Fenwick

```cpp
int lg(unsigned int x) {
    return std::bit_width(x) - 1;
}

template<typename T = int>
class Fenwick {
private:
    int n;
    std::vector<T> tr;
    struct Proxy {
        Fenwick<T>& fen{};
        int idx{};
        constexpr Proxy& operator+=(const T& v) {
            fen.add(idx, v);
            return *this;
        }
        constexpr Proxy& operator-=(const T& v) {
            fen.add(idx, -v);
            return *this;
        }
        constexpr Proxy& operator++() {
            fen.add(idx, 1);
            return *this;
        }
        constexpr Proxy& operator++(int) {
            fen.add(idx, 1);
            return *this;
        }
        constexpr Proxy& operator--() {
            fen.add(idx, -1);
            return *this;
        }
        constexpr Proxy& operator--(int) {
            fen.add(idx, -1);
            return *this;
        }
    };
public:
    explicit Fenwick(int n = 0, const T& init_ = T()) {
        init(n);
        for (int i = 0; i < n; i++) {
            add(i, init_);
        }
    }
    explicit Fenwick(const std::vector<T>& init_) {
        init(init_);
    }
    void init(int n_) {
        n = n_;
        tr.assign(n, {});
```

```
    }
    void init(const std::vector<T>& init_) {
        init(init_.size());
        for (int i = 0; i < n; i++) {
            add(i, init_[i]);
        }
    }
    void add(int x, const T& v) {
        for (++x; x <= n; x += x & -x) {
            tr[x - 1] += v;
        }
    }
    T sum(int x) {
        T ans = T();
        for (; x; x -= x & -x) {
            ans += tr[x - 1];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }
    int find(const T& k) {
        int x = 0;
        T cur{};
        for (int i = 1 << lg(n); i; i /= 2) {
            if (x + i <= n && cur + tr[x + i - 1] <= k) {
                x += i;
                cur = cur + tr[x - 1];
            }
        }
        return x;
    }
    constexpr Proxy operator[](int i) {
        return Proxy{ *this, i };
    }
    constexpr T operator() (int r) {
        return sum(r);
    }
    constexpr T operator() (int l, int r) {
        return rangeSum(l, r);
    }
};
```

LazySegmentTree

```cpp
int lg(unsigned int x) {
    return std::bit_width(x) - 1;
}

template <class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> tr;
    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();
        tr.assign(4 << lg(n), Info());
        tag.assign(4 << lg(n), Tag());
        auto build = [&](auto&& self, int p, int l, int r) {
            if (r - l == 1) {
                tr[p] = init_[l];
                return;
            }
            int m = std::midpoint(l, r);
            self(self, 2 * p, l, m);
            self(self, 2 * p + 1, m, r);
            pull(p);
        };
        build(build, 1, 0, n);
    }

    void pull(int p) {
        tr[p] = tr[2 * p] + tr[2 * p + 1];
    }

    void apply(int p, const Tag& t) {
        tr[p].apply(t);
        tag[p].apply(t);
    }

    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) {
            tr[p] = v;
            return;
        }
        int m = std::midpoint(l, r);
        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 x, const Info& v) {
        modify(1, 0, n, x, 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 tr[p];
        }
        int m = std::midpoint(l, r);
        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& t) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            return apply(p, t);
        }
        int m = std::midpoint(l, r);
        push(p);
        rangeApply(2 * p, l, m, x, y, t);
        rangeApply(2 * p + 1, m, r, x, y, t);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag& t) {
        return rangeApply(1, 0, n, l, r, t);
    }

    int findFirst(int p, int l, int r, int x, int y, auto&& pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(tr[p])) {
```

```
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = std::midpoint(l, r);
        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;
    }
    int findFirst(int l, int r, auto&& pred) {
        return findFirst(1, 0, n, l, r, pred);
    }

    int findLast(int p, int l, int r, int x, int y, auto&& pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(tr[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = std::midpoint(l, r);
        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;
    }
    int findLast(int l, int r, auto&& pred) {
        return findLast(1, 0, n, l, r, pred);
    }

    struct Proxy {
        LazySegmentTree& seg;
        int i, j;
        Info val;
        Proxy(LazySegmentTree& seg, int i, int j) :seg(seg), i(i), j(j), val(seg.rangeQuery(i,
j)) {}
        constexpr Info* operator->() {
            return &val;
        }
        constexpr Proxy& operator=(const Info& info) {
            assert(i + 1 == j);
            seg.modify(i, info);
            return *this;
        }
        constexpr Proxy& operator+=(const Tag& tag) {
            seg.rangeApply(i, j, tag);
            return *this;
        }
    };
```

```
    constexpr Proxy operator[](int i) {
        return Proxy(*this, i, i + 1);
    }
    constexpr Proxy operator()(int i, int j) {
        return Proxy(*this, i, j);
    }
};
```

