ACM 常用算法模板

therehello

2023 年 9 月 18 日

目录

1 数据结构

1.1 并查集

```
struct dsu {
    int n;
    vector<int> fa, sz;
    dsu(int _n) : n(_n), fa(n + 1), sz(n + 1, 1) {
        iota(fa.begin(), fa.end(), 0);
    }
    int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
    int merge(int x, int y) {
        int fax = find(x), fay = find(y);
        if (fax == fay) return 0;  // 一个集合
        sz[fay] += fax;
        return fa[fax] = fay;  // 合并到哪个集合了
    }
    int size(int x) { return sz[find(x)]; }
};
```

1.2 树状数组

1.2.1 一维

```
template <class T>
struct fenwick {
    int n;
    vector<T> t;
    fenwick(int _n) : n(_n), t(n + 1) {}
    T query(int l, int r) {
        auto query = [&](int pos) {
            T res = 0;
            while (pos) {
                res += t[pos];
                pos -= lowbit(pos);
            }
            return res;
        };
        return query(r) - query(l - 1);
    }
    void add(int pos, T num) {
        while (pos <= n) {
            t[pos] += num;
            pos += lowbit(pos);
        }
    }
};
```

1.2.2 二维

```cpp
template <class T>
struct Fenwick_tree_2 {
    Fenwick_tree_2(int n, int m) : n(n), m(m), tree(n + 1, vector<T>(m + 1)) {}
    T query(int l1, int r1, int l2, int r2) {
        auto query = [&](int l, int r) {
            T res = 0;
            for (int i = l; i; i -= lowbit(i))
                for (int j = r; j; j -= lowbit(j)) res += tree[i][j];
            return res;
        };
        return query(l2, r2) - query(l2, r1 - 1) - query(l1 - 1, r2) +
               query(l1 - 1, r1 - 1);
    }
    void update(int x, int y, T num) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= m; j += lowbit(j)) tree[i][j] += num;
    }
private:
    int n, m;
    vector<vector<T>> tree;
};
```

1.2.3 三维

```cpp
template <class T>
struct Fenwick_tree_3 {
    Fenwick_tree_3(int n, int m, int k)
        : n(n),
          m(m),
          k(k),
          tree(n + 1, vector<vector<T>>(m + 1, vector<T>(k + 1))) {}
    T query(int a, int b, int c, int d, int e, int f) {
        auto query = [&](int x, int y, int z) {
            T res = 0;
            for (int i = x; i; i -= lowbit(i))
                for (int j = y; j; j -= lowbit(j))
                    for (int p = z; p; p -= lowbit(p)) res += tree[i][j][p];
            return res;
        };
        T res = query(d, e, f);
        res -= query(a - 1, e, f) + query(d, b - 1, f) + query(d, e, c - 1);
        res += query(a - 1, b - 1, f) + query(a - 1, e, c - 1) +
               query(d, b - 1, c - 1);
        res -= query(a - 1, b - 1, c - 1);
        return res;
    }
    void update(int x, int y, int z, T num) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= m; j += lowbit(j))
                for (int p = z; p <= k; p += lowbit(p)) tree[i][j][p] += num;
```

```
    }
private:
    int n, m, k;
    vector<vector<vector<T>>> tree;
};
```

