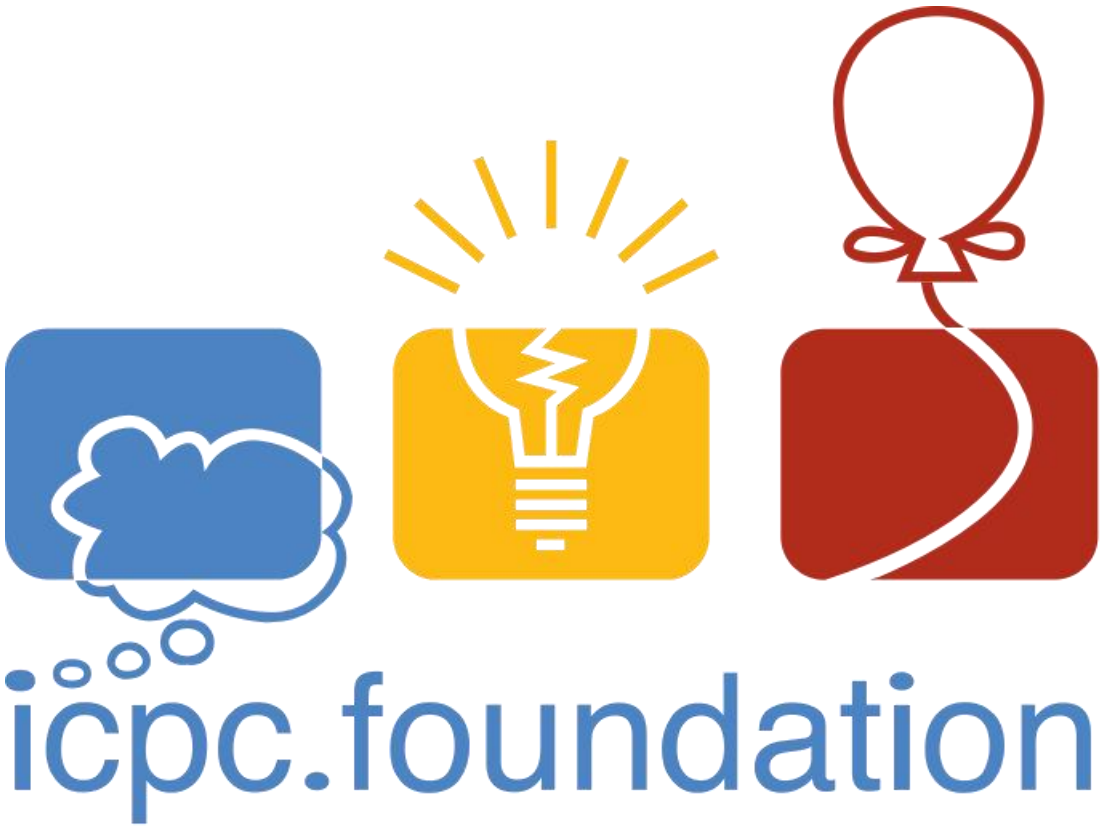

jzz


无数次渺小的思考填满了人的一生

目录

DataStructure

DSU

```
struct DSU {
    std::vector<int> f, siz;
    DSU(int n) : f(n + 1), siz(n + 1, 1) {
        std::iota(f.begin(), f.end(), 0);
    }
    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)];
    }
};
```

fenwick

```
template <typename T>
struct fenwick {
    int n;
    std::vector<T> tr;
    fenwick(int _n = 0) {
        n = _n + 1;
        tr.assign(n, T{});
    }
    void add(int x, const T &v) {
```

```
        for (int i = x + 1; i <= n; i += i & -i) {
            tr[i - 1] = tr[i - 1] + v;
        }
    }
    T Sum(int x) {
        T ans{};
        for (int i = x + 1; i > 0; i -= i & -i) {
            ans = ans + tr[i - 1];
        }
        return ans;
    }
    T Sum(int l, int r) {
        return Sum(r) - Sum(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 + tr[x + i - 1] < k) {
                x += i;
                cur = cur + tr[x - 1];
            }
        }
        return x;
    }
};
```

fenwick2D

```
template<typename T>
struct fenwick2D{
    vector<vector<T>> tr;
    int n, m;
    fenwick2D(int _n, int _m) : n(_n), m(_m) {
        tr.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            tr[i].resize(m + 1);
        }
    }
    void add(int x, int y, const T &val) { // 1 <= x <= n, 1 <= y <= m
        for(int i = x; i <= n; i += i & -i) {
            for(int j = y; j <= m; j += j & -j) {
                tr[i][j] += val;
            }
```

```
        }
    }
    T Sum(int x, int y) { // 1 <= x <= n, 1 <= y <= m
        T res = 0;
        for(int i = x; i > 0; i -= i & -i) {
            for(int j = y; j > 0; j -= j & -j) {
                res += tr[i][j];
            }
        }
        return res;
    }
    T Sum(int x1, int y1, int x2, int y2) {  // sum[x1..x2, y1..y2]
        return Sum(x2, y2) - Sum(x2, y1 - 1) - Sum(x1 - 1, y2) + Sum(x1 -
1, y1 - 1);
    }
};
```

区间修改-fenwick

```
template <typename T>
struct fenwick {
    int n;
    std::vector<std::vector<T>> tr;
    fenwick(int _n) : n(_n) {
        tr.resize(2);
        for (int i = 0; i < 2; i++) {
            tr[i].resize(n + 2);
        }
    }
    void add(int i, int x, const T &v) {
        for ( ; x <= n; x += x & -x) {
            tr[i][x] += v;
        }
    }
    void modify(int l, int r, const T &v) {
        add(0, l, v);
        add(0, r + 1, -v);
        add(1, l, l * v);
        add(1, r + 1, (r + 1) * (-v));
    }
    T sum(int i, int x) {
        T ans = 0;
        for ( ; x > 0; x -= x & -x) {
            ans += tr[i][x];
```

```
        }
        return ans;
    }
    T Sum(int x) {
        return sum(0, x) * (x + 1) - sum(1, x);
    }
    T Sum(int l, int r) {
        return Sum(r) - Sum(l - 1);
    }
};
```