StringHash

```cpp
constexpr 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;
}

constexpr int findPrime(int n) {
    while (!isprime(n)) {
        n++;
    }
    return n;
}

std::mt19937 rnd(std::chrono::high_resolution_clock::now().time_since_epoch().count());
const int P = findPrime(rnd() % 900000000 + 100000000);
const int B = rnd() % P;

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

struct Z {
    i64 x;
    constexpr Z() : x{ 0 } {}
    constexpr Z(i64 x) : x{ norm(x % P) } {}

    constexpr i64 norm(i64 x) const {
        if (x < 0) {
            x += P;
        }
        if (x >= P) {
            x -= P;
        }
        return x;
    }
    constexpr i64 val() const {
        return x;
```

```
    }
    constexpr Z operator-() const {
        Z res;
        res.x = norm(P - x);
        return res;
    }
    constexpr Z inv() const {
        return power(*this, P - 2);
    }
    constexpr Z& operator*=(Z rhs)& {
        x = x * rhs.x % P;
        return *this;
    }
    constexpr Z& operator+=(Z rhs)& {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr Z& operator-=(Z rhs)& {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr Z& operator/=(Z rhs)& {
        return *this *= rhs.inv();
    }
    friend constexpr Z operator*(Z lhs, Z rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr Z operator+(Z lhs, Z rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr Z operator-(Z lhs, Z rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr Z operator/(Z lhs, Z rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr bool operator==(Z lhs, Z rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(Z lhs, Z rhs) {
        return lhs.val() != rhs.val();
    }
    friend constexpr bool operator<(Z lhs, Z rhs) {
        return lhs.val() < rhs.val();
    }
};
const Z invB = Z(B).inv();

namespace coef {
    int n;
```

```
    std::vector<Z> _p{ 1 }, _q{ 1 };
    void init(int m) {
        if (m <= n) {
            return;
        }
        _p.resize(m + 1);
        _q.resize(m + 1);
        for (int i = n + 1; i <= m; i++) {
            _p[i] = _p[i - 1] * B;
            _q[i] = _q[i - 1] * invB;
        }
        n = m;
    }
    Z q(int m);
    Z p(int m) {
        if (m < 0) {
            return q(-m);
        }
        if (m > n) {
            init(2 * m);
        }
        return _p[m];
    }
    Z q(int m) {
        if (m < 0) {
            return p(-m);
        }
        if (m > n) {
            init(2 * m);
        }
        return _q[m];
    }
};

struct Hash {
    Z x;
    int siz;
    Hash(Z x = 0, int siz = 0) : x(x), siz(siz) {}
    Z val() const {
        return x.val();
    }
    constexpr friend Hash operator+(const Hash& a, const Hash& b) {
        return Hash(a.val() + b.val() * coef::p(a.siz), a.siz + b.siz);
    }
    constexpr friend Hash operator-(const Hash& a, const Hash& b) {
        assert(a.siz >= b.siz);
        return Hash((a.val() - b.val()) * coef::q(b.siz), a.siz - b.siz);
    }
    constexpr friend bool operator==(const Hash& a, const Hash& b) {
        return a.val() == b.val();
    }
    constexpr friend bool operator!=(const Hash& a, const Hash& b) {
        return a.val() != b.val();
    }
};

struct StringHash { // a0 * b ^ 0 + a1 * b ^ 1 + ... + ai * B ^ i + ... + an * B ^ n;
    int n;
```

```
    std::vector<Z> h, r;
    StringHash() { n = 0; h.push_back(0), r.push_back(0); }
    StringHash(const char* s) {
        init(std::string_view(s));
    }
    StringHash(std::string_view s) {
        init(s);
    }
    void init(std::string_view s) {
        n = s.size();
        h.assign(n + 1, 0);
        r.assign(n + 1, 0);
        for (int i = 0; i < n; i++) {
            h[i + 1] = h[i] + coef::p(i) * s[i];
            r[i + 1] = r[i] + coef::p(i) * s[n - 1 - i];
        }
    }
    void add(char c) { // r is not available from now
        h.push_back(h.back() + coef::p(n++) * c);
    }
    Hash get(int l, int r) {
        return Hash((h[r] - h[l]) * coef::q(l), r - l);
    }
    bool same(int x, int y, int l, int r) {
        return get(x, y) == get(l, r);
    }
    bool isPalindrom(int x, int y) { // only available when not add
        return (r[n - x] - r[n - y]) * coef::q(n - y) == get(x, y).val();
    }
    constexpr Hash operator()(int l, int r) {
        return get(l, r);
    }
    constexpr bool operator()(int a, int b, int c, int d) {
        return same(a, b, c, d);
    }
};
```