1.3 线段树

```
template <class Data, class Num>
struct Segment_Tree {
    inline void update(int l, int r, Num x) { update(1, l, r, x); }
    inline Data query(int l, int r) { return query(1, l, r); }
    Segment_Tree(vector<Data>& a) {
        n = a.size();
        tree.assign(n * 4 + 1, {});
        build(a, 1, 1, n);
    }
private:
    int n;
    struct Tree {
        int l, r;
        Data data;
    };
    vector<Tree> tree;
    inline void pushup(int pos) {
        tree[pos].data = tree[pos << 1].data + tree[pos << 1 | 1].data;
    }
    inline void pushdown(int pos) {
        tree[pos << 1].data = tree[pos << 1].data + tree[pos].data.lazytag;
        tree[pos << 1 | 1].data =
            tree[pos << 1 | 1].data + tree[pos].data.lazytag;
        tree[pos].data.lazytag = Num::zero();
    }
    void build(vector<Data>& a, int pos, int l, int r) {
        tree[pos].l = l;
        tree[pos].r = r;
        if (l == r) {
            tree[pos].data = a[l - 1];
            return;
        }
        int mid = (tree[pos].l + tree[pos].r) >> 1;
        build(a, pos << 1, l, mid);
        build(a, pos << 1 | 1, mid + 1, r);
        pushup(pos);
    }
    void update(int pos, int& l, int& r, Num& x) {
        if (l > tree[pos].r || r < tree[pos].l) return;
        if (l <= tree[pos].l && tree[pos].r <= r) {
            tree[pos].data = tree[pos].data + x;
            return;
        }
```

```
        pushdown(pos);
        update(pos << 1, l, r, x);
        update(pos << 1 | 1, l, r, x);
        pushup(pos);
    }
    Data query(int pos, int& l, int& r) {
        if (l > tree[pos].r || r < tree[pos].l) return Data::zero();
        if (l <= tree[pos].l && tree[pos].r <= r) return tree[pos].data;
        pushdown(pos);
        return query(pos << 1, l, r) + query(pos << 1 | 1, l, r);
    }
};
struct Num {
    ll add;
    inline static Num zero() { return {0}; }
    inline Num operator+(Num b) { return {add + b.add}; }
};
struct Data {
    ll sum, len;
    Num lazytag;
    inline static Data zero() { return {0, 0, Num::zero()}; }
    inline Data operator+(Num b) {
        return {sum + len * b.add, len, lazytag + b};
    }
    inline Data operator+(Data b) {
        return {sum + b.sum, len + b.len, Num::zero()};
    }
};
```

1.4 普通平衡树

1.4.1 树状数组实现

需要预先处理出来所有可能的数。

```
template <typename T>
struct treap {
    int n, size;
    vector<int> t;
    vector<T> t2, S;
    treap(const vector<T>& b) {
        S = b;
        sort(S.begin(), S.end());
        S.erase(unique(S.begin(), S.end()), S.end());
        n = S.size();
        size = 0;
        t = vector<int>(n + 1);
        t2 = vector<T>(n + 1);
    }
    int pos(T x) { return lower_bound(S.begin(), S.end(), x) - S.begin() + 1; }
    int sum(int pos) {
        int res = 0;
```

```
        while (pos) {
            res += t[pos];
            pos -= lowbit(pos);
        }
        return res;
    }

    // 插入cnt个x
    void insert(T x, int cnt) {
        size += cnt;
        for (int i = pos(x); i <= n; i += lowbit(i)) {
            t[i] += cnt;
            t2[i] += cnt * x;
        }
    }

    // 删除cnt个x
    void erase(T x, int cnt) { insert(x, -cnt); }

    // x的排名
    int rank(T x) { return sum(pos(x) - 1) + 1; }

    // 统计出现次数
    int count(T x) { return sum(pos(x)) - sum(pos(x) - 1); }

    // 第k小
    T kth(int k) {
        int cnt = 0, x = 0;
        for (int i = log2(n); i >= 0; i--) {
            x += 1 << i;
            if (x >= n || cnt + t[x] >= k) x -= 1 << i;
            else cnt += t[x];
        }
        return S[x];
    }

    // 前k小的数之和
    T pre_sum(int k) {
        int cnt = 0, x = 0;
        T res = 0;
        for (int i = log2(n); i >= 0; i--) {
            x += 1 << i;
            if (x >= n || cnt + t[x] >= k) x -= 1 << i;
            else {
                cnt += t[x];
                res += t2[x];
            }
        }
        return res + (k - cnt) * S[x];
    }

    // 小于x，最大的数
```

```cpp
    T prev(int x) { return kth(sum(pos(x) - 1)); }

    // 大于x, 最小的数
    T next(int x) { return kth(sum(pos(x)) + 1); }
};
```