LCA

```
template <typename T>
struct LCA {
    int m;
    std::vector<int> d;
    std::vector<T> dist;
    std::vector<std::vector<pair<int, T>>> g;
    std::vector<std::vector<int>> f;
    LCA(int n) : d(n + 1), dist(n + 1), g(n + 1), f(n + 1) {
        m = __lg(n) + 1;
        for (int i = 0; i <= n; i++) {
            f[i].resize(m + 1);
        }
    }
    void add(int u, int v, T w = 1) {
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }
    void build(int root = 1) {
        queue<int> q;
        q.push(root);
        d[root] = 1;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto [v, w] : g[u]) {
                if (d[v]) continue;
                d[v] = d[u] + 1;
                dist[v] = dist[u] + w;
                f[v][0] = u;
                for (int k = 1; k <= m; k++) {
                    f[v][k] = f[f[v][k - 1]][k - 1];
                }
```

```
                q.push(v);
            }
        }
    }
    int lca(int a, int b) {
        if (d[a] < d[b]) swap(a, b);
        for (int k = m; k >= 0; k--) {
            if (d[f[a][k]] >= d[b]) a = f[a][k];
        }
        if (a == b) return a;
        for (int k = m; k >= 0; k--) {
            if (f[a][k] != f[b][k]) {
                a = f[a][k]; b = f[b][k];
            }
        }
        return f[a][0];
    }
    T dis(int u, int v) {
        return dist[u] + dist[v] - dist[lca(u, v)] * 2;
    }
};
```

ODT

```
struct ODT {
    struct odt {
        int l, r;
        mutable int x;
        bool operator < (const odt &a) const {
            return l < a.l;
        }
    };
    set<odt> tr;
    typedef set <odt> :: iterator IT;
    ODT(int l, int r, int x) {
        tr.insert({l, r, x});
    }
    IT split(int pos) { //将 pos-1 和 pos 之间切开，返回 pos 所在区间指针
        auto it = tr.lower_bound({pos, 0, 0});
        if (it != tr.end() && it->l == pos) return it;
        it--;
        int l = it->l, r = it->r, x = it->x;
        tr.erase(it);
        tr.insert({l, pos - 1, x});
```

```
        return tr.insert({pos, r, x}).first;
    }
    void assign(int l, int r, int x) {
        auto R = split(r + 1);
        auto L = split(l);
        tr.erase(L, R);
        tr.insert({l, r, x});
    }
    void modify(int l, int r) {
        auto R = split(r + 1);
        auto L = split(l);
        for (auto it = L; it != R; it++) {
            // 对 it->x 暴力修改
        }
    }
    int query() {
        int ans = 0;
        for (auto it = tr.begin(); it != tr.end(); it++) {

        }
        return ans;
    }
};
```

RMQ

```
template <class Info>
struct RMQ {
    std::vector<int> lg;
    std::vector<std::vector<Info>> f;
    RMQ(std::vector<int> a) : lg(a.size()) {
        int n = a.size() - 1, m = __lg(n) + 1;
        f.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            f[i].resize(m + 1);
            f[i][0] = {a[i]};
            if (i) lg[i] = __lg(i);
        }
        for (int j = 1; j <= m; j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                f[i][j] = f[i][j - 1] + f[i + (1 << j - 1)][j - 1];
            }
        }
    }
```

```
    Info Query(int l, int r) {
        int k = lg[r - l + 1];
        return f[l][k] + f[r - (1 << k) + 1][k];
    }
};
struct Info {
    int x;
    friend Info operator+(const Info &a, const Info &b) {

    }
};
```

RMQ-2D-子矩阵为矩形

```
struct RMQ_2D {
    int n, m;
    std::vector<std::vector<std::vector<std::vector<int>>>> f;
    std::vector<std::vector<std::vector<std::vector<int>>>> g;
    RMQ_2D(std::vector<std::vector<int>> a) : n(a.size() - 1), m(a[0].size()
- 1) {
        f.resize(n + 1);
        g.resize(n + 1);
        int N = __lg(n) + 1, M = __lg(m) + 1;
        for (int i = 0; i <= n; i++) {
            f[i].resize(m + 1);
            g[i].resize(m + 1);
            for (int j = 0; j <= m; j++) {
                f[i][j].resize(N + 1);
                g[i][j].resize(N + 1);
                for (int k = 0; k <= N; k++) {
                    g[i][j][k].resize(M + 1);
                    f[i][j][k].resize(M + 1);
                }
            }
        }
        for (int k = 0; k <= N; k++) {
            for (int l = 0; l <= M; l++) {
                for (int i = 1; i + (1 << k) - 1 <= n; i++) {
                    for (int j = 1; j + (1 << l) - 1 <= m; j++) {
                        if (k == 0 && l == 0) {
                            f[i][j][k][l] = a[i][j];
                            g[i][j][k][l] = a[i][j];
                        } else if (k == 0) {
                            f[i][j][k][l] = max(f[i][j][k][l - 1], f[i][j +
```

```
(1 << l - 1)][k][l - 1]);
                        g[i][j][k][l] = min(g[i][j][k][l - 1], g[i][j +
(1 << l - 1)][k][l - 1]);
                    } else if (l == 0) {
                        f[i][j][k][l] = max(f[i][j][k - 1][l], f[i + (1
<< k - 1)][j][k - 1][l]);
                        g[i][j][k][l] = min(g[i][j][k - 1][l], g[i + (1
<< k - 1)][j][k - 1][l]);
                    } else {
                        f[i][j][k][l] = max({f[i][j][k - 1][l - 1], f[i
+ (1 << k - 1)][j][k - 1][l - 1], f[i][j + (1 << l - 1)][k - 1][l - 1], f[i
+ (1 << k - 1)][j + (1 << l - 1)][k - 1][l - 1]});
                        g[i][j][k][l] = min({g[i][j][k - 1][l - 1], g[i
+ (1 << k - 1)][j][k - 1][l - 1], g[i][j + (1 << l - 1)][k - 1][l - 1], g[i
+ (1 << k - 1)][j + (1 << l - 1)][k - 1][l - 1]});
                    }
                }
            }
        }
    }
    int Max(int x1, int y1, int x2, int y2) {
        int k = __lg(x2 - x1 + 1) / __lg(2);
        int l = __lg(y2 - y1 + 1) / __lg(2);
        return max({f[x1][y1][k][l], f[x2 - (1 << k) + 1][y1][k][l], f[x1][y2
- (1 << l) + 1][k][l], f[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]});
    };
    int Min(int x1, int y1, int x2, int y2) {
        int k = __lg(x2 - x1 + 1) / __lg(2);
        int l = __lg(y2 - y1 + 1) / __lg(2);
        return min({g[x1][y1][k][l], g[x2 - (1 << k) + 1][y1][k][l], g[x1][y2
- (1 << l) + 1][k][l], g[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]});
    };
};
```