DSU

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

    DSU() :n(0) {}
    explicit DSU(int n) :n(n) {
        init(n);
    }

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

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

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

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

    std::vector<std::vector<int>> groups() {
        std::vector<int> p(n), psize(n);
        for (int i = 0; i < n; i++) {
            p[i] = find(i);
            psize[p[i]]++;
        }
        std::vector<std::vector<int>> ans(n);
```

```
        for (int i = 0; i < n; i++) {
            ans[i].reserve(psize[i]);
        }
        for (int i = 0; i < n; i++) {
            ans[p[i]].push_back(i);
        }
        ans.erase(std::remove_if(ans.begin(), ans.end(), [](const std::vector<int>& v) {
            return v.empty();
        }), ans.end());
        return ans;
    }

    constexpr int operator[](int i) {
        return find(i);
    }
};
```

莫队

```
constexpr int M = 500;

constexpr int bel(int i) {
    return i / M;
}

std::sort(qry.begin(), qry.end(), [&](const auto& a, const auto& b) {
    if (a[0] / M != b[0] / M) return a[0] < b[0];
    return a[0] / M % 2 ? a[1] > b[1] : a[1] < b[1];
});

int l = 1, r = 0, res = 0;
std::vector<int> ans(m);

auto add = [&](int i) {

};
auto del = [&](int i) {

};

for (auto [x, y, i] : qry) {
    while (l < x) del(l++);
    while (l > x) add(--l);
    while (r < y) add(++r);
    while (r > y) del(r--);
    ans[i] = res;
}
```

String

```
namespace internal {
    std::vector<int> sa_naive(const std::vector<int>& s) {
        int n = int(s.size());
        std::vector<int> sa(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(), [&](int l, int r) {
            if (l == r) return false;
            while (l < n && r < n) {
                if (s[l] != s[r]) return s[l] < s[r];
                l++;
                r++;
            }
            return l == n;
        });
        return sa;
    }

    std::vector<int> sa_doubling(const std::vector<int>& s) {
        int n = int(s.size());
        std::vector<int> sa(n), rnk = s, tmp(n);
        std::iota(sa.begin(), sa.end(), 0);
        for (int k = 1; k < n; k *= 2) {
            auto cmp = [&](int x, int y) {
                if (rnk[x] != rnk[y]) return rnk[x] < rnk[y];
                int rx = x + k < n ? rnk[x + k] : -1;
                int ry = y + k < n ? rnk[y + k] : -1;
                return rx < ry;
            };
            std::sort(sa.begin(), sa.end(), cmp);
            tmp[sa[0]] = 0;
            for (int i = 1; i < n; i++) {
                tmp[sa[i]] = tmp[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 : 0);
            }
            std::swap(tmp, rnk);
        }
        return sa;
    }

