, Tag>;
```

并查集

普通

```
struct DSU {
    std::vector<int> f, siz;

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

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
```

```
};
```

可撤回

```
struct DSU {
    std::vector<int> fa, size_;
    std::stack <pair<int &, int>> history_size;
    std::stack <pair<int &, int>> history_fa;

    DSU() {}

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

    void init(int n) {
        fa.resize(n);
        std::iota(fa.begin(), fa.end(), 0);
        size_.assign(n, 1);
    }

    int find(int x) {
        while (x != fa[x]) x = fa[x];
        return x;
    }

    int size(int x) {
        return size_[find(x)];
    }

    bool same(int u, int v) {
        return find(u) == find(v);
    }

    void merge(int u, int v) {
        int x = find(u);
        int y = find(v);
        if (x == y) return;
        if (size_[x] < size_[y]) std::swap(x, y);
        history_size.emplace(size_[x], size_[x]);
        size_[x] = size_[x] + size_[y];
        history_fa.emplace(fa[y], fa[y]);
        fa[y] = x;
    }

    int history() {
        return history_fa.size();
    }

    void roll(int h) {
        while (history_fa.size() > h) {
            history_fa.top().first = history_fa.top().second;
            history_fa.pop();
            history_size.top().first = history_size.top().second;
            history_size.pop();
```

```
        }
    }
};
```

小波树

```
struct BitRank {
  // block 管理一行一行的bit
  std::vector<unsigned long long> block;
  std::vector<unsigned int> count;
  BitRank() {}
  // 位向量长度
  void resize(const unsigned int num) {
    block.resize(((num + 1) >> 6) + 1, 0);
    count.resize(block.size(), 0);
  }
  // 设置i位bit
  void set(const unsigned int i, const unsigned long long val) {
    block[i >> 6] |= (val << (i & 63));
  }
  void build() {
    for (unsigned int i = 1; i < block.size(); i++) {
      count[i] = count[i - 1] + __builtin_popcountll(block[i - 1]);
    }
  }
  // [0, i) 1的个数
  unsigned int rank1(const unsigned int i) const {
    return count[i >> 6] +
           __builtin_popcountll(block[i >> 6] & ((1ULL << (i & 63)) - 1ULL));
  }
  // [i, j) 1的个数
  unsigned int rank1(const unsigned int i, const unsigned int j) const {
    return rank1(j) - rank1(i);
  }
  // [0, i) 0的个数
  unsigned int rank0(const unsigned int i) const { return i - rank1(i); }
  // [i, j) 0的个数
  unsigned int rank0(const unsigned int i, const unsigned int j) const {
    return rank0(j) - rank0(i);
  }
};


class WaveletMatrix {
 private:
  unsigned int height;
  std::vector<BitRank> B;
  std::vector<int> pos;

 public:
  WaveletMatrix() {}
  WaveletMatrix(std::vector<int> vec)
      : WaveletMatrix(vec, *std::max_element(vec.begin(), vec.end()) + 1) {}
  // sigma: 字母表大小(字符串的话)，数字序列的话是数的种类
  WaveletMatrix(std::vector<int> vec, const unsigned int sigma) {
```

```
    init(vec, sigma);
  }
  void init(std::vector<int>& vec, const unsigned int sigma) {
    height = (sigma == 1) ? 1 : (64 - __builtin_clzll(sigma - 1));
    B.resize(height), pos.resize(height);
    for (unsigned int i = 0; i < height; ++i) {
      B[i].resize(vec.size());
      for (unsigned int j = 0; j < vec.size(); ++j) {
        B[i].set(j, get(vec[j], height - i - 1));
      }
      B[i].build();
      auto it = stable_partition(vec.begin(), vec.end(), [&](int c) {
        return !get(c, height - i - 1);
      });
      pos[i] = it - vec.begin();
    }
  }

  int get(const int val, const int i) { return val >> i & 1; }
  // [l, r) 中val出现的频率

  int rank(const int val, const int l, const int r) {
    return rank(val, r) - rank(val, l);
  }
  // [0, i) 中val出现的频率
  int rank(int val, int i) {
    int p = 0;
    for (unsigned int j = 0; j < height; ++j) {
      if (get(val, height - j - 1)) {
        p = pos[j] + B[j].rank1(p);
        i = pos[j] + B[j].rank1(i);
      } else {
        p = B[j].rank0(p);
        i = B[j].rank0(i);
      }
    }
    return i - p;
  }
  // [l, r) 中k小
  int quantile(int k, int l, int r) {
    int res = 0;
    for (unsigned int i = 0; i < height; ++i) {
      const int j = B[i].rank0(l, r);
      if (j > k) {
        l = B[i].rank0(l);
        r = B[i].rank0(r);
      } else {
        l = pos[i] + B[i].rank1(l);
        r = pos[i] + B[i].rank1(r);
        k -= j;
        res |= (1 << (height - i - 1));
      }
    }
    return res;
  }
  int rangefreq(const int i, const int j, const int a, const int b, const int l,
```

```
                const int r, const int x) {
    if (i == j || r <= a || b <= l) return 0;
    const int mid = (l + r) >> 1;
    if (a <= l && r <= b) {
      return j - i;
    } else {
      const int left =
          rangefreq(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1);
      const int right = rangefreq(pos[x] + B[x].rank1(i),
                                  pos[x] + B[x].rank1(j), a, b, mid, r, x + 1);
      return left + right;
    }
  }
  // [l,r) 在[a, b) 值域的数字个数
  int rangefreq(const int l, const int r, const int a, const int b) {
    return rangefreq(l, r, a, b, 0, 1 << height, 0);
  }
  int rangemin(const int i, const int j, const int a, const int b, const int l,
               const int r, const int x, const int val) {
    if (i == j || r <= a || b <= l) return -1;
    if (r - l == 1) return val;
    const int mid = (l + r) >> 1;
    const int res =
        rangemin(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1, val);
    if (res < 0)
      return rangemin(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(j), a, b, mid,
                      r, x + 1, val + (1 << (height - x - 1)));
    else
      return res;
  }
  // [l,r) 在[a,b) 值域内存在的最小值是什么，不存在返回-1
  int rangemin(int l, int r, int a, int b) {
    return rangemin(l, r, a, b, 0, 1 << height, 0, 0);
  }
};
```

two_sat

```
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    void notClause(int u, bool f, int v, bool g) {
        addClause(u, !f, v, !g) ;
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
```

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

小矩阵

```
struct Matrix {
    Z a[4];
    Matrix() :a{} {}
    friend Matrix operator*(const Matrix &lhs, const Matrix &rhs) {
        Matrix res;
        res.a[0] = (lhs.a[0] * rhs.a[0] + lhs.a[1] * rhs.a[2]) ;
        res.a[1] = (lhs.a[0] * rhs.a[1] + lhs.a[1] * rhs.a[3]) ;
        res.a[2] = (lhs.a[2] * rhs.a[0] + lhs.a[3] * rhs.a[2]) ;
        res.a[3] = (lhs.a[2] * rhs.a[1] + lhs.a[3] * rhs.a[3]) ;
        return res;
    }
};
struct Matrix {
    Z a[9];

    Matrix() : a{} {}

    friend Matrix operator*(const Matrix &lhs, const Matrix &rhs) {
        Matrix res;
        res.a[0] = lhs.a[0] * rhs.a[0] + lhs.a[1] * rhs.a[3] + lhs.a[2] *
rhs.a[6];
        res.a[1] = lhs.a[0] * rhs.a[1] + lhs.a[1] * rhs.a[4] + lhs.a[2] *
rhs.a[7];
```

```
        res.a[2] = lhs.a[0] * rhs.a[2] + lhs.a[1] * rhs.a[5] + lhs.a[2] *
rhs.a[8];

        res.a[3] = lhs.a[3] * rhs.a[0] + lhs.a[4] * rhs.a[3] + lhs.a[5] *
rhs.a[6];
        res.a[4] = lhs.a[3] * rhs.a[1] + lhs.a[4] * rhs.a[4] + lhs.a[5] *
rhs.a[7];
        res.a[5] = lhs.a[3] * rhs.a[2] + lhs.a[4] * rhs.a[5] + lhs.a[5] *
rhs.a[8];

        res.a[6] = lhs.a[6] * rhs.a[0] + lhs.a[7] * rhs.a[3] + lhs.a[8] *
rhs.a[6];
        res.a[7] = lhs.a[6] * rhs.a[1] + lhs.a[7] * rhs.a[4] + lhs.a[8] *
rhs.a[7];
        res.a[8] = lhs.a[6] * rhs.a[2] + lhs.a[7] * rhs.a[5] + lhs.a[8] *
rhs.a[8];

        return res;
    }
};
```

数学

取模类

```
using i64 = long long;
template<class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr i64 mul(i64 a, i64 b, i64 p) {
    i64 res = a * b - i64(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template<i64 P>
struct MLong {
    i64 x;
    constexpr MLong() : x{} {}
    constexpr MLong(i64 x) : x{norm(x % getMod())} {}

    static i64 Mod;
    constexpr static i64 getMod() {
```

```
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(i64 Mod_) {
        Mod = Mod_;
    }
    constexpr i64 norm(i64 x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr i64 val() const {
        return x;
    }
    explicit constexpr operator i64() const {
        return x;
    }
    constexpr MLong operator-() const {
        MLong res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MLong inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MLong &operator*=(MLong rhs) & {
        x = mul(x, rhs.x, getMod());
        return *this;
    }
    constexpr MLong &operator+=(MLong rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MLong &operator-=(MLong rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MLong &operator/=(MLong rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MLong operator*(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MLong operator+(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res += rhs;
```