RMQ-2D-子矩阵为正方形

```
struct RMQ_2D {
    int n, m;
    std::vector<std::vector<std::vector<int>>> f;
    std::vector<std::vector<std::vector<int>>> g;
    RMQ_2D(std::vector<std::vector<int>> a) : n(a.size() - 1), m(a[0].size()
- 1) {
        f.resize(n + 1);
```

```
        g.resize(n + 1);
        int N = __lg(max(n, m)) + 1;
        for (int i = 0; i <= n; i++) {
            f[i].resize(m + 1);
            g[i].resize(m + 1);
            for (int j = 0; j <= m; j++) {
                f[i][j].resize(N + 1);
                g[i][j].resize(N + 1);
            }
        }
        for (int k = 0; k <= N; k++) {
            for (int i = 1; i + (1 << k) - 1 <= n; i++) {
                for (int j = 1; j + (1 << k) - 1 <= m; j++) {
                    if (k == 0) {
                        f[i][j][k] = a[i][j];
                        g[i][j][k] = a[i][j];
                    } else {
                        f[i][j][k] = max({f[i][j][k - 1], f[i + (1 << k -
1)][j][k - 1], f[i][j + (1 << k - 1)][k - 1], f[i + (1 << k - 1)][j + (1
<< k - 1)][k - 1]});
                        g[i][j][k] = min({g[i][j][k - 1], g[i + (1 << k -
1)][j][k - 1], g[i][j + (1 << k - 1)][k - 1], g[i + (1 << k - 1)][j + (1
<< k - 1)][k - 1]});
                    }
                }
            }
        }
    }
    int Max(int x1, int y1, int x2, int y2) {
        int k = __lg(x2 - x1 + 1) / __lg(2);
        return max({f[x1][y1][k], f[x2 - (1 << k) + 1][y1][k], f[x1][y2 -
(1 << k) + 1][k], f[x2 - (1 << k) + 1][y2 - (1 << k) + 1][k]});
    };
    int Min(int x1, int y1, int x2, int y2) {
        int k = __lg(x2 - x1 + 1) / __lg(2);
        return min({g[x1][y1][k], g[x2 - (1 << k) + 1][y1][k], g[x1][y2 -
(1 << k) + 1][k], g[x2 - (1 << k) + 1][y2 - (1 << k) + 1][k]});
    };
};
```

LazySegmentTree

```
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_ + 1, v_));
}
template<class T>
void init(std::vector<T> init_) {
    n = init_.size() - 1;
    info.assign(4 << std::__lg(n + 1), Info());
    tag.assign(4 << std::__lg(n + 1), Tag());
    auto build = [&](auto build, int u, int l, int r) -> void {
        if (l == r) {
            info[u] = {init_[l]};
            return;
        }
        int mid = l + r >> 1;
        build(build, u << 1, l, mid);
        build(build, u << 1 | 1, mid + 1, r);
        pushup(u);
    };
    build(build, 1, 0, n);
}
void pushup(int u) {
    info[u] = info[u << 1] + info[u << 1 | 1];
}
void apply(int u, const Tag &v) {
    info[u].apply(v);
    tag[u].apply(v);
}
void pushdown(int u) {
    apply(u << 1, tag[u]);
    apply(u << 1 | 1, tag[u]);
    tag[u] = Tag();
}
void modify(int u, int l, int r, int x, const Info &v) {
```

```
    if (l == r) {
        info[u] = v;
        return;
    }
    int mid = l + r >> 1;
    pushdown(u);
    if (x <= mid) {
        modify(u << 1, l, mid, x, v);
    } else {
        modify(u << 1 | 1, mid + 1, r, x, v);
    }
    pushup(u);
}
void modify(int p, const Info &v) {
    modify(1, 0, n, p, v);
}
void rangeApply(int u, int l, int r, int x, int y, const Tag &v) {
    if (r < x || l > y) {
        return;
    }
    if (l >= x && r <= y) {
        apply(u, v);
        return;
    }
    int mid = l + r >> 1;
    pushdown(u);
    rangeApply(u << 1, l, mid, x, y, v);
    rangeApply(u << 1 | 1, mid + 1, r, x, y, v);
    pushup(u);
}
void Apply(int p, const Tag &v) {
    rangeApply(1, 0, n, p, p, v);
}
void rangeApply(int l, int r, const Tag &v) {
    rangeApply(1, 0, n, l, r, v);
}
Info rangeQuery(int u, int l, int r, int x, int y) {
    if (r < x || l > y) {
        return Info();
    }
    if (x <= l && r <= y) {
        return info[u];
    }
    int mid = l + r >> 1;
```

```
        pushdown(u);
        if (y <= mid) {
            return rangeQuery(u << 1, l, mid, x, y);
        } else if (x > mid) {
            return rangeQuery(u << 1 | 1, mid + 1, r, x, y);
        }
        auto left = rangeQuery(u << 1, l, mid, x, y);
        auto right = rangeQuery(u << 1 | 1, mid + 1, r, x, y);
        return left + right;
    }
    Info Query(int p) {
        return rangeQuery(1, 0, n, p, p);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int u, int l, int r, int x, int y, F &&pred) {
        if (r < x || l > y) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[u])) {
            return -1;
        }
        if (l == r) {
            return l;
        }
        int mid = l + r >> 1;
        int res = findFirst(u << 1, l, mid, x, y, pred);
        if (res == -1) {
            res = findFirst(u << 1 | 1, mid + 1, 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 u, int l, int r, int x, int y, F &&pred) {
        if (r < x || l > y) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[u])) {
```

```
            return -1;
        }
        if (l == r) {
            return l;
        }
        int mid = l + r >> 1;
        int res = findLast(u << 1 | 1, mid + 1, r, x, y, pred);
        if (res == -1) {
            res = findLast(u << 1, l, mid, 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 {

    void apply(const Tag &t) {

    }
};

struct Info {

    void apply(const Tag &t) {

    }
};

Info operator+(const Info &a, const Info &b) {

}
```