    // SA-IS, linear-time suffix array construction
    // Reference:
    // G. Nong, S. Zhang, and W. H. Chan,
    // Two Efficient Algorithms for Linear Time Suffix Array Construction
    template <int THRESHOLD_NAIVE = 10, int THRESHOLD_DOUBLING = 40>
    std::vector<int> sa_is(const std::vector<int>& s, int upper) {
        int n = int(s.size());
        if (n == 0) return {};
        if (n == 1) return { 0 };
        if (n == 2) {
```

```
    if (s[0] < s[1]) {
        return { 0, 1 };
    } else {
        return { 1, 0 };
    }
}
if (n < THRESHOLD_NAIVE) {
    return sa_naive(s);
}
if (n < THRESHOLD_DOUBLING) {
    return sa_doubling(s);
}

std::vector<int> sa(n);
std::vector<bool> ls(n);
for (int i = n - 2; i >= 0; i--) {
    ls[i] = (s[i] == s[i + 1]) ? ls[i + 1] : (s[i] < s[i + 1]);
}
std::vector<int> sum_l(upper + 1), sum_s(upper + 1);
for (int i = 0; i < n; i++) {
    if (!ls[i]) {
        sum_s[s[i]]++;
    } else {
        sum_l[s[i] + 1]++;
    }
}
for (int i = 0; i <= upper; i++) {
    sum_s[i] += sum_l[i];
    if (i < upper) sum_l[i + 1] += sum_s[i];
}

auto induce = [&](const std::vector<int>& lms) {
    std::fill(sa.begin(), sa.end(), -1);
    std::vector<int> buf(upper + 1);
    std::copy(sum_s.begin(), sum_s.end(), buf.begin());
    for (auto d : lms) {
        if (d == n) continue;
        sa[buf[s[d]]++] = d;
    }
    std::copy(sum_l.begin(), sum_l.end(), buf.begin());
    sa[buf[s[n - 1]]++] = n - 1;
    for (int i = 0; i < n; i++) {
        int v = sa[i];
        if (v >= 1 && !ls[v - 1]) {
            sa[buf[s[v - 1]]++] = v - 1;
        }
    }
    std::copy(sum_l.begin(), sum_l.end(), buf.begin());
    for (int i = n - 1; i >= 0; i--) {
        int v = sa[i];
        if (v >= 1 && ls[v - 1]) {
            sa[--buf[s[v - 1] + 1]] = v - 1;
        }
    }
};

std::vector<int> lms_map(n + 1, -1);
int m = 0;
```

```
        for (int i = 1; i < n; i++) {
            if (!ls[i - 1] && ls[i]) {
                lms_map[i] = m++;
            }
        }
        std::vector<int> lms;
        lms.reserve(m);
        for (int i = 1; i < n; i++) {
            if (!ls[i - 1] && ls[i]) {
                lms.push_back(i);
            }
        }

        induce(lms);

        if (m) {
            std::vector<int> sorted_lms;
            sorted_lms.reserve(m);
            for (int v : sa) {
                if (lms_map[v] != -1) sorted_lms.push_back(v);
            }
            std::vector<int> rec_s(m);
            int rec_upper = 0;
            rec_s[lms_map[sorted_lms[0]]] = 0;
            for (int i = 1; i < m; i++) {
                int l = sorted_lms[i - 1], r = sorted_lms[i];
                int end_l = (lms_map[l] + 1 < m) ? lms[lms_map[l] + 1] : n;
                int end_r = (lms_map[r] + 1 < m) ? lms[lms_map[r] + 1] : n;
                bool same = true;
                if (end_l - l != end_r - r) {
                    same = false;
                } else {
                    while (l < end_l) {
                        if (s[l] != s[r]) {
                            break;
                        }
                        l++;
                        r++;
                    }
                    if (l == n || s[l] != s[r]) same = false;
                }
                if (!same) rec_upper++;
                rec_s[lms_map[sorted_lms[i]]] = rec_upper;
            }

            auto rec_sa =
                sa_is<THRESHOLD_NAIVE, THRESHOLD_DOUBLING>(rec_s, rec_upper);

            for (int i = 0; i < m; i++) {
                sorted_lms[i] = lms[rec_sa[i]];
            }
            induce(sorted_lms);
        }
        return sa;
    }