```
        return res;
    }
    friend constexpr MLong operator-(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MLong operator/(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
        i64 v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MLong lhs, MLong rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
i64 MLong<0LL>::Mod = i64(1E18) + 9;

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
```

```
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    explicit constexpr operator i64() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
```

```
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 998244353;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 998244353;
using Z = MInt<P>;
```

多项式

```
std::vector<int> rev;
template<int P>
std::vector<MInt<P>> roots{0, 1};

template<int P>
constexpr MInt<P> findPrimitiveRoot() {
    MInt<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}

template<int P>
constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();

template<>
constexpr MInt<998244353> primitiveRoot<998244353> {31};

template<int P>
constexpr void dft(std::vector<MInt<P>> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
```

```
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k -
1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                MInt<P> u = a[i + j];
                MInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

template<int P>
constexpr void idft(std::vector<MInt<P>> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    MInt<P> inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}

template<int P = ::P>
struct Poly : public std::vector<MInt<P>> {
    using Value = MInt<P>;

    Poly() : std::vector<Value>() {}
    explicit constexpr Poly(int n) : std::vector<Value>(n) {}
```

```
    explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a)
{}
    constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a)
{}

    template<class InputIt, class = std::_RequireInputIter<InputIt>>
    explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>
(first, last) {}

    template<class F>
    explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
        for (int i = 0; i < n; i++) {
            (*this)[i] = f(i);
        }
    }

    constexpr Poly shift(int k) const {
        if (k >= 0) {
            auto b = *this;
            b.insert(b.begin(), k, 0);
            return b;
        } else if (this->size() <= -k) {
            return Poly();
        } else {
            return Poly(this->begin() + (-k), this->end());
        }
    }
    constexpr Poly trunc(int k) const {
        Poly f = *this;
        f.resize(k);
        return f;
    }
    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] += b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] -= b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a) {
        std::vector<Value> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
```

```
        }
        return Poly(res);
    }
    constexpr friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        if (a.size() < b.size()) {
            std::swap(a, b);
        }
        int n = 1, tot = a.size() + b.size() - 1;
        while (n < tot) {
            n *= 2;
        }
        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {

            Poly c(a.size() + b.size() - 1);
            for (int i = 0; i < a.size(); i++) {
                for (int j = 0; j < b.size(); j++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        a.resize(n);
        b.resize(n);
        dft(a);
        dft(b);
        for (int i = 0; i < n; ++i) {
            a[i] *= b[i];
        }
        idft(a);
        a.resize(tot);
        return a;
    }
    constexpr friend Poly operator*(Value a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    constexpr friend Poly operator*(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    constexpr friend Poly operator/(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] /= b;
        }
        return a;
    }
    constexpr Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
```

```
    constexpr Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    constexpr Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator*=(Value b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator/=(Value b) {
        return (*this) = (*this) / b;
    }
    template <class T>
    constexpr Value operator() ( T x ) {
        Value ans = 0 ;
        Value cnt = 1 ;
        for ( int i = 0 ; i < this->size () ; ++ i ) {
            ans += (* this) [ i ] * cnt ;
            cnt *= x ;
        }
        return ans ;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        assert (this->size() != 0) ;
        Poly res(this->size() - 1);
        for (int i = 0; i < this->size() - 1; ++i) {
            res[i] = (i + 1) * (*this)[i + 1];
        }
        return res;
    }
    constexpr Poly integr() const {
        Poly res(this->size() + 1);
        for (int i = 0; i < this->size(); ++i) {
            res[i + 1] = (*this)[i] / (i + 1);
        }
        return res;
    }
    constexpr Poly inv(int m) const {
        Poly x{(*this)[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly log(int m) const {
        return (deriv() * inv(m)).integr().trunc(m);
    }
    constexpr Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
```

```
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        Value v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
    }
    constexpr Poly pow(int k, int m, int k2) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        Value v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k2);
    }
    constexpr Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
        }
        return x.trunc(m);
    }
    constexpr Poly inv() const {
        return move (inv(this->size ())) ;
    }
    constexpr Poly log() const {
        return move(log(this->size ()));
    }
    constexpr Poly exp() const {
        return move(exp(this->size ()));
    }
    constexpr Poly pow(i64 b) const {
        Poly<> res (vector <Z> { 1 }) ;
        auto a = * this ;
        for (; b; b /= 2, a *= a) {
            if (b % 2) {
                res *= a;
            }
        }
        return res;
```

```
    }
    constexpr Poly sqrt() const {
        return move(sqrt(this->size()));
    }
    constexpr Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.begin(), b.end());
        return ((*this) * b).shift(-(n - 1));
    }
    constexpr std::vector<Value> eval(std::vector<Value> x) const {
        if (this->size() == 0) {
            return std::vector<Value>(x.size(), 0);
        }
        const int n = std::max(x.size(), this->size());
        std::vector<Poly> q(4 * n);
        std::vector<Value> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l,
int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                auto need = move(num.mulT(q[2 * p + 1]));
                need.resize ( m - l ) ;
                work(2 * p, l, m, need);
                need = move(num.mulT(q[2 * p]));
                need.resize ( r - m ) ;
                work(2 * p + 1, m, r, need);
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template<int P = ::P>
Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
```

```
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            auto d = oldC;
            d *= -1;
            d.insert(d.begin(), 1);
            MInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}


template<int P = ::P>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
```

```
        n /= 2;
    }
    return p[0] / q[0];
}
```

多项式扩展包

```
/**
 * 多项式扩展包
*/
namespace ExPoly {
    template<int P = ::P, class T1, class T2>
    constexpr static Poly <P> Lagrange(T1 x, T2 y) {
        int n = x.size();
        vector <Poly<>> M(4 * n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                M[p] = Poly{(int) -x[l], 1};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                M[p] = M[2 * p] * M[2 * p + 1];
            }
        };
        build(1, 0, n);
        auto M_ = M[1].deriv().eval(x);
        for (int i = 0; i < n; ++i) {
            M_[i] = y[i] * M_[i].inv();
        }
        vector <Poly<>> f(4 * n);
        std::function<void(int, int, int)> work = [&](int p, int l, int r) ->
void {
            if (r - l == 1) {
                if (l < n) {
                    f[p] = Poly{(int) M_[l]};
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m);
                work(2 * p + 1, m, r);
                f[p] = f[2 * p] * M[2 * p + 1] + f[2 * p + 1] * M[2 * p];
            }
        };
        work(1, 0, n);
        return f[1];
    }

/**
 *作用：对多项式进行平移操作
 *时间复杂度O(nlog(n))
 */
    template<int P = ::P>
    constexpr static Poly <P> Polynomial_translation(Poly <P> f, int k) {
        i64 n = (i64) f.size() - 1;
```

```
        Poly <P> g(n + 1);
        Z res = 1;
        for (int i = 0; i <= n; ++i) {
            g[n - i] = res * comb.invfac(i);
            res *= k;
            f[i] *= comb.fac(i);
        }
        Poly <P> here = g * f;
        here = here.shift(-n);
        for (int i = 0; i <= n; ++i) {
            here[i] *= comb.invfac(i);
        }
        return here;
    }

/**
 *作用：对相同的n对i \in ( 0 , n ) 求出将n个不同的元素划分为i个非空集的方案数
 *第二类Stirling数
 *时间复杂度O(nlog(n))
 */
    template<int P = ::P>
    constexpr static Poly <P> Second_Stirling_Same_N(int n) {
        Poly <P> f(n + 1), g(n + 1);
        for (int i = 0; i <= n; ++i) {
            g[i] = (i & 1 ? (Z) - 1 : Z(1)) * comb.invfac(i);
            f[i] = power((Z) i, n) * comb.invfac(i);
        }
        f *= g;
        f.resize(n + 1);
        return f;
    }

/**
 *作用：对相同的k对不同n 求出将n个不同的元素划分为k个非空集的方案数
 *第二类Stirling数
 *时间复杂度O(nlog(n))
 */
    template<int P = ::P>
    constexpr static Poly <P> Second_Stirling_Same_K(int Max_n, int k) {
        comb.init(Max_n + 1);
        Poly <P> f(vector<Z>(comb._invfac.begin(), comb._invfac.begin() + Max_n +
1));
        f[0] = 0;
        f = f.pow(k, Max_n + 1);
        for (int i = 0; i <= Max_n; ++i) {
            f[i] = f[i] * comb.fac(i) * comb.invfac(k);
        }
        return f;
    }

/**
 *作用：对相同的n对i \in ( 0 , n ) 求出将n个不同的元素划分为i个非空轮换的方案数
 *第一类Stirling数
 *时间复杂度O(nlog(n))
 */
    template<int P = ::P>
```