HLD

```
template<class Info, class Tag>
struct TreeChainPartition : LazySegmentTree<Info, Tag> {
    using LazySegmentTree<Info, Tag>::n;
    using LazySegmentTree<Info, Tag>::modify;
    using LazySegmentTree<Info, Tag>::rangeApply;
    using LazySegmentTree<Info, Tag>::rangeQuery;
```

```
    std::vector<int> d, f, son, sz;
    std::vector<int> dfn, seq, top;
    int cnt;
    std::vector<std::vector<int>> g;

    TreeChainPartition(int _n, vector<std::vector<int>> G) :
LazySegmentTree<Info, Tag> (_n) {
        g = G;
        d.resize(n + 1);
        f.resize(n + 1);
        son.resize(n + 1);
        sz.resize(n + 1);
        dfn.resize(n + 1);
        seq.resize(n + 1);
        top.resize(n + 1);
        cnt = 0;
        dfs1(1, -1, 0);
        dfs2(1, 1);
    }
    void add(int u, int v) {
        g[u].push_back(v);
        g[v].push_back(u);
    }
    void dfs1(int u, int fa, int dep) {
        d[u] = dep, f[u] = fa, sz[u] = 1;
        for (auto v : g[u]) {
            if (v == fa) continue;
            dfs1(v, u, dep + 1);
            sz[u] += sz[v];
            if (sz[v] > sz[son[u]]) son[u] = v;
        }
    }
    void dfs2(int u, int t) {
        dfn[u] = ++cnt;
        seq[cnt] = u;
        top[u] = t;
        if (son[u] == 0) return;
        dfs2(son[u], t);
        for (auto v : g[u]) {
            if (v == f[u] || v == son[u]) continue;
            dfs2(v, v);
        }
    }
```

```
void modify_point(int u, const Info &k) {
    modify(dfn[u], k);
}
void modify_path(int u, int v, const Info &k) {
    while (top[u] != top[v]) {
        if (d[top[u]] < d[top[v]]) swap(u, v);
        rangeApply(dfn[top[u]], dfn[u], k);
        u = f[top[u]];
    }
    if (d[u] > d[v]) swap(u, v);
    rangeApply(dfn[u], dfn[v], k);
}
void modify_tree(int u, const Info &k) {
    rangeApply(dfn[u], dfn[u] + sz[u] - 1, k);
}

Info query_path(int u, int v) {
    Info ans = Info();
    while (top[u] != top[v]) {
        if (d[top[u]] < d[top[v]]) swap(u, v);
        ans = ans + rangeQuery(dfn[top[u]], dfn[u]);
        u = f[top[u]];
    }
    if (d[u] > d[v]) swap(u, v);
    ans = ans + rangeQuery(dfn[u], dfn[v]);
    return ans;
}
Info query_tree(int u) {
    return rangeQuery(dfn[u], dfn[u] + sz[u] - 1);
}

int lca(int u, int v) {
    while (top[u] != top[v]) {
        if (d[top[u]] > d[top[v]]) {
            u = f[top[u]];
        } else {
            v = f[top[v]];
        }
    }
    return d[u] < d[v] ? u : v;
}
int dist(int u, int v) {
    return d[u] + d[v] - 2 * d[lca(u, v)];
```

```
    }
    bool isAncester(int u, int v) {
        return dfn[u] <= dfn[v] && dfn[v] <= dfn[u] + sz[u] - 1;
    }
    int jump(int u, int k) {
        if (d[u] < k) {
            return -1;
        }
        int dx = d[u] - k;
        while (d[top[u]] > dx) {
            u = f[top[u]];
        }
        return seq[dfn[u] - d[u] + dx];
    }
};
```

ST

```
template <class Info>
struct ST {
    int n, m;
    std::vector<int> lg;
    std::vector<std::vector<Info>> f;
    ST(std::vector<int> a) : n(a.size() - 1), lg(a.size()) {
        m = __lg(n) + 1;
        f.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            f[i].resize(m + 1);
            f[i][0] = {a[i]};
            if (i) lg[i] = __lg(i);
        }
        for (int j = 1; j <= m; j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                f[i][j] = f[i][j - 1] + f[i + (1 << j - 1)][j - 1];
            }
        }
    }
    Info Query(int l, int r) {
        Info ans = f[l][0];
        int p = l + 1;
        for (int i = m; i >= 0; i--) {
            if (p + (1 << i) - 1 <= r) {
                ans = ans + f[p][i];
                p += 1 << i;
```

```
            }
        }
        return ans;
    }
};
struct Info {
    int x;
    friend Info operator+(const Info &a, const Info &b) {

    }
};
```