}  // namespace internal
```

```cpp
std::vector<int> suffix_array(const std::vector<int>& s, int upper) {
    assert(0 <= upper);
    for (int d : s) {
        assert(0 <= d && d <= upper);
    }
    auto sa = internal::sa_is(s, upper);
    return sa;
}

template <class T> std::vector<int> suffix_array(const std::vector<T>& s) {
    int n = int(s.size());
    std::vector<int> idx(n);
    std::iota(idx.begin(), idx.end(), 0);
    std::sort(idx.begin(), idx.end(), [&](int l, int r) { return s[l] < s[r]; });
    std::vector<int> s2(n);
    int now = 0;
    for (int i = 0; i < n; i++) {
        if (i && s[idx[i - 1]] != s[idx[i]]) now++;
        s2[idx[i]] = now;
    }
    return internal::sa_is(s2, now);
}

std::vector<int> suffix_array(const std::string& s) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return internal::sa_is(s2, 255);
}
std::vector<int> suffix_array(const char* s) {
    return suffix_array(std::string(s));
}

// Reference:
// T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park,
// Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its
// Applications
template <class T>
std::vector<int> lcp_array(const std::vector<T>& s,
                           const std::vector<int>& sa) {
    int n = int(s.size());
    assert(n >= 1);
    std::vector<int> rnk(n);
    for (int i = 0; i < n; i++) {
        rnk[sa[i]] = i;
    }
    std::vector<int> lcp(n - 1);
    int h = 0;
    for (int i = 0; i < n; i++) {
        if (h > 0) h--;
        if (rnk[i] == 0) continue;
        int j = sa[rnk[i] - 1];
        for (; j + h < n && i + h < n; h++) {
            if (s[j + h] != s[i + h]) break;
        }
        lcp[rnk[i] - 1] = h;
```

```
    }
    return lcp;
}

std::vector<int> lcp_array(const std::string& s, const std::vector<int>& sa) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return lcp_array(s2, sa);
}

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[std::countr_zero(stk[r - 1] >> (l - x)) + l];
        }
    }
};

// Reference:
// D. Gusfield,
// Algorithms on Strings, Trees, and Sequences: Computer Science and
// Computational Biology
template <class T> std::vector<int> Zfunction(const std::vector<T>& s) {
    int n = int(s.size());
    if (n == 0) return {};
    std::vector<int> z(n);
    z[0] = 0;
    for (int i = 1, j = 0; i < n; i++) {
        int& k = z[i];
        k = (j + z[j] <= i) ? 0 : std::min(j + z[j] - i, z[i - j]);
        while (i + k < n && s[k] == s[i + k]) k++;
        if (j + z[j] < i + z[i]) j = i;
    }
    z[0] = n;
    return z;
}

std::vector<int> Zfunction(const std::string& s) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return Zfunction(s2);
}

std::vector<int> kmp(std::string_view s) {
```

```
    const int n = s.size();
    std::vector<int> f(n + 1);
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) {
            j = f[j];
        }
        j += (s[i] == s[j]);
        f[i + 1] = j;
    }
    return f;
}
```

Flow

```
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;
    }
};
```

HLD

```
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 directChild(int u, int v) {
    return *std::prev(std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
        return in[x] < in[y];
    }));
}
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);
    }
};
```

RMQ

```cpp
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[std::countr_zero(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

sieve

```
std::vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

SCC

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

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

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

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

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            }
            else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                vec[cnt].push_back(y);
                stk.pop_back();
            } while (y != x);
```

```
            cnt++;
        }
    }

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

EBCC

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

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

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

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

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                dfs(y, x);
                low[x] = std::min(low[x], low[y]);
            }
            else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
```

```
            cnt++;
        }
    }

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

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

BASIS2