```
    constexpr static Poly <P> First_Stirling_Same_N(int n) {
        ll len = __lg(n);
        Poly <P> f = {1};
        ll cnt = 0;
        for (int i = len; i >= 0; --i) {
            f *= Polynomial_translation(f, cnt);
            cnt <<= 1;
            if (n >> i & 1) f *= Poly{cnt, 1}, cnt += 1;
        }
        return f;
    }

/**
 *作用：对相同的k对不同n 求出将n个不同的元素划分为k个非轮换的方案数
 *第一类Stirling数
 *时间复杂度O(nlog(n))
 */
    template<int P = ::P>
    constexpr static Poly <P> First_Stirling_Same_K(int Max_n, int k) {
        comb.init(Max_n + 1);
        Poly <P> f(comb._inv.begin(), comb._inv.begin() + Max_n + 1);
        f = f.pow(k, Max_n + 1);
        for (int i = 0; i <= Max_n; ++i) {
            f[i] *= comb.fac(i) * comb.invfac(k);
        }
        return f;
    }
};
```

矩阵

```
namespace matrix {
    using i64 = long long;

    template<typename T>
    struct Matrix : public std::vector<std::vector<T>> {
        using std::vector<std::vector<T>>::vector;

        Matrix(int x) : std::vector<std::vector<T>>(x, std::vector<T>(x)) {};
        Matrix(int x, int y) : std::vector<std::vector<T>>(x, std::vector<T>(y))
{};
        Matrix(int x, int y, T c) : std::vector<std::vector<T>>(x, std::vector<T>
(y, c)) {};

        constexpr Matrix operator+(Matrix a);
        constexpr Matrix operator-(Matrix a);
        constexpr Matrix operator*(Matrix a);

        template <typename T1, typename T2>
        friend constexpr Matrix<T1> operator*(Matrix<T1> x, T2 a);

        constexpr Matrix& operator+=(Matrix a);
        constexpr Matrix& operator-=(Matrix a);
        constexpr Matrix& operator*=(Matrix a);
```

```
    template <typename T1, typename T2>
    friend constexpr Matrix<T1>& operator*=(Matrix<T1>& x, T2 a);

    constexpr Matrix pow(i64 b);
    constexpr Matrix Transpose();
    constexpr Matrix inv();
};

template <typename T>
constexpr Matrix<T> Matrix<T>::operator+(Matrix<T> a) {
    auto it = *this;
    int n = (int)a.size();
    int m = (int)a.back().size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            it[i][j] += a[i][j];
    return it;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::operator-(Matrix<T> a) {
    auto it = *this;
    int n = (int)a.size();
    int m = (int)a.back().size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            it[i][j] -= a[i][j];
    return it;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::operator*(Matrix<T> a) {
    int n = (int)this->size();
    int mid = (int)a.size();
    int m = (int)a.back().size();
    Matrix<T> it(n, m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            for (int k = 0; k < mid; ++k)
                it[i][j] += (*this)[i][k] * a[k][j];
    return it;
}

template <typename T1, typename T2>
constexpr Matrix<T1> operator*(Matrix<T1> x, T2 a) {
    int n = (int)x.size();
    int m = (int)x.back().size();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            x[i][j] *= a;
    return x;
}

template <typename T>
constexpr Matrix<T>& Matrix<T>::operator+=(Matrix<T> a) {
    return *this = *this + a;
```

```
}

template <typename T>
constexpr Matrix<T>& Matrix<T>::operator-=(Matrix<T> a) {
    return *this = *this - a;
}

template <typename T>
constexpr Matrix<T>& Matrix<T>::operator*=(Matrix<T> a) {
    return *this = *this * a;
}

template <typename T1, typename T2>
constexpr Matrix<T1>& operator*=(Matrix<T1>& x, T2 a) {
    return x = x * a;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::pow(i64 b) {
    auto res = Matrix(this->size(), this->size());
    for (int i = 0; i < (int)this->size(); ++i)
        res[i][i] = 1;
    auto a = *this;
    for (; b; b /= 2, a *= a)
        if (b % 2) res *= a;
    return res;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::Transpose() {
    int n = this->back().size(), m = this->size();
    auto it = Matrix(n, m);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            it[i][j] = (*this)[j][i];
    return it;
}

template <typename T>
constexpr Matrix<T> Matrix<T>::inv() {
    int n = this->size();
    Matrix<T> it(n, 2 * n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            it[i][j] = (*this)[i][j];
    for (int i = 0; i < n; ++i)
        it[i][i + n] = 1;
    for (int i = 0; i < n; ++i) {
        int r = i;
        for (int k = i; k < n; ++k)
            if ((i64)it[k][i]) { r = k; break; }
        if (r != i)
            swap(it[r], it[i]);
        if (!(i64)it[i][i])
            return Matrix<T>();
```

```
            T x = (T) 1 / it[i][i];
            for (int k = 0; k < n; ++k) {
                if (k == i)
                    continue;
                T t = it[k][i] * x;
                for (int j = i; j < 2 * n; ++j)
                    it[k][j] -= t * it[i][j];
            }
            for (int j = 0; j < 2 * n; ++j)
                it[i][j] *= x;
        }
        Matrix<T> ans(n, n);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                ans[i][j] = it[i][j + n];
        return ans;
    }
    // namespace Matrix
};
```

数学类

```
/**
 * 数学工具箱
*/

namespace Math {
    using i64 = long long;
    using Int = __int128;
    using ui64 = unsigned long long;
    std::mt19937
rng(std::chrono::system_clock::now().time_since_epoch().count());

    struct math {

/**
 * @brief   带模乘
 * @return  (a ^ b)% m
*/
        i64 static mul(i64 a, i64 b, i64 m);


/**
 * @brief   快速幂
*/
        template<class T>
        constexpr static T power(T a, i64 b);

        i64 static power(i64 a, i64 b, i64 m);

/**
 * @brief   求和
*/
        template<typename T>
```

```
        constexpr static T __sum1(T it);

        template<typename T>
        constexpr static T __sum2(T it);

/**
 * 欧几里得算法相关
*/

/**
 * 算法：扩展欧几里得算法
 * 作用：求解 ax + by = gcd ( a , b )
 * 返回：gcd,x,y
 */
        template<typename T = i64>
        constexpr array<T, 3> static Exgcd(T a, T b);

/**
 * 算法：扩展欧几里得算法
 * 作用：求解 ax + by = res
 * 限制：gcd(a, b) | res
 */
        template<typename T = i64>
        constexpr array<T, 3> static __Exgcd(T a, T b, T res);

/**
 * 算法：线性同余方程
 * 作用：求解 ax == b ( mod n )
 *       的最小整数解
 * 要求：gcd ( a , n ) | b
 */
        template<typename T = i64>
        constexpr T static Linear_congruence_equation(i64 a, i64 b, i64 mod);

/**
 * 算法：扩展欧几里得算法求逆元
 * 作用：求解 ax == 1 ( mod n )的最小整数解
 * 要求：a 与 n 互质
 */
        template<typename T = i64>
        constexpr T static inv(i64 a, i64 mod);

/**
 * 扩展欧几里得结束
*/

/**
 * 算法：Miller_Rabin_Test
 * 作用：在long long范围内快速判断质数
 * 时间复杂度：O(log^3(n))
 */
        constexpr static bool Miller_Rabin_Test(i64 n);

/**
 * 算法：Pollard_Rho
 * 作用：能快速找到大整数的一个非1、非自身的因子的算法
```

```
 * 时间复杂度：O(n^{1/4}log(n))
 */
        static i64 Pollard_Rho(i64 N);

/**
 * 算法：使用Pollard_Rho进行质因数分解
 * 返回：顺序所有质因子(重复)
*/
        std::vector <i64> static factorize(i64 n);

/**
 * 算法：中国剩余定理
 * 作用：求解一元线性同余方程（ x == a （ mod m ））在模n（所有的模积）的解
 * 限制：所有模互质
 */
        template<typename T = i64>
        constexpr static T Chinese_remainder_theorem
                (vector <i64> &a, vector <i64> &m);

/**
 * 算法：扩展中国剩余定理
 * 作用：求解一元线性同余方程（ x == a （ mod m ））在模n（所有模的最小公倍数）的解
 * 无限制：所有模互质
*/
        template<typename T = i64>
        constexpr static T Extend_the_Chinese_remainder_theorem
                (vector <i64> &a, vector <i64> &m);

/**
 * 算法：欧拉函数
 * 作用：求欧拉函数
 * 时间复杂度：O(sqrt ( n ))
 */
        template<typename T = i64>
        constexpr static T Euler_phi(T n);

/**
 * 算法：扩展欧拉定理（欧拉降幂）
 * 作用：大指数快速幂
 * 时间复杂度：O(sqrt ( m ))
 */
        static i64 Extending_Euler_theorem(i64 a, string b, i64 m);


/**
 * 算法：求最小原根
 * 要求：请自行保证这个数有原根(2,4,p^q,2*p^q)
 * 时间复杂度：O(sqrt(n))
*/
        static i64 min_primitive_root(i64 m);