主席树

```
const int N = 2e5 + 10;
int n, m, k;
int len, idx, a[N], b[N], root[N];

struct node {
    int l, r, s;
} tr[N * 21];  //  n*log2(n) 个结点
int find(int x) {
    return lower_bound(b + 1, b + 1 + len, x) - b;
}
int build(int l, int r) {
    int u = ++idx;
    tr[u].s = 0;
    if (l == r) return u;
    int mid = l + r >> 1;
    tr[u].l = build(l, mid);
    tr[u].r = build(mid + 1, r);
    return u;
}
int insert(int p, int l, int r, int x) {
    int u = ++idx;
    tr[u] = tr[p];
    if (l == r) return tr[u].s++, u;
    int mid = l + r >> 1;
    if (x <= mid) tr[u].l = insert(tr[p].l, l, mid, x);
    else tr[u].r = insert(tr[p].r, mid + 1, r, x);
    tr[u].s = tr[tr[u].l].s + tr[tr[u].r].s;
    return u;
}
// 大于等于 k 的个数
```

```
int query1(int x, int y, int l, int r, int k) {
    if (k > b[r]) return 0;
    if (k <= b[l]) return tr[x].s - tr[y].s;
    int mid = l + r >> 1;
    int ans = 0;
    if(k <= b[mid]) {
        ans += tr[tr[x].r].s - tr[tr[y].r].s;
        ans += query1(tr[x].l, tr[y].l, l, mid, k);
    } else {
        ans += query1(tr[x].r, tr[y].r, mid + 1, r, k);
    }
    return ans;
}
// 小于等于 k 的个数
int query2(int x, int y, int l, int r, int k) {
    if(k >= b[r]) return tr[x].s - tr[y].s;
    if (k < b[l]) return 0;
    int mid = l + r >> 1;
    int ans = 0;
    if(k <= b[mid]) {
        ans += query2(tr[x].l, tr[y].l, l, mid, k);
    } else {
        ans += tr[tr[x].l].s - tr[tr[y].l].s;
        ans += query2(tr[x].r, tr[y].r, mid + 1, r, k);
    }
    return ans;
}
// 大于 k 的个数
int query3(int x, int y, int l, int r, int k) {
    if(k >= b[r]) return 0;
    if(k < b[l]) return tr[x].s - tr[y].s;
    int mid = l + r >> 1;
    int ans = 0;
    if(k <= b[mid]) {
        ans += tr[tr[x].r].s - tr[tr[y].r].s;
        ans += query3(tr[x].l, tr[y].l, l, mid, k);
    } else {
        ans += query3(tr[x].r, tr[y].r, mid + 1, r, k);
    }
    return ans;
}
// 小于 k 的个数
int query4(int x, int y, int l, int r, int k) {
    if(k > b[r]) return tr[x].s - tr[y].s;
```

```
    if(k <= b[l]) return 0;
    int mid = l + r >> 1;
    int ans = 0;
    if(k <= b[mid]) {
        ans += query4(tr[x].l, tr[y].l, l, mid, k);
    } else {
        ans += tr[tr[x].l].s - tr[tr[y].l].s;
        ans += query4(tr[x].r, tr[y].r, mid + 1, r, k);
    }
    return ans;
}
// 等于 k 的个数
int query5(int x, int y, int l, int r, int k) {
    if(l == r) return k == b[l] ? tr[x].s - tr[y].s : 0;
    int mid = l + r >> 1;
    if(k <= b[mid]) return query5(tr[x].l, tr[y].l, l, mid, k);
    else return query5(tr[x].r, tr[y].r, mid + 1, r, k);
}
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i]; b[i] = a[i];
    }
    sort(b + 1, b + 1 + n);
    len = unique(b + 1, b + 1 + n) - b - 1;

    root[0] = build(1, len);
    for (int i = 1; i <= n; i++) {
        root[i] = insert(root[i - 1], 1, len, find(a[i]));
    }
    /*-------------------------------------*/
}
```

单调队列

```
    deque<int> q;
    for (int i = 1; i <= n; i++) {
        if (q.size() && q.front() < i - k + 1) q.pop_front();
        while (q.size() && a[i] <= a[q.back()]) q.pop_back();
        q.push_back(i);
        if (i >= k) cout << a[q.front()] << " \n"[i == n];
    }
    while (q.size()) q.pop_back();
    for (int i = 1; i <= n; i++) {
```

```
        if (q.size() && q.front() < i - k + 1) q.pop_front();
        while (q.size() && a[i] >= a[q.back()]) q.pop_back();
        q.push_back(i);
        if (i >= k) cout << a[q.front()] << " \n"[i == n];
    }
```

单调栈

```
    vector<int> LMin(n + 1);
    stack<int> st;
    for (int i = 1; i <= n; i++) {
        while (st.size() && a[st.top()] >= a[i]) st.pop();
        LMin[i] = st.size() ? a[st.top()] : -1;
        st.push(i);
    }
```

DynamicMedian

```
struct DynamicMedian {
    priority_queue<int> down;
    priority_queue<int, vector<int>, greater<int>> up;
    DynamicMedian() {}
    void insert(int x) {
        if (down.empty() || x <= down.top()) {
            down.push(x);
        } else {
            up.push(x);
        }
        if (down.size() > 1 + up.size()) {
            up.push(down.top());
            down.pop();
        }
        if (up.size() > down.size()) {
            down.push(up.top());
            up.pop();
        }
    };
    double Ans() {
        if (up.size() + down.size() & 1) {
            return down.top();
        } else {
            return (down.top() + up.top()) / 2.0;
        }
```

```
    };
};
```