```cpp
using u64 = unsigned long long;
template<int N>
struct Basis {
    std::array<u64, N> p{};

    Basis() {
        p.fill(0);
    }

    bool add(u64 x) {
        for (int i = N - 1; i >= 0; i--) {
            if (x >> i & 1) {
                if (p[i] == 0) {
                    p[i] = x;
                    return true;
                }
                x ^= p[i];
            }
        }
        return false;
    }

    u64 query() {
        u64 ans = 0;
        for (int i = N - 1; i >= 0; i--) {
            if ((ans ^ p[i]) > ans) {
                ans ^= p[i];
            }
        }
        return ans;
    }

    bool find(u64 x) {
        for (int i = N - 1; i >= 0; i--) {
            if (x >> i & 1) {
                if (p[i] == 0) {
                    return true;
                }
                x ^= p[i];
            }
        }
        return false;
    }
};
```

ModInt

```
using u32 = unsigned;
using u64 = unsigned long long;
template<typename T>
constexpr T power(T a, u64 b) {
    T res{ 1 };
    for (; b != 0; b /= 2, a *= a) {
        if (b % 2 == 1) {
            res *= a;
        }
    }
    return res;
}

template<u32 P>
constexpr u32 mulMod(u32 a, u32 b) {
    return 1ULL * a * b % P;
}

template<u64 P>
constexpr u64 mulMod(u64 a, u64 b) {
    u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
    res %= P;
    return res;
}

template<typename U, U P>
    requires std::unsigned_integral<U>
class ModIntBase {
public:
    constexpr ModIntBase() : x{ 0 } {}

    template<typename T>
        requires std::integral<T>
    constexpr ModIntBase(T x_) : x{ norm(x_ % T {P}) } {}

    constexpr static U norm(U x) {
        if ((x >> (8 * sizeof(U) - 1) & 1) == 1) {
            x += P;
        }
        if (x >= P) {
            x -= P;
        }
        return x;
    }

    constexpr U val() const {
        return x;
    }

    constexpr ModIntBase operator-() const {
```

```
    ModIntBase res;
    res.x = norm(P - x);
    return res;
}
constexpr ModIntBase operator+() const {
    return *this;
}
constexpr ModIntBase& operator++() {
    x++;
    if (x == P) {
        x = 0;
    }
    return *this;
}
constexpr ModIntBase& operator--() {
    if (x == 0) {
        x = P;
    }
    x--;
    return *this;
}
constexpr ModIntBase operator++(int) {
    ModIntBase result = *this;
    ++*this;
    return result;
}
constexpr ModIntBase operator--(int) {
    ModIntBase result = *this;
    --*this;
    return result;
}

constexpr ModIntBase inv() const {
    return power(*this, P - 2);
}

constexpr ModIntBase& operator*=(const ModIntBase& rhs)& {
    x = mulMod<P>(x, rhs.val());
    return *this;
}

constexpr ModIntBase& operator+=(const ModIntBase& rhs)& {
    x = norm(x + rhs.x);
    return *this;
}

constexpr ModIntBase& operator-=(const ModIntBase& rhs)& {
    x = norm(x - rhs.x);
    return *this;
}

constexpr ModIntBase& operator/=(const ModIntBase& rhs)& {
    return *this *= rhs.inv();
}

friend constexpr ModIntBase operator*(ModIntBase lhs, const ModIntBase& rhs) {
    lhs *= rhs;
    return lhs;
```