/**
 * 求一个数的所有原根
 * 注意提前使用质数筛，名称为s，开到n，并筛出欧拉函数
 * 需要Linear_sieves_max、s
 * 时间复杂度：O(sqrt ( m ))
```

```
 */
# ifdef _Linear_sieves
        std::vector <i64> static primitive_root(i64 n);
# endif


/**
 * 算法：扩展BSGS
 * 作用：求解 a ^ x = b ( mod m )
 * 无要求: a与m互质
 * 返回：问题的最小非负x，无解返回-1
 * 建议使用自定义Hash
*/
        constexpr i64 static exBSGS(i64 a, i64 b, i64 m, i64 k = 1);

/**
 * 算法：n次剩余
 * 作用：求解 x ^ a = b ( mod m )
 * 要求: m是质数
 * 返回：x，无解返回-1e15
 * 建议使用自定义Hash
*/
        static std::vector <i64> n_times_remaining(i64 a, i64 b, i64 m);


/**
 * 算法：扩展lucas
 * 作用：在p为非质数情况下，大数组合数C(n,m)
 * 必要情况下，预处理降低复杂度
*/

        static i64 Exlucas(i64 n, i64 m, i64 P);

        //struct math
    };


    i64 math::mul(i64 a, i64 b, i64 m) {
        return static_cast<__int128>(a) * b % m;
    }


    template<class T>
    constexpr T math::power(T a, i64 b) {
        T res = 1;
        for (; b; b /= 2, a *= a)
            if (b % 2) res *= a;
        return res;
    }

    i64 math::power(i64 a, i64 b, i64 m) {
        i64 res = 1 % m;
        for (; b; b >>= 1, a = mul(a, a, m))
            if (b & 1)
                res = mul(res, a, m);
        return res;
```

```
    }


    template<typename T>
    constexpr T math::__sum1(T it) { return (it * (it + 1)) / ((T) 2); }

    template<typename T>
    constexpr T math::__sum2(T it) { return it * (it + 1) * (2 * it + 1) / ((T)
6); }


    template<typename T>
    constexpr array<T, 3> math::Exgcd(T a, T b) {
        T x1 = 1, x2 = 0, x3 = 0, x4 = 1;
        while (b != 0) {
            T c = a / b;
            std::tie(x1, x2, x3, x4, a, b) =
                    std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b *
c);
        }
        return {a, x1, x2}; //x = x1, y = x2;
    }

    template<typename T>
    constexpr array<T, 3> math::__Exgcd(T a, T b, T res) {
        assert(res % __gcd(a, b) == 0);
        auto [gcd, x, y] = Exgcd(a, b);
        return {gcd, res / gcd * x, res / gcd * y};
    }

    template<typename T>
    constexpr T math::Linear_congruence_equation(i64 a, i64 b, i64 mod) {
        auto [gcd, x, k] = __Exgcd<T>((T) a, (T) mod, (T) b);
        T t = mod / gcd;
        return (x % t + t) % t;
    }

    template<typename T>
    constexpr T math::inv(i64 a, i64 mod) {
        auto [gcd, x, k] = Exgcd<T>((T) a, (T) mod);
        return (x % mod + mod) % mod;
    }


    constexpr bool math::Miller_Rabin_Test(i64 n) {
        if (n < 3 || n % 2 == 0) return n == 2;//特判
        i64 u = n - 1, t = 0;
        while (u % 2 == 0) u /= 2, ++t;
        constexpr std::array<i64, 7> ud = {2, 325, 9375, 28178, 450775, 9780504,
1795265022};
        for (i64 a: ud) {
            i64 v = power(a, u, n);
            if (v == 1 || v == n - 1 || v == 0) continue;
            for (int j = 1; j <= t; j++) {
                v = mul(v, v, n);
                if (v == n - 1 && j != t) {
```

```
                    v = 1;
                    break;
                }//出现一个n-1，后面都是1，直接跳出
                if (v == 1) return 0;//这里代表前面没有出现n-1这个解，二次检验失败
            }
            if (v != 1) return 0;//Fermat检验
        }
        return 1;
    }

    i64 math::Pollard_Rho(i64 N) {
        if (N == 4) // 特判4
            return 2;
        if (Miller_Rabin_Test(N)) // 特判质数
            return N;
        auto randint = [&](i64 l, i64 r) {
            return l + rng() % (r - l + 1);
        };
        while (true) {
            i64 c = randint(1, N - 1); // 生成随机的c
            auto f = [=](i64 x) { return ((Int) x * x + c) % N; }; // Int表示
__int128，防溢出
            i64 t = f(0), r = f(f(0));
            while (t != r) {
                i64 d = gcd(abs(t - r), N);
                if (d > 1)
                    return d;
                t = f(t), r = f(f(r));
            }
        }
    }

    std::vector <i64> math::factorize(i64 n) {
        std::vector <i64> p;
        std::function<void(i64)> f = [&](i64 n) {
            if (n <= 10000) {
                for (int i = 2; i * i <= n; ++i)
                    for (; n % i == 0; n /= i)
                        p.push_back(i);
                if (n > 1)
                    p.push_back(n);
                return;
            }
            if (Miller_Rabin_Test(n)) {
                p.push_back(n);
                return;
            }
            auto g = [&](i64 x) {
                return (mul(x, x, n) + 1) % n;
            };
            i64 x0 = 2;
            while (true) {
                i64 x = x0;
                i64 y = x0;
                i64 d = 1;
                i64 power = 1, lam = 0;
```

```
                i64 v = 1;
                while (d == 1) {
                    y = g(y);
                    ++lam;
                    v = mul(v, std::abs(x - y), n);
                    if (lam % 127 == 0) {
                        d = std::gcd(v, n);
                        v = 1;
                    }
                    if (power == lam) {
                        x = y;
                        power *= 2;
                        lam = 0;
                        d = std::gcd(v, n);
                        v = 1;
                    }
                }
                if (d != n) {
                    f(d);
                    f(n / d);
                    return;
                }
                ++x0;
            }
        };
        f(n);
        std::sort(p.begin(), p.end());
        return p;
    }

    template<typename T>
    constexpr T math::Chinese_remainder_theorem
            (vector <i64> &a, vector <i64> &m) {
        T n = accumulate(m.begin(), m.end(), (T) 1, multiplies<T>()), ans = 0;

        for (int i = 0; i < (i64) a.size(); ++i) {
            T m1 = n / m[i], b;
            b = inv(m1, m[i]);
            ans = (ans + a[i] * m1 * b % n) % n;
        }
        return (ans % n + n) % n;
    }

    template<typename T>
    constexpr T math::Extend_the_Chinese_remainder_theorem
            (vector <i64> &a, vector <i64> &m) {
        T m1 = m[0], a1 = a[0];
        for (int i = 1; i < (i64) a.size(); ++i) {
            T m2 = m[i], a2 = a[i];
            auto [gcd, p, q] = __Exgcd(m1, m2, a2 - a1);
            a1 = m1 * p + a1;
            m1 = m1 * m2 / gcd;
            a1 = (a1 % m1 + m1) % m1;
        }
        return a1;
    }
```

```cpp
template<typename T>
constexpr T math::Euler_phi(T n) {
    T ans = n;
    for (i64 i = 2; i * i <= n; i++)
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}

i64 math::Extending_Euler_theorem(i64 a, string b, i64 m) {
    i64 gcd = __gcd(a, m);
    i64 phi = Euler_phi(m);
    i64 res = 0;
    bool flag = 0;
    for (auto u: b) {
        res = res * 10 + u - '0';
        while (res >= phi) {
            res -= phi;
            if (!flag) flag = 1;
        }
    }
    if (gcd != 1 && flag) res += phi;
    return power(a, res, m);
}

i64 math::min_primitive_root(i64 m) {
    i64 phi = math::Euler_phi(m);
    auto div = [&](i64 x) {
        vector <i64> f;
        for (i64 i = 2; i * i <= x; ++i) {
            if (x % i != 0) continue;
            f.push_back(i);
            while (x % i == 0) x /= i;
        }
        if (x != 1 && x != phi) f.push_back(x);
        return f;
    };
    auto d = div(phi);
    i64 root = -1;
    auto check = [&](i64 x) {
        for (auto u: d)
            if (math::power(x, u, m) == 1)
                return false;
        root = x;
        return true;
    };
    for (i64 i = 1;; ++i) {
        if (__gcd(i, m) != 1)
            continue;
        if (check(i)) break;
    }
    return root;
```