莫队

```
    int len = sqrtl(n) + 1;
    sort(all(qry), [&](array<int, 3> x, array<int, 3> y) {
        if (x[0] / len == y[0] / len) {
            return x[1] < y[1];
        }
        return x[0] / len < y[0] / len;
    });
    int res = 0;
    auto add = [&](int x) -> void {

    };
    auto del = [&](int x) -> void {

    };
    int L = 1, R = 0;
    vector<int> ans(m);
    for (auto [l, r, i] : qry) {
        while (R < r) add(a[++R]);
        while (R > r) del(a[R--]);
        while (L < l) del(a[L++]);
        while (L > l) add(a[--L]);
        ans[i] = res;
    }
    for (int i = 0; i < m; i++) {
        cout << ans[i] << " \n"[i == m - 1];
    }
```

Graphs

Tarjan

```
struct tarjan {
    int timestamp;
    vector<int> dfn, low, id;
    vector<vector<array<int, 2>>> g;
    stack<int> st;
    vector<bool> is_bridge;
    vector<vector<int>> dcc;

    tarjan(int n, int m, vector<int> &x, vector<int> &y) {
        g.resize(n + 1);
        for (int i = 1; i <= m; i++) {
            g[x[i]].push_back({y[i], i});
            g[y[i]].push_back({x[i], i});
        }
        dfn.resize(n + 1);
        low.resize(n + 1);
        timestamp = 0;
        id.resize(n + 1);
        is_bridge.resize(m + 1);
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                dfs(i, -1);
            }
        }
    }
    void dfs(int u, int fa) {
        dfn[u] = low[u] = ++timestamp;
        st.push(u);
        for (auto &[v, i] : g[u]) {
            if (v == fa) continue;
            if (!dfn[v]) {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] > dfn[u]) {
                    is_bridge[i] = true;
                }
            } else if (dfn[v] < dfn[u]) {
```

```
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            vector<int> t;
            do {
                t.push_back(st.top());
                st.pop();
                id[t.back()] = dcc.size() + 1;
            } while (t.back() != u);
            dcc.push_back(t);
        }
    }
};
```

二分图

```
bool find(int u) {
    for (auto v : g[u]) {
        if (st[v]) continue;
        st[v] = 1;
        if (!match[v] || find(match[v])) {
            match[v] = u;
            return true;
        }
    }
    return false;
}
```

Math

exgcd

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

exgcd-inv

```
int inv(int n, int M) {
    // n * x = 1 (mod M) -> n * x + M * y = 1
    int x, y;
    exgcd(n, M, x, y);
    x = (x % M + M) % M;
    return x;
}
```

exgcd 求解不定方程

求解 a * x + b * y = c; a, b, c 已知，求出满足条件的一对 x, y.

```
int d = exgcd(a, b, x, y);
if (c % d == 0) {
    x *= c / d;
    y *= c / d;
    int dx = abs(b / d);
    int dy = abs(a / d);
} else {
    // 无解
}
```

dx, dy 是满足等式两边平衡的 x, y 的最小变化量

Matrix

```
const int MatSize = 2;
struct Matrix {
    array<array<Z, MatSize>, MatSize> x;

    Matrix(int _val = 0) {
        for (int i = 0; i < MatSize; i++) {
            x[i][i] = _val;
        }
    }
    Matrix(array<array<Z, 2>, 2> _x) {
        x = _x;
    }

    friend Matrix operator *(const Matrix &lhs, const Matrix &rhs) {
        Matrix res = Matrix();
        for (int i = 0; i < MatSize; i++) {
            for (int j = 0; j < MatSize; j++) {
                for (int k = 0; k < MatSize; k++) {
                    res.x[i][j] += lhs.x[i][k] * rhs.x[k][j];
                }
            }
        }
        return res;
    }
    friend Matrix operator +(const Matrix &lhs, const Matrix &rhs) {
        Matrix res = Matrix();
        for (int i = 0; i < MatSize; i++) {
            for (int j = 0; j < MatSize; j++) {
                res.x[i][j] = lhs.x[i][j] + rhs.x[i][j];
            }
        }
        return res;
    }
};
```

phi

```
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
```

```
            while (n % i == 0) {
                n /= i;
            }
            res = res / i * (i - 1);
        }
    }
    if (n > 1) {
        res = res / n * (n - 1);
    }
    return res;
}
```

PrimeTable

```
namespace PrimeTable {
    std::vector<int> primes;
    std::vector<int> minp;
    std::vector<int> phi;
    std::vector<bool> is_prime;
    void primes_init(int n) {
        minp.resize(n + 1);
        phi.resize(n + 1);
        is_prime.resize(n + 1, true);
        is_prime[0] = is_prime[1] = false;
        for (int i = 2; i <= n; i++) {
            if (is_prime[i]) {
                primes.push_back(i);
                minp[i] = i;
                phi[i] = i - 1;
            }
            for (int j = 0; primes[j] * i <= n; j++) {
                is_prime[primes[j] * i] = false;
                minp[primes[j] * i] = primes[j];
                if(i % primes[j] == 0) {
                    phi[primes[j] * i] = phi[i] * primes[j];
                    break;
                }
                phi[primes[j] * i] = phi[i] * (primes[j] - 1);
            }
        }
    }
    bool IsPrime(int n) {
        if (n <= 1) return false;
        return is_prime[n];
```

```
    }
}
using namespace PrimeTable;
```