```
    }

    friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase& rhs) {
        lhs += rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase& rhs) {
        lhs -= rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase& rhs) {
        lhs /= rhs;
        return lhs;
    }

    friend constexpr std::ostream& operator<<(std::ostream& os, const ModIntBase& a) {
        return os << a.val();
    }

    friend constexpr bool operator==(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() == rhs.val();
    }

    friend constexpr bool operator!=(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() != rhs.val();
    }

    friend constexpr bool operator<(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() < rhs.val();
    }

    friend constexpr bool operator>(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() > rhs.val();
    }
private:
    U x;
};

template<u32 P>
using ModInt = ModIntBase<u32, P>;

template<u64 P>
using ModInt64 = ModIntBase<u64, P>;

constexpr u32 P = 998244353;
using mint = ModInt<P>;
template<>
struct std::formatter<mint> {
    constexpr auto parse(std::format_parse_context& ctx) {
        return ctx.begin();
    }

    constexpr auto format(const mint& x, std::format_context& ctx) const {
        return std::format_to(ctx.out(), "{}", x.val());
    }
};
```

前缀和 (0base)

一维:

```
std::vector a(n, 0), s(n + 1, 0LL);
for (int i = 0; i < n; i++) {
    s[i + 1] = s[i] + a[i];
}
```

二维:

```
#include <bits/stdc++.h>

int main() {
    int n, m, q;
    scanf("%d%d%d", &n, &m, &q);

    std::vector a(n, std::vector<int>(m));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    std::vector s(n + 1, std::vector<int>(m + 1));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            s[i + 1][j + 1] = s[i][j + 1] + s[i + 1][j] - s[i][j] + a[i][j];
        }
    }

    while (q--) {
        int l1, r1, l2, r2;
        scanf("%d%d%d%d", &l1, &r1, &l2, &r2);
        l1--, l2--, r1--, r2--;
        printf("%d\n", s[l2 + 1][r2 + 1] - s[l1][r2 + 1] - s[l2 + 1][r1] + s[l1][r1]);
    }

    return 0;
}
```

差分(0base)

一维

```
#include <bits/stdc++.h>

int main() {
    int n, m;
    scanf("%d%d", &n, &m);

    std::vector<int> a(n);

    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    std::vector<int> diff(n + 1);
    for (int i = 0; i < n; i++) {
        diff[i] = a[i] - (i - 1 >= 0 ? a[i - 1] : 0);
    }

    while (m--) {
        int l, r, v;
        scanf("%d%d%d", &l, &r, &v);
        l--;
        diff[l] += v;
        diff[r] -= v;
    }

    auto ans = diff;
    for (int i = 0; i < n; i++) {
        ans[i + 1] += ans[i];
    }

    for (int i = 0; i < n; i++) {
        printf("%d%c", ans[i], " \n"[i == n - 1]);
    }

    return 0;
}
```

二维

```
#include <bits/stdc++.h>

int main() {
    int n, m, q;
    scanf("%d%d%d", &n, &m, &q);
```

```
    std::vector a(n, std::vector<int>(m));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    std::vector diff(n + 1, std::vector<int>(m + 1));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            diff[i][j] = a[i][j];
            if (i > 0) {
                diff[i][j] -= a[i - 1][j];
            }
            if (j > 0) {
                diff[i][j] -= a[i][j - 1];
            }
            if (i > 0 && j > 0) {
                diff[i][j] += a[i - 1][j - 1];
            }
        }
    }

    while (q--) {
        int l1, r1, l2, r2, v;
        scanf("%d%d%d%d%d", &l1, &r1, &l2, &r2, &v);
        l1--, r1--, l2--, r2--;
        diff[l1][r1] += v;
        diff[l1][r2 + 1] -= v;
        diff[l2 + 1][r1] -= v;
        diff[l2 + 1][r2 + 1] += v;
    }

    std::vector ans(n + 1, std::vector<int>(m + 1));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            ans[i + 1][j + 1] = ans[i + 1][j] + ans[i][j + 1] - ans[i][j] + diff[i][j];
        }
    }

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

    return 0;
}
```

二分图判定

```
auto judge = [&]() {
    std::queue<int> q;
    std::vector<int> col(n, -1);
    for (int i = 0; i < n; i++) {
        if (col[i] != -1) continue;
        q.push(i);
        col[i] = 0;
        while (!q.empty()) {
            auto u = q.front();
            q.pop();
            for (int v : adj[u]) {
                if (col[v] == -1) {
                    q.push(v);
                    col[v] = col[u] ^ 1;
                } else if (col[v] == col[u]) {
                    return false;
                }
            }
        }
    }
    return true;
};
```

2sat