```
    }

# ifdef _Linear_sieves
    std::vector <i64> math::primitive_root(i64 n) {
        static vector<bool> exist(Linear_sieves_max + 1);
        auto __exist = [&]() {
            static bool __existed = 0;
            if (__existed) return;
            __existed = 1;
            exist[2] = 1;
            exist[4] = 1;
            for (ll p: s.Prime) {
                if ((p & 1) == 0) continue;
                for (ll now = p; now <= (ll) exist.size() - 1; now *= p) {
                    exist[now] = 1;
                    if (now * 2 <= (ll) exist.size() - 1)
                        exist[now * 2] = 1;
                }
            }
        };
        __exist();
        if (!exist[n]) return vector<i64>();
        vector <ll> f;
        ll phi = s.eu(n);
        ll pphi = s.eu(phi);
        ll m = phi;
        for (int i = 2; i * i <= m; ++i) {
            if (m % i == 0) {
                f.push_back(i);
                while (m % i)
                    m /= i;
            }
        }
        if (m != 1) f.push_back(m);
        // Debug ( f ) ;
        ll root = -1;
        auto check = [&](ll x) {
            for (auto u: f)
                if (power(x, phi / u, n) == 1)
                    return false;
            root = x;
            return true;
        };
        for (i64 i = 1;; ++i) {
            if (__gcd(i, n) != 1) continue;
            if (check(i)) break;
        }
        vector <ll> ans;
        for (i64 now = root, i = 1; i <= phi; ++i) {
            if (__gcd(phi, i) == 1)
                ans.push_back(now);
            now = (now * root) % n;
        }
        sort(ans.begin(), ans.end());
        return ans;
    }
```

```
# endif

    constexpr i64 math::exBSGS(i64 a, i64 b, i64 m, i64 k) {
        constexpr i64 inf = 1e15;
        auto BSGS = [&](i64 a, i64 b, i64 m, i64 k = 1) {
# ifdef _Hash
            unordered_map <ui64, ui64, Hash> map;
# else
            std::map <ui64, ui64> map;
# endif
            i64 cur = 1, t = sqrt(m) + 1;
            for (i64 B = 1; B <= t; ++B) {
                (cur *= a) %= m;
                map[b * cur % m] = B;
            }
            ll now = cur * k % m;
            for (i64 A = 1; A <= t; ++A) {
                auto it = map.find(now);
                if (it != map.end())
                    return A * t - (i64) it->second;
                (now *= cur) %= m;
            }
            return -inf; // 无解
        };
        i64 A = a %= m, B = b %= m, M = m;
        if (b == 1) return 0;
        i64 cur = 1 % m;
        for (int i = 0;; i++) {
            if (cur == B) return i;
            cur = cur * A % M;
            i64 d = __gcd(a, m);
            if (b % d) return -inf;
            if (d == 1) {
                auto ans = BSGS(a, b, m, k * a % m);
                if (ans == -inf) return -1;
                else return ans + i + 1;
            }
            k = k * a / d % m, b /= d, m /= d;
        }
    }

    std::vector <i64> math::n_times_remaining(i64 a, i64 b, i64 m) {
        auto root = min_primitive_root(m);
        i64 now = math::power(root, a, m);
        i64 c = math::exBSGS(now, b, m);
        if (c == -1) return vector<i64>();
        i64 x0 = math::power(root, c, m);
        i64 phi = math::Euler_phi(m);
        i64 gcd = __gcd(a, phi);
        vector <i64> ans;
        i64 cnt = math::power(root, phi / gcd, m);
        for (int i = 0; i < gcd; ++i) {
            ans.push_back(x0);
            x0 = math::mul(x0, cnt, m);
        }
```

```
        return ans;
    }

    i64 math::Exlucas(i64 n, i64 m, i64 P) {
        std::vector <i64> p, a;
        function <i64(i64, i64, i64)> calc = [&](i64 n, i64 x, i64 P) mutable ->
i64 {
            if (!n) return 1;
            i64 s = 1;
            for (i64 i = 1; i <= P; ++i)  //求阶乘，可预处理降低复杂度
                if (i % x != 0) s = math::mul(s, i, P);
            s = math::power(s, n / P, P);
            for (i64 i = n / P * P + 1; i <= n; ++i)
                if (i % x != 0) s = math::mul(i, s, P);
            return math::mul(s, calc(n / x, x, P), P);
        };
        function <i64(i64, i64, i64, i64)> multilucas = [&](i64 n, i64 m, i64 x,
i64 P) -> i64 {
            i64 cnt = 0;
            for (i64 i = n; i != 0; i /= x) cnt += i / x;
            for (i64 i = m; i != 0; i /= x) cnt -= i / x;
            for (i64 i = n - m; i != 0; i /= x) cnt -= i / x;
            return static_cast<__int128>(1) * math::power(x, cnt, P) % P *
calc(n, x, P) % P
                    * math::inv(calc(m, x, P), P) % P * math::inv(calc(n - m, x,
P), P) % P;
        };
        for (i64 i = 2; i * i <= P; ++i) {
            if (P % i == 0) {
                p.emplace_back(1);
                while (P % i == 0) p.back() *= i, P /= i;
                a.emplace_back(multilucas(n, m, i, p.back()));
            }
        }
        if (P > 1) p.emplace_back(P), a.emplace_back(multilucas(n, m, P, P));
        return math::Extend_the_Chinese_remainder_theorem(a, p);
    }
    // namespace Math
}

using namespace Math;
```

线性基

```
struct Linear_Base {
    int siz;
    vector<int> a;
    Linear_Base(int _siz) {
        siz = _siz;
        a.resize(siz + 1);
    }
    void insert(int x) {//插入
        for (int i = siz; i >= 0; i--) if (x & (1ll << i)) {
            if (!a[i]) { a[i] = x; return; }
            else x ^= a[i];
```

```
        }
    }
    bool check(int x) {//查询x是否能被异或出来
        for (int i = siz; i >= 0; i--) if (x & (1ll << i)) {
            if (!a[i]) break;
            x ^= a[i];
        }
        return x == 0;
    }
    int querymax(int res) {//查询最大异或和
        for (int i = siz; i >= 0; i--) if ((res ^ a[i]) > res) res ^= a[i];
        return res;
    }
    int querymin(int res) {//查询最小
        for (int i = siz; i >= 0; i--) if (res & (1ll << i)) res ^= a[i];
        return res;
    }
    int querykth(int k) {//查询第k大的异或和
        vector<int> tmp(siz + 10);
        int res = 0, cnt = 0;
        for (int i = 0; i <= siz; i++) {
            for (int j = i - 1; j >= 0; j--) if (a[i] & (1ll << j)) a[i] ^= a[j];
            if(a[i]) tmp[cnt++] = a[i];
        }
        for (int i = 0; i < cnt; i++) if (k & (1ll << i)) res ^= tmp[i];
        return res;
    }
    void merge(const Linear_Base& other)//合并
    {
        for (int i = 0; i <= siz; i++) insert(other.a[i]);
    }
};
```

线性筛

```
struct Linear_sieves {
# define _Linear_sieves
    int n;
    vector<int> Prime, Euler, Morbius, Approximate, Approximate_cnt;
    vector<bool> notprime;
    vector<array<ll, 2>> div;

    Linear_sieves() {};

    Linear_sieves(int _n) { init(_n); };

    void init(int _n) {
        n = _n;
        Prime_work();
    }

    void Prime_work() {
        notprime.assign(n + 1, 0);
        notprime[0] = 1;
```

```
        notprime[1] = 1;
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) {
                Prime.push_back(i);
            }
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                notprime[i * Prime[j]] = 1;

                if (i % Prime[j] == 0) break;
            }
        }
    }

    void Euler_work() {
        Euler.assign(n + 1, 0);
        Euler[1] = 1;
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) Euler[i] = i - 1;
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                i64 now = i * Prime[j];
                if (i % Prime[j] != 0) {
                    Euler[now] = (Prime[j] - 1) * Euler[i];
                } else {
                    Euler[now] = Prime[j] * Euler[i];
                    break;
                }
            }
        }
    }

    void Morbius_work() {
        Morbius.assign(n + 1, 0);
        Morbius[1] = 1;
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) Morbius[i] = -1;
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                i64 now = i * Prime[j];
                if (i % Prime[j] != 0) {
                    Morbius[now] = -Morbius[i];
                } else break;
            }
        }
    }

    void Div_work() {
        div.resize(n + 1);
        div[0] = {1, 1};
        div[1] = {1, 1};
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) {
                div[i] = {1, i};
            }
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                div[i * Prime[j]] = {Prime[j], i};
                if (i % Prime[j] == 0) break;
            }
```

```
        }
    }

/**
 * 求约数个数
*/
    void Approximate_work() {
        Approximate.assign(n + 1, 0);
        Approximate_cnt.assign(n + 1, 0);
        Approximate[1] = 1;
        Approximate_cnt[1] = 0;
        for (i64 i = 2; i <= n; ++i) {
            if (notprime[i] == 0) {
                Approximate[i] = 2;
                Approximate_cnt[i] = 1;
            }
            for (i64 j = 0; i * Prime[j] <= n; ++j) {
                i64 now = i * Prime[j];
                if (i % Prime[j] != 0) {
                    Approximate_cnt[now] = 1;
                    Approximate[now] = Approximate[i] * 2;
                } else {
                    Approximate_cnt[now] = Approximate_cnt[i] + 1;
                    Approximate[now] = Approximate[i] / Approximate_cnt[now] *
(Approximate_cnt[now] + 1);
                    break;
                }
            }
        }
    }

    std::vector<i64> get_frac(i64 x) {
        vector<i64> f;
        for (; x > 1; f.push_back(div[x][0]), x = div[x][1]);
        return f;
    }

    i64 size() { return (ll) Prime.size(); }

    bool isprime(int n) { return !notprime[n]; }

    i64 eu(int n) { return Euler[n]; }

    i64 mo(int n) { return Morbius[n]; }
};
```

组合数学

```
template<class T>
struct Comb {
    int n;
    std::vector <T> _fac;
    std::vector <T> _invfac;
    std::vector <T> _inv;
```