组合数

```
namespace Combinatorics {
    int n;
    std::vector<Z> _fac = {1}, _inv = {1};
    void init(int m) {
        if (m <= n) return;
        _fac.resize(m + 1);
        _inv.resize(m + 1);
        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _inv[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _inv[i - 1] = _inv[i] * i;
        }
        n = m;
    }
    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z C(int a, int b) {
        if (a < b || b < 0) return Z(0);
        return fac(a) * inv(b) * inv(a - b);
    }
    Z A(int a, int b) {
        if (a < b || b < 0) return Z(0);
        return fac(a) * inv(a - b);
    }
    Z H(int a, int b) {
        return C(a + b - 1, b);
    }
}
using namespace Combinatorics;
```

lucas

```
int C(int a, int b) {
    if (a < b) return 0;
    int ans = 1;
    for (int i = a, j = 1; i >= b + 1; i--, j++) {
        ans = ans * i % p;
        ans = ans * qmi(j, p - 2, p) % p;
    }
    return ans;
}
int lucas(int a, int b, int p) {
    if (a < p && b < p) {
        return C(a, b, p);
    }
    return C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}
```

int128 重载运算

```
istream &operator>>(istream &is, __int128 &n) {
    n = 0;
    string s;
    cin >> s;
    for (auto c : s) {
        n = n * 10 + c - '0';
    }
    return is;
}
ostream &operator<<(ostream &os, __int128 n) {
    string s;
    while (n) {
        s += '0' + n % 10;
        n /= 10;
    }
    reverse(s.begin(), s.end());
    return os << s;
}
```

龟速乘

```
__int128 mul(__int128 a, __int128 b, __int128 M) {
```

```
    a = (a % M + M) % M;
    b = (b % M + M) % M;
    __int128 ans = 0;
    while (b) {
        if (b & 1) {
            ans += a;
            if (ans >= M) {
                ans -= M;
            }
        }
        a <<= 1;
        if (a >= M) {
            a -= M;
        }
        b >>= 1;
    }
    return ans;
}
```

Miller_Rabin 大质数判定

```
__int128 power(__int128 n, __int128 k, __int128 M) {
    __int128 ans = 1;
    while (k) {
        if (k & 1) {
            ans = mul(ans, n, M);
        }
        n = mul(n, n, M);
        k >>= 1;
    }
    return ans;
}
bool check(__int128 x,__int128 M){
    __int128 mid = power(x, M - 1, M);
    if (mid != 1) return false;
    __int128 n = M - 1;
    while (n % 2 == 0 && mid == 1) {
        n >>= 1;
        mid = power(x, n, M);
    }
    if (mid == 1 || mid == M - 1) return true;
    return false;
}
bool Miller_Rabin(LL x) {
```

```
    vector<int> prime = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47};
    if (x <= 47) {
        for (int i = 0; i < 15; i++) {
            if (prime[i] == x) {
                return true;
            }
        }
        return false;
    }
    for (int i = 0; i < 15; i++) {
        if (!check(prime[i], x)) {
            return false;
        }
    }
    return true;
}
```

AndSum

```
int AndSum(int l, int r) {
    int ans = r, pos = 0;
    for(int i = 1; r > 0; i++) {
        if ((l & 1) ^ (r & 1)) pos = i;
        l >>= 1;
        r >>= 1;
    }
    ans >>= pos;
    ans <<= pos;
    return ans;
}
```

OrSum

```
int OrSum(int l, int r) {
    if (l == r) return l;
    int pos = 63;
    while (!((l ^ r) & 1LL << pos)) --pos;
    while (~pos) l |= 1LL << pos--;
    return l;
}
```

XorSum

```
int PreXorSum(int n) {
    int ans = 0;
    for (int i = n / 4 * 4; i <= n; i++) {
        ans ^= i;
    }
    return ans;
}
int XorSum(int l, int r) {
    return PreXorSum(r) ^ PreXorSum(l - 1);
}
```

博弈

巴什博弈

定义：一堆 n 个物品，两个人轮流从中取出不多于 m 个，最后取光者胜，不能继续取的人输；

结论：若 `n % (m + 1) != 0`，则先手必胜，反之先手必输

尼姆博弈

定义：n 堆物品，每堆物品的个数任意，两人轮流取，每次取某堆中不少于 1 个，最后取完者必胜。

结论：将每堆物品的数量全都异或起来，若值为 0，则先手必败，否则先手必胜。

威佐夫博弈

定义：有一堆物品，共 n 个，两人轮流取物，先手可取任意件，但不能不取，也不能把物品取完，之后每次取的物品不能超过上一次的两倍，且至少为 1 件，取走最后一件物品的人获胜。

结论：当且仅当 n 不是斐波那契数时，先手胜。

斐波那契博弈

定义：有两堆物品，数量分别为 a 个和 b 个，两个人轮流取物，每次可以从一堆中取出任意个，也可以从两堆中取出相同数量的物品，每次至少要取一个，最后取完所有物品的人获胜。

结论：若 `abs(a - b) * ((sqrt(5) + 1) / 2) == min(a, b)`成立，则后手获胜，否则先手获胜。

String

KMP

```
vector<int> KMP(string s) {
    int n = s.size() - 1;
    vector<int> nxt(n + 1); // nxt[i] 是字符串 s[1...i] 的最大 border
    for (int i = 2, j = 0; i <= n; i++) {
        while (j > 0 && s[j + 1] != s[i]) {
            j = nxt[j];
        }
        if (s[j + 1] == s[i]) {
            j += 1;
        }
        nxt[i] = j;
    }
    return nxt;
}
vector<array<int, 2>> match(string s, string t) { // t 是不是 s 的子串
    int n = s.size() - 1, m = t.size() - 1;
    auto nxt = KMP(t);
    vector<array<int, 2>> ranges;
    for (int i = 1, j = 0; i <= n; i++) {
        while (j > 0 && t[j + 1] != s[i]) {
            j = nxt[j];
        }
        if (t[j + 1] == s[i]) {
            j += 1;
        }
        if (j == m) {
            ranges.push_back({i - m + 1, i});
        }
    }
    return ranges;
}
```

zFunction

```
vector<int> zFunction(string s) {
    int n = s.size();
```

```
    vector<int> z(n + 1);
    z[0] = n;
    for (int i = 1, j = 1; i < n; i++) {
        z[i] = max(0, min(j + z[j] - i, z[i - j]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > j + z[j]) {
            j = i;
        }
    }
    return z;
}
```