```cpp
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<int> 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 add(int u, int v) {e[u].push_back(v);}
    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<int> answer() { return ans; }
};
```

Combination

```cpp
namespace comb {
    constexpr int N = 1e6;
    int P;

    bool is_init;
    int _fac[N + 1], _ifac[N + 1];

    constexpr i64 power(i64 a, i64 b) {
        i64 res = 1;
        for (; b; b /= 2, a = a * a % P) {
            if (b % 2) {
                res = res * a % P;
            }
        }
        return res;
    }

    void init(int N = comb::N) {
        _fac[0] = 1;
        for (int i = 1; i <= N; i++) {
            _fac[i] = 1LL * _fac[i - 1] * i % P;
        }
        _ifac[N] = power(_fac[N], P - 2);
        for (int i = N; i >= 1; i--) {
            _ifac[i - 1] = 1LL * _ifac[i] * i % P;
        }
        is_init = true;
    }

    int fac(int n) {
        if (!is_init) init();
        return n < 0 ? 0 : _fac[n];
    }

    int ifac(int n) {
        if (!is_init) init();
        return n < 0 ? 0 : _ifac[n];
    }

    int perm(int n, int m) {
        if (n < m || m < 0) return 0;
        return 1LL * fac(n) * ifac(n - m) % P;
    }

    int binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return 1LL * fac(n) * ifac(m) % P * ifac(n - m) % P;
    }

    int lucas(i64 n, i64 m) { // 使用前 init(P - 1);
```

```
        if (n < m || m < 0) return 0;
        if (m == 0) return 1;
        return 1LL * binom(n % P, m % P) * lucas(n / P, m / P) % P; // assume P is prime
    }
}
using namespace comb;
```

主席树

```
constexpr int V = 1e9 + 1;

struct Node {
    Node* l = nullptr;
    Node* r = nullptr;
    int cnt = 0;
    Node(Node* t) {
        if (t != nullptr) {
            *this = *t;
        }
    }
};

constexpr int lcnt(const Node* t) {
    return t && t->l ? t->l->cnt : 0;
}

constexpr int rcnt(const Node* t) {
    return t && t->r ? t->r->cnt : 0;
}

constexpr Node* rs(Node* t) {
    return t ? t->r : t;
}

constexpr Node* ls(Node* t) {
    return t ? t->l : t;
}

Node* add(Node* p, int l, int r, int x, int v) {
    p = new Node(p);
    p->cnt += v;
    if (r - l == 1) {
        return p;
    }
    int m = (l + r) / 2;
    if (x < m) {
        p->l = add(p->l, l, m, x, v);
    }
    else {
        p->r = add(p->r, m, r, x, v);
    }
    return p;
}
Node* add(Node* p, int x, int v) {
    return add(p, 0, V, x, v);
}

int kmin(Node* t1, Node* t2, int l, int r, int k) {
    if (r - l == 1) {
```

```
        return l;
    }
    int cnt = lcnt(t2) - lcnt(t1);
    int m = (l + r) / 2;
    if (k > cnt) {
        return kmin(rs(t1), rs(t2), m, r, k - cnt);
    }
    return kmin(ls(t1), ls(t2), l, m, k);
}
int kmin(Node* t1, Node* t2, int k) {
    return kmin(t1, t2, 0, V, k);
}

int kmax(Node* t1, Node* t2, int l, int r, int k) {
    if (r - l == 1) {
        return l;
    }
    int cnt = rcnt(t2) - rcnt(t1);
    int m = (l + r) / 2;
    if (cnt > k) {
        return kmax(rs(t1), rs(t2), m, r, k);
    }
    return kmax(ls(t1), ls(t2), l, m, k - cnt);
}
int kmax(Node* t1, Node* t2, int k) {
    return kmax(t1, t2, 0, V, k);
}
```