```
    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}

    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = std::min(m, T::getMod() - 1);
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    T fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }

    T invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }

    T inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }

    T binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }

/**
 * 第二类斯特林数
 * 时间复杂度 : O (m * log (m))
*/
    T Stirling(int n, int m) {
        T ans = 0;
        for (int i = 0; i <= m; ++i) {
            ans += (((m - i) & 1) == 1 ? -1 : 1) * power((T) i, n) * invfac(i) *
invfac(m - i);
        }
        return ans;
    }
```

```
    T Catalan(int n) {
        return binom(2 * n, n) * inv(n + 1);
    }

/**
 * 算法：卢卡斯定理
 * 作用：大数组合数
 * 注意在p较小时使用p
 * p为Z的质数
 * 时间复杂度为O(logp)
*/
    T lucas(i64 n, i64 m) {
        if (m == 0) return T(1);
        return binom(n % T::getMod(), m % T::getMod()) * lucas(n / T::getMod(), m
/ T::getMod());
    }
};

Comb<Z> comb;
```

图论

SCC

一般

```
struct SCC {
    int n, cnt = 0, tot = -1;
    vector<vector<int>> map;
    vector<int> d, id, stack, tag;
    vector<bool> instack;

    SCC(int n): n(n), map(n), d(n, -1), id(n), tag(n, -1), instack(n, 0) {}

private:
    void _scc(int now) {
        d[now] = id[now] = ++tot;
        stack.push_back(now);
        instack[now] = 1;
        for (auto u : map[now]) {
            if (!~d[u]) {
                _scc(u);
                id[now] = min(id[now], id[u]);
            } else if (instack[u]) {
                id[now] = min(id[now], id[u]);
            }
        }
        if (d[now] == id[now]) {
            ++cnt;
            do {
                instack[stack.back()] = 0;
                tag[stack.back()] = cnt;
                stack.pop_back();
```

```
            } while (instack[now]);
        }
    }

public:
    void addedge(int u, int v) {
        map[u].push_back(v);
    }
    void scc(int now) {
        --cnt;
        _scc(now);
        ++cnt;
    }
};
```

割边

```
struct CutEdge {
    int n, tot = -1;
    vector<pair<int, int>> edge;
    vector<vector<int>> map;
    vector<int> d, id, ans;

    CutEdge(int n) :n(n), d(n, -1), id(n, -1), map(n) {};

private :
    void _cutedge(int now, int _edge) {
        d[now] = id[now] = ++tot;
        for (auto tag: map[now]) {
            auto &here = edge[tag].second;
            if (!~d[here]) {
                _cutedge(here, tag);
                id[now] = min(id[now], id[here]);
                if (id[here] > d[now]) {
                    ans.push_back(tag);
                }
            } else if (tag != (_edge ^ 1)) {
                id[now] = min(id[here], id[now]);
            }
        }
    }

public:
    void addedge(int u, int v) {
        edge.push_back({u, v});
        map[u].push_back(int(edge.size()) - 1);
    }

    void cutedge(int u, int _edge) {
        _cutedge(u, _edge);
    }
};
```

割点

```
struct CutPoint {
    int n, tot = -1, root = -1;
    vector<vector<int>> map;
    vector<int> d, id;
    vector<bool> iscutpoint;

    CutPoint(int n): n(n), map(n), d(n, -1), id(n, -1), iscutpoint(n, 0) {};

private:
    void _cutpoint(int now) {
        d[now] = id[now] = ++tot;
        int child = 0;
        for (auto u: map[now]) {
            if (!~d[u]) {
                _cutpoint(u);
                id[now] = min(id[now], id[u]);
                if (id[u] >= d[now]) {
                    ++child;
                    if (now != root || child >= 2) {
                        iscutpoint[now] = 1;
                    }
                }
            } else id[now] = min(d[u], id[now]);
        }
    }

public:
    void cutpoint(int now, int root) {
        this->root = root;
        _cutpoint(now);
        this->root = -1;
    }

};


struct CutPoint {
    int n, tot = -1, root = -1;
    vector<vector<int>> map;
    vector<int> d, id, stack;
    vector<bool> iscutpoint, instack;
    vector<vector<int>> ans;

    CutPoint(int n) : n(n), map(n), d(n, -1), id(n, -1), iscutpoint(n, 0),
instack(n, 0) {};

private:
    void _cutpoint(int now) {
        d[now] = id[now] = ++tot;
        stack.push_back(now);
        instack[now] = 1;
        int child = 0;
        for (auto u: map[now]) {
```

```
            if (!~d[u]) {
                _cutpoint(u);
                id[now] = min(id[now], id[u]);
                if (id[u] >= d[now]) {
                    ++child;
                    if (now != root || child >= 2) {
                        iscutpoint[now] = 1;
                        ans.push_back(vector<int>(0));
                        auto &bk = ans.back();
                        while (instack[u]) {
                            bk.push_back(stack.back());
                            instack[stack.back()] = 0;
                            stack.pop_back();
                        }
                        bk.push_back(now);
                    }
                }
            } else id[now] = min(d[u], id[now]);
            if (now == root && child) {
                ans.push_back(vector<int>(0));
                auto &bk = ans.back();
                while (instack[now]) {
                    bk.push_back(stack.back());
                    instack[stack.back()] = 0;
                    stack.pop_back();
                }
            }
        }
    }

public:
    void cutpoint(int now, int root) {
        this->root = root;
        _cutpoint(now);
        this->root = -1;
    }
};
```

Lca、dfn、虚树

```
template<class T,
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
```

```
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
```

```
        }
    }
};

struct DFN {
    int n;
    vector<int> dfn, dep, sz, fa;
    RMQ<array<int, 2>> rmq;
    DFN() = default;
    template<class T>
    DFN(const std::vector<std::vector<T>> &adj, T root = 0) {
        init(adj, root);
    }
    template<class T>
    void init(const std::vector<std::vector<T>> &adj, T root = 0) {
        n = adj.size();
        dfn.assign(n, 0);
        dep.assign(n, 0);
        sz.assign(n, 0);
        fa.assign(n, 0);
        virtual_tree.assign(n, {});
        vector<array<int, 2>> inrmq(n);
        int tot = 0;
        auto &pa = fa;
        auto dfs = [&] (auto&&dfs, int now, int fa) -> void {
            dfn[now] = tot ++;
            dep[now] = dep[fa] + 1;
            pa[now] = fa;
            for (auto here : adj[now]) {
                if (here == fa) continue;
                dfs(dfs, here, now);
                sz[now] += sz[here];
            }
            sz[now] += 1;
        };
        dfs(dfs, root, root);
        for (int i = 0; i < n; i += 1) {
            inrmq[dfn[i]] = {dep[i], i};
        }
        rmq.init(inrmq);
    }
    int lca (int lhs, int rhs) {
        if (lhs == rhs) return lhs;
        if (dfn[lhs] > dfn[rhs]) swap(lhs, rhs);
        return fa[rmq(dfn[lhs] + 1, dfn[rhs] + 1)[1]];
    }
    std::vector<std::vector<int>> virtual_tree;
    std::vector<int> real_key;
    template<class T>
    std::vector<std::vector<int>> &build_virtual_tree(std::vector<T> key) {
        for (auto u : real_key) {
            virtual_tree[u].clear();
        }
        real_key.clear();
        sort(key.begin(), key.end(), [&] (T x, T y) {return dfn[x] < dfn[y];});
        for (int i = 0; i < int(key.size()) - 1; i += 1) {
```

```
            real_key.push_back(key[i]);
            real_key.push_back(lca(key[i], key[i + 1]));
        }
        real_key.push_back(key.back());
        sort(real_key.begin(), real_key.end(), [&] (T x, T y) {return dfn[x] <
dfn[y];});
        real_key.erase(std::unique(real_key.begin(), real_key.end()),
real_key.end());
        for (int i = 0; i < int(real_key.size()) - 1; i += 1 ){
            int Lca = lca(real_key[i], real_key[i + 1]);
            virtual_tree[Lca].push_back(real_key[i + 1]);
            virtual_tree[real_key[i + 1]].push_back(Lca);
        }
        return virtual_tree;
    }
};
```

重链剖分

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

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
```

```
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

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

    int rootedParent(int u, int v) {
```

```
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
        }
        auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x,
int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

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

流

网络流

```
constexpr int inf = 1E9;
template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

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

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

    void init(int n) {
```

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

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

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

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

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

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

费用流

```
template < typename T >
struct Min_Cost_Flow {
    using i64 = int64_t ;
    struct info { T v , f , c ; info ( T v , T f , T c ): v ( v ) , f ( f ) , c (
c ) {}};

    i64 n ;
    vector < info > e ;
    vector < vector < T > > g ;
    std::vector < i64 > dis ,h ;
    std::vector < T > pre ;