Trie

```
template <typename T>
struct Trie {
    std::vector<std::vector<int>> son;
    std::vector<T> cnt;
    int idx;

    Trie(int n, int m) {
        son.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            son[i].resize(m);
        }
        cnt.resize(n + 1);
        idx = 0;
    }
    void insert(string s) {
        int p = 0;
        for (auto c : s) {
            int u = c - 'a';
            if (!son[p][u]) son[p][u] = ++idx;
            p = son[p][u];
        }
        cnt[p] += 1;
    }
    int count(string s) {
        int p = 0;
        for (auto c : s) {
            int u = c - 'a';
            if (!son[p][u]) return 0;
```

```
            p = son[p][u];
        }
        return cnt[p];
    }
};
```

single_hash

```
using ULL = unsigned long long;
const ULL M = (1ull << 61) - 1;
ULL add(ULL a, ULL b) {
    a += b;
    if (a >= M) {
        a -= M;
    }
    return a;
}
ULL mul(ULL a, ULL b) {
    __uint128_t c = __uint128_t(a) * b;
    return add(c >> 61, c & M);
}

template <typename T, const int P>
struct hash {
    int n;
    std::vector<T> h, hi, p;
    hash(string s) : n(s.size() - 1), h(n + 1), hi(n + 2), p(n + 1) {
        p[0] = 1;
        for (int i = 1; i <= n; i++) {
            p[i] = p[i - 1] * P;
            h[i] = h[i - 1] * P + s[i];
        }
        for (int i = n; i >= 1; i--) {
            hi[i] = hi[i + 1] * P + s[i];
        }
    }
    T get(int l, int r) {
        return h[r] - h[l - 1] * p[r - l + 1];
    }
    T geti(int l, int r) {
        return hi[l] - hi[r + 1] * p[r - l + 1];
    }
    bool ispalindrome(int l, int r) {
        return get(l, r) == geti(l, r);
```

```
    }
    bool same(int l1, int r1, int l2, int r2) {
        return get(l1, r1) == get(l2, r2);
    }
    T MergeFF(int l1, int r1, int l2, int r2) { // Forward and Forward
        return get(l1, r1) * p[r2 - l2 + 1] + get(l2, r2);
    }
    T MergeFR(int l1, int r1, int l2, int r2) { // Forward and reverse
        return get(l1, r1) * p[r2 - l2 + 1] + geti(l2, r2);
    }
    T MergeRF(int l1, int r1, int l2, int r2) { // reverse and Forward
        return geti(l1, r1) * p[r2 - l2 + 1] + get(l2, r2);
    }
    T MergeRR(int l1, int r1, int l2, int r2) { // reverse and reverse
        return geti(l1, r1) * p[r2 - l2 + 1] + geti(l2, r2);
    }
};
using SingleHash = hash<MInt<1410412741>, 131>;
```

Other

MInt

```
template<class T>
constexpr T power(T a, LL b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(LL 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;
}
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) {
        LL 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 = 1000000007;
using Z = MInt<P>;
```

O2 优化

```
#pragma GCC optimize(1)
#pragma GCC optimize(2)
#pragma GCC optimize(3, "Ofast", "inline")
```

mt19937_64

```
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
```

求三角形面积

已知三角形三边长分别为 a, b, c.

```
p = (a + b + c) / 2
area = sqrt(p * (p - a) * (p - b) * (p - c))
```

已知三角形三个顶点坐标 A, B, C.

求出两条边的向量 v1 = (Bx - Ax, By - Ay) = (x1, y2)

```
v2 = (Cx - Ax, Cy - Ay) = (x2, y2)
area = (x1 * y2 - x2 * y1) * 0.5;
area = fabs(area);
```

求多边形面积

已知多边形 n 个顶点 p1, p2, ..., pn, 将多边形面积分解为 area(p1, p2, p3) + area(p1, p3, p4) + ... + area(p1, pn - 1, pn)

```
area = fabs(area)
```

已知两点求直线

已知 P1(X1, Y1) P2(X2, Y2)，求直线 Ax + By + C = 0

```
A = Y2 - Y1
B = X1 - X2
C = X2 * Y1 - X1 * Y2
```

求两直线交点

已知直线 1：A1x + B1y + C1 = 0、直线 2：A2x + B2y + C2 = 0

求两条直线的交点(X, Y)

```
X = (B1 * C2 - B2 * C1) / (A1 * B2 - A2 * B1);
Y = (A1 * C2 - A2 * C1) / (A2 * B1 - A1 * B2);
```

特别的，若 A1 * B2 == A2 * B1，则两直线平行

数位 dp

求区间[1, r]内数字 x 出现次数

```
const int N = 110;
int n, m, k;
int f[N][N], a[N];

int dfs(int len, int s, int lead, int t, int x) {
    if (!len) return s;
```

```
    if (!t && lead && f[len][s] != -1) return f[len][s];
    int maxx = t ? a[len] : 9, ans = 0;
    for (int i = 0; i <= maxx; i++) {
        ans += dfs(len - 1, s + ((lead | i) && i == x), lead | i, t && i ==
maxx, x);
    }
    if (!t && lead) f[len][s] = ans;
    return ans;
}
int dp(int n, int x) {
    if (!n) return 0;
    int len = 0;
    while (n) a[++len] = n % 10, n /= 10;
    return dfs(len, 0, 0, 1, x);
}
int cal(int l, int r, int x) {
    return dp(r, x) - dp(l - 1, x);
}
```

约数个数

$N = p_1^{c1} \times p_2^{c2} \times ... \times p_k^{ck}$

约数个数 $= (c_1 + 1) \times (c_2 + 1) \times ... \times (c_k + 1)$

约数之和

$N = p_1^{c1} \times p_2^{c2} \times ... \times p_k^{ck}$

约数个数 $= (p_1^0 + p_1^1 + ... + p_1^{c1}) \times ... \times (p_k^0 + p_k^1 + ... + p_k^{c1})$