    Min_Cost_Flow ( i64 n ): n ( n ) , g ( n ) {}
    void add ( T u , T v , T f , T c ) {
        if ( c < 0 ) {
```

```
            g [ u ].push_back ( e.size () ) ;
            e.emplace_back ( v , 0 , c ) ;
            g [ v ].push_back ( e.size () ) ;
            e.emplace_back ( u , f , -c ) ;
        } else {
            g [ u ].push_back ( e.size () ) ;
            e.emplace_back ( v , f , c ) ;
            g [ v ].push_back ( e.size () ) ;
            e.emplace_back ( u , 0 , -c ) ;
        }
    }
    bool dijkstra ( i64 s , i64 t ) {
        dis.assign ( n , std::numeric_limits < i64 >::max () ) ;
        pre.assign ( n , -1 ) ;
        priority_queue < pair < i64 , i64 > , std::vector < pair < i64 , i64 > >
                , std::greater < pair < i64 , i64 > > > que ;
        dis [ s ] = 0 ;
        que.emplace ( 0 , s ) ;
        while ( !que.empty () ) {
            auto [ d , u ] = que.top () ;
            que.pop () ;
            if ( dis [ u ] < d ) continue ;
            for ( i64 i : g [ u ] ) {
                auto [ v , f , c ] = e [ i ] ;
                if ( f > 0 && dis [ v ] > d + h [ u ] - h [ v ] + c ) {
                    dis [ v ] = d + h [ u ] - h [ v ] + c ;
                    pre [ v ] = i ;
                    que.emplace ( dis [ v ] , v ) ;
                }
            }
        }
        return dis [ t ] != std::numeric_limits < i64 >::max () ;
    }

    std::pair < i64 , i64 > flow ( i64 s , i64 t ) {
        int flow = 0 ;
        i64 cost = 0 ;
        h.assign ( n , 0 ) ;
        while ( dijkstra ( s , t ) ) {
            for ( int i = 0 ; i < n ; ++ i ) h [ i ] += dis [ i ] ;
            i64 aug = std::numeric_limits < i64 >::max () ;
            for ( int i = t ; i != s ; i = e [ pre [ i ] ^ 1 ].v )
                aug = std::min ( aug , (i64)e [ pre [ i ] ].f ) ;
            for ( int i = t ; i != s ; i = e [ pre [ i ] ^ 1 ].v ) {
                e [ pre [ i ] ].f -= aug ;
                e [ pre [ i ] ^ 1 ].f += aug ;
            }
            flow += aug ;
            cost += h [ t ] * aug ;
        }
        return std::make_pair ( flow , cost ) ;
    }

    struct Edge {
        int from;
        int to;
```

```
        T cap;
        T flow;
    };

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

笛卡尔树

```
template<class T>
struct Descartes {
    int n;
    vector <T> v;
    vector<int> ls, rs;

    Descartes(int n) : ls(n, -1), rs(n, -1), v(n) {}

    Descartes(vector <T> &v) : n((ll) v.size()), ls(n, -1), rs(n, -1), v(v) {}

    int build() /* return root */ {
        vector<int> s(n);
        int top = 0;
        int root = -1;
        for (int i = 0; i < n; ++i) {
            int realtop = top;
            while (top != 0 && v[s[top]] > v[i]) { --top; }
            if (top < realtop) ls[i] = s[top + 1];
            if (top != 0) rs[s[top]] = i;
            s[++top] = i;
        }
        root = s[1];
        assert(!s.empty());
        return root;
    }
};
```

杂

初始

```
# include <bits/stdc++.h>
using namespace std;
# ifdef LOCAL
    # include "C:\Users\Kevin\Desktop\demo\save\debug.h"
# else
# define debug(...) 114514
# define ps 114514
# endif

using ll = long long;
using i64 = long long;

void solve() {

}

int  main () {
# ifndef cin
    std::ios::sync_with_stdio (false);
    std::cin.tie (nullptr) ;
# endif
    // __fin("C:\\Users\\Kevin\\Desktop\\cpp\\in.in") ;

    i64 _ = 1 ;
    // cin >> _ ;
    while (_ --) {
        solve ();
# ifdef LOCAL
# ifndef cin
    cout.flush();
# endif
# endif
    }
    return 0 ;
}
```

对拍

- 一共 4 个文件:
    - `baoli.cpp`
    - `std.cpp`
    - `data.cpp`
        - 关键

```
std::mt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

auto rnd = (i64 l, i64 r) [&] {
    uniform_int_distribution<long long> _rnd(l, r);
    return _rnd(rng);
};

shuffle(v.begin(), v.end(), rng);
```

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

using ll = long long;
using i64 = long long;

int  main () {
# ifndef cin
    std::ios::sync_with_stdio (false);
    std::cin.tie (nullptr) ;
# endif
    std::mt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());
    auto rnd = (i64 l, i64 r) [&] {
        uniform_int_distribution<long long> _rnd(l, r);
        return _rnd(rng);
    };
    vector<int> a;
    shuffle(a.begin(), a.end(), rng);

    return 0 ;
}
```

- 对拍.cpp
  - ```
    #include <iostream>
    #include <cstdio>
    #include <windows.h>
    #include <cstdlib>
    #include <ctime>

    using namespace std;

    int main() {
        int ok = 0;
        int n = 50;
        for (int i = 1;; ++i) {
            system("data.exe > in.txt");
            system("std.exe < in.txt > std.txt");
            double begin = clock();
            system("baoli.exe < in.txt > baoli.txt");
            double end = clock();

            double t = (end - begin);
            if (system("fc std.txt baoli.txt")) {
    ```
    ```

```
            printf("测试点#%d Wrong Answer\n", i);
            system("pause");
        } else if (t > 2000) {
            printf("测试点#%d Time Limited Exceeded 用时 %.0lfms\n",
i, t);
        } else {
            printf("测试点#%d Accepted 用时%.0lfms\n", i, t);
            ok++; //AC数量+1
        }
    }
    printf("\n");
    double res = 100.0 * ok / n;
    printf("共 %d 组测试数据，AC数据 %d 组。 得分%.1lf。", n, ok, res);
}
```

简易版取模类

```
template<typename T>
T power(T x, long long b) {
    T res = 1;
    while (b) {
        if (b & 1) res *= x;
        x *= x;
        b >>= 1;
    }
    return res;
}

template<int P>
struct mod_int {
    int x;
    static int mod;
    mod_int() : x{} {}
    mod_int(long long x) : x(norm(x % getMod())) {}

    int norm(int x) {
        if (x >= P) x -= P;
        if (x < 0) x += P;
        return x;
    }

    static void setMod(int x) {
        mod = x;
    }
    static int getMod() {
        return (P > 0 ? P : mod);
    }
    mod_int operator-() {
        return -x;
    }

    mod_int &operator+=(mod_int rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
```

```
    mod_int &operator-=(mod_int rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    mod_int &operator*=(mod_int rhs) {
        x = 1ll * x * rhs.x % getMod();
        return *this;
    }

    mod_int inv() {
        return power(*this, P - 2);
    }
    mod_int &operator/=(mod_int rhs) {
        x = 1ll * x * rhs.inv().x % getMod();
        return *this;
    }

    friend mod_int operator+(mod_int lhs, mod_int rhs) {
        return lhs += rhs;
    }
    friend mod_int operator-(mod_int lhs, mod_int rhs) {
        return lhs -= rhs;
    }
    friend mod_int operator*(mod_int lhs, mod_int rhs) {
        return lhs *= rhs;
    }
    friend mod_int operator/(mod_int lhs, mod_int rhs) {
        return lhs /= rhs;
    }
    friend bool operator==(mod_int lhs, mod_int rhs) {
        return lhs.x == rhs.x;
    }
    friend bool operator!=(mod_int lhs, mod_int rhs) {
        return lhs.x != rhs.x;
    }

    template<class istream>
    friend istream &operator>>(istream &input, mod_int &rhs) {
        long long x;
        input >> x;
        rhs = x;
        return input;
    }
    template<class ostream>
    friend ostream &operator<<(ostream &output, mod_int rhs) {
        return output << rhs.x;
    }
};

template<>
int mod_int<0>::mod = 998244353;

constexpr int P = 1e9 + 7;
using Z = mod_int<P>;
```

debug.h

```
template<typename A, typename B> ostream& operator<<(ostream &os, const pair<A,
B> &p) { return os << '(' << p.first << ", " << p.second << ')'; }
template<typename T_container, typename T = typename
enable_if<!is_same<T_container, string>::value, typename
T_container::value_type>::type> ostream& operator<<(ostream &os, const
T_container &v) { os << '{'; string sep; for (const T &x : v) os << sep << x, sep
= ", "; return os << '}'; }

void debug_out() { cerr << endl; }
template<typename Head, typename... Tail> void debug_out(Head H, Tail... T) {
cerr << ' ' << H; debug_out(T...); }

# define ps cerr << "YES" << endl
# define debug(...) cerr << "(" << #__VA_ARGS__ << "):" << endl,
debug_out(__VA_ARGS__)
```

hash

```
struct Hash {
# define _Hash
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }

  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }

  // 针对 std::pair<int, int> 作为主键类型的哈希函数
  size_t operator()(pair<uint64_t, uint64_t> x) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x.first + FIXED_RANDOM) ^
           (splitmix64(x.second + FIXED_RANDOM) >> 1);
  }
};
```

O2优化

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,fma")
#pragma GCC optimize("unroll-loops")
```

快读

```
struct Input {
    using Long = long long ;
    # define cin input
    Input () {}
    # define MAXSIZE (1 << 20)
    # define isdigit(x) (x >= '0' && x <= '9')

    char buf[MAXSIZE], *p1 = buf, *p2 = buf;
    #define gc()
\
        (p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin), p1 == p2)
\
            ? EOF
\
            : *p1++)

    bool blank(char ch) {
        return ch == ' ' || ch == '\n'
            || ch == '\r' || ch == '\t' || ch == EOF;
    }
    template <typename T>
    Input &operator>> (T &x) {
        x = 0 ;
        bool sign = 0;
        char ch = gc();
        for (; !isdigit(ch); ch = gc())
            if (ch == '-') sign = 1;
        for (; isdigit(ch); ch = gc())
            x = (x << 3) + (x << 1) + ch - '0';
        if (sign)x = -x;
        return *this ;
    }
    Input &operator>> (char &x) {
        x = ' ';
        for (; blank(x); x = gc ()) ;
        return *this ;
    }
    Input &operator>> (double &x) {
        x = 0 ;
        double tmp = 1;
        bool sign = 0;
        char ch = gc();
        for (; !isdigit(ch); ch = gc())
            if (ch == '-') sign = 1;
        for (; isdigit(ch); ch = gc())
            x = x * 10 + ch - '0';
        if (ch == '.')
        for (ch = gc(); isdigit(ch); ch = gc())
            tmp /= 10.0, x += tmp * (ch - '0');
        if (sign) x = -x;
        return *this ;
    }
    Input &operator>>(string &s) {
```

```
        s.clear() ;
        char ch = gc();
        for (; blank(ch); ch = gc());
        for (; !blank(ch); ch = gc()) {
            s += ch ;
        }
        return *this ;
    }
}input;
struct __setprecision {
    int precision ;
};
__setprecision setprecision (int x) { return __setprecision {x};}
struct Output {
# define MAXSIZE (1 << 20)
# define cout output
char pbuf[MAXSIZE], *pp = pbuf ;
    void push(const char &c) {
        // putchar (c);
        if (pp - pbuf == MAXSIZE)
            fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
    }
    int precision ;
    Output () { precision = 6 ;}
    ~Output () { fwrite(pbuf, 1, pp - pbuf, stdout);}
    template<class T>
    Output &operator<< (const T &x) {
        char stack[35]; int top = 0 ;
        T tmp = x ;
        bool _ = tmp < 0 ;
        if (_) tmp *= -1;
        while (tmp) stack[++ top] = '0' + tmp % 10, tmp /= 10;
        if (_) stack[++ top] = '-';
        while (top) push (stack [top]), -- top;
        if (x == 0)push ('0');
        return *this ;
    }
    Output &operator<< (const string &x) {
        for (auto&u : x) push (u);
        return *this;
    }
    template<size_t N>
    Output &operator<< (const char(&x)[N]) {
        *this << string (x) ;
        return *this;
    }
    Output &operator<< (const char &x) {
        push (x);
        return *this;
    }
    Output &operator<< (const bool &x) {
        push (x ? '1' : '0');
        return *this;
    }
    Output &operator<<(const double &x) {
```

```
        int intPart = static_cast<int>(x);
        *this << intPart; // Output the integer part

        push('.'); // Decimal point

        double decimalPart = x - intPart;
        for (int i = 0; i < precision; ++i) {
            decimalPart *= 10;
            int digit = static_cast<int>(decimalPart);
            *this << char('0' + digit);
            decimalPart -= digit;
        }
        return *this;
    }
    Output &operator<< (const __setprecision &x) {
        precision = x.precision ;
        return *this ;
    }
}output;
```

字符串

Ac自动机

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        int top;
        int val;
        int d;
        std::array<int, ALPHABET> next;
        Node() : len{}, link{}, next{}, top{}, val {-1}, d{} {}
    };

    std::vector<Node> t;

    AhoCorasick() {
        init();
    }

    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }

    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    int add(const std::vector<int> &a) {
```

```
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        apply (t[p].val);
        return p;
    }

    int add(const std::string &a, char offset = 'a') {
        std::vector<int> b(a.size());
        for (int i = 0; i < a.size(); i++) {
            b[i] = a[i] - offset;
        }
        return add(b);
    }

    void work() {
        std::queue<int> q;
        q.push(1);

        while (!q.empty()) {
            int x = q.front();
            q.pop();

            t[x].top = t[link(x)].val >= 0 ? link(x) : top(link(x));

            for (int i = 0; i < ALPHABET; i++) {
                if (t[x].next[i] == 0) {
                    t[x].next[i] = t[t[x].link].next[i];
                } else {
                    t[t[x].next[i]].link = t[t[x].link].next[i];
                    t[t[t[x].link].next[i]].d += 1;
                    q.push(t[x].next[i]);
                }
            }
        }
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
```

```
    }

    int& val(int p) {
        return t[p].val;
    }

    int top (int p) {
        return t[p].top;
    }

    int size() {
        return t.size();
    }

    int& d ( int p ) {
        return t[p].d;
    }

    void apply (auto& val) {
        val = 0 ;
    }
};
```

字符串哈希

```
std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());

bool isprime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
            return false;
    return true;
}
int findPrime(int n) {
    while (!isprime(n))
        n++;
    return n;
}

template<int N>
struct StringHash {
    static array<int, N> mod;
    static array<int, N> base;
    vector<array<int, N>> p, h;
    StringHash() = default;
    StringHash(const string& s) {
        int n = s.size();
        p.resize(n);
        h.resize(n);
        fill(p[0].begin(), p[0].end(), 1);
        for (int i = 0; i < n; i++)
        for (int j = 0; j < N; j++) {
            p[i][j] = 1ll * (i == 0 ? 1ll : p[i - 1][j]) * base[j] % mod[j];
```

```
            h[i][j] = (1ll * (i == 0 ? 0ll : h[i - 1][j]) * base[j] + s[i]) %
mod[j];
        }
    }
    array<int, N> query(int l, int r) {
        assert(r >= l - 1);
        array<int, N> ans{};
        if (l > r) return {0, 0};
        for (int i = 0; i < N; i++) {
            ans[i] = (h[r][i] - 1ll * (l == 0 ? 0ll : h[l - 1][i]) * (r - l + 1
== 0 ? 1ll : p[r - l][i]) % mod[i] + mod[i]) % mod[i];
        }
        return ans;
    }
};

constexpr int HN = 2;
template<>
array<int, 2> StringHash<HN>::mod =
    {findPrime(rng() % 900000000 + 100000000),
     findPrime(rng() % 900000000 + 100000000)};
template<>
array<int, 2> StringHash<HN>::base {13331, 131};
using Hashing = StringHash<HN>;
```

后缀数组

```
using i64 = long long;
struct SuffixArray {
    int n;
    std::vector<int> sa, rk, lc;
    SuffixArray(const std::string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(), [&](int a, int b) {return s[a] < s[b];});
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
```

```
            cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; --i)
            sa[--cnt[rk[tmp[i]]]] = tmp[i];
        std::swap(rk, tmp);
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i
- 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
        k *= 2;
    }
    for (int i = 0, j = 0; i < n; ++i) {
        if (rk[i] == 0) {
            j = 0;
        } else {
            for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j]
== s[sa[rk[i] - 1] + j]; )
                ++j;
            lc[rk[i] - 1] = j;
        }
    }
}
};
```

KMP

```
struct KMP{
    int n;
    std::vector<int> pi;
    std::vector<vector<int>> aut;

    KMP(const std::string &s) {
        n = (int)s.length();
        prefix_function(s);
        compute_automaton(s);
    }

    void prefix_function(string s) {
        pi.resize(n);
        for (int i = 1; i < n; i++) {
            int j = pi[i - 1];
            while (j > 0 && s[i] != s[j]) j = pi[j - 1];
            if (s[i] == s[j]) j++;
            pi[i] = j;
        }
    }

    void compute_automaton(string s) {
        aut.resize(n, vector<int>(26));
        for (int i = 0; i < n; i++) {
            for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i - 1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
```

```
            }
        }
    }
};
```

Trie

```
constexpr int N = 2e5;

struct node{
    int _ch[2];
    int R = -1, L = -1;
    node *ch(int x) const;
    void _new(int x);
    bool is_null();
    bool not_null();
};
node B[N * 30];
int idx = 1;
node *node::ch(int x) const {
    return B + _ch[x];
}
void node::_new(int x) {
    _ch[x] = ++ idx;
}
bool node::is_null() {
    return this == B;
}
bool node::not_null() {
    return this != B;
}
```