1.5 可持久化线段树

```cpp
constexpr int MAXN = 200000;
vector<int> root(MAXN << 5);
struct Persistent_seg {
    int n;
    struct Data {
        int ls, rs;
        int val;
    };
    vector<Data> tree;
    Persistent_seg(int n, vector<int>& a) : n(n) { root[0] = build(1, n, a); }
    int build(int l, int r, vector<int>& a) {
        if (l == r) {
            tree.push_back({0, 0, a[l]});
            return tree.size() - 1;
        }
        int mid = l + r >> 1;
        int ls = build(l, mid, a), rs = build(mid + 1, r, a);
        tree.push_back({ls, rs, tree[ls].val + tree[rs].val});
        return tree.size() - 1;
    }
    int update(int rt, const int& idx, const int& val, int l, int r) {
        if (l == r) {
            tree.push_back({0, 0, tree[rt].val + val});
            return tree.size() - 1;
        }
        int mid = l + r >> 1, ls = tree[rt].ls, rs = tree[rt].rs;
        if (idx <= mid) ls = update(ls, idx, val, l, mid);
        else rs = update(rs, idx, val, mid + 1, r);
        tree.push_back({ls, rs, tree[ls].val + tree[rs].val});
        return tree.size() - 1;
    }
    int query(int rt1, int rt2, int k, int l, int r) {
        if (l == r) return l;
        int mid = l + r >> 1;
        int lcnt = tree[tree[rt2].ls].val - tree[tree[rt1].ls].val;
        if (k <= lcnt) return query(tree[rt1].ls, tree[rt2].ls, k, l, mid);
        else return query(tree[rt1].rs, tree[rt2].rs, k - lcnt, mid + 1, r);
    }
};
```

1.6 st 表

```cpp
auto lg = []() {
    array<int, 10000001> lg;
    lg[1] = 0;
    for (int i = 2; i <= 10000000; i++) lg[i] = lg[i >> 1] + 1;
    return lg;
}();
template <typename T>
struct st {
    int n;
    vector<vector<T>> a;
    st(vector<T>& _a) : n(_a.size()) {
        a.assign(lg[n] + 1, vector<int>(n));
        for (int i = 0; i < n; i++) a[0][i] = _a[i];
        for (int j = 1; j <= lg[n]; j++)
            for (int i = 0; i + (1 << j) - 1 < n; i++)
                a[j][i] = max(a[j - 1][i], a[j - 1][i + (1 << (j - 1))]);
    }
    T query(int l, int r) {
        int k = lg[r - l + 1];
        return max(a[k][l], a[k][r - (1 << k) + 1]);
    }
};
```

2 图论

存图

```
struct Graph {
    int n;
    struct Edge {
        int to, w;
    };
    vector<vector<Edge>> graph;
    Graph(int _n) {
        n = _n;
        graph.assign(n + 1, vector<Edge>());
    };
    void add(int u, int v, int w) { graph[u].push_back({v, w}); }
};
```

2.1 最短路

2.1.1 dijkstra

```
void dij(Graph& graph, vector<int>& dis, int t) {
    vector<int> visit(graph.n + 1, 0);
    priority_queue<pair<int, int>> que;
    dis[t] = 0;
    que.emplace(0, t);
    while (!que.empty()) {
        int u = que.top().second;
        que.pop();
        if (visit[u]) continue;
        visit[u] = 1;
        for (auto& [to, w] : graph.graph[u]) {
            if (dis[to] > dis[u] + w) {
                dis[to] = dis[u] + w;
                que.emplace(-dis[to], to);
            }
        }
    }
}
```

2.2 树上问题

2.2.1 最近公公祖先

倍增法

```
vector<int> dep;
vector<array<int, 21>> fa;
dep.assign(n + 1, 0);
fa.assign(n + 1, array<int, 21>{});
void binary_jump(int root) {
    function<void(int)> dfs = [&](int t) {
```

```cpp
        dep[t] = dep[fa[t][0]] + 1;
        for (auto& [to] : graph[t]) {
            if (to == fa[t][0]) continue;
            fa[to][0] = t;
            dfs(to);
        }
    };
    dfs(root);
    for (int j = 1; j <= 20; j++)
        for (int i = 1; i <= n; i++) fa[i][j] = fa[fa[i][j - 1]][j - 1];
}
int lca(int x, int y) {
    if (dep[x] < dep[y]) swap(x, y);
    for (int i = 20; i >= 0; i--)
        if (dep[fa[x][i]] >= dep[y]) x = fa[x][i];
    if (x == y) return x;
    for (int i = 20; i >= 0; i--) {
        if (fa[x][i] != fa[y][i]) {
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
}
```

树剖

```cpp
int lca(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    if (dep[x] < dep[y]) swap(x, y);
    return y;
}
```

2.2.2 树链剖分

```cpp
vector<int> fa, siz, dep, son, dfn, rnk, top;
fa.assign(n + 1, 0);
siz.assign(n + 1, 0);
dep.assign(n + 1, 0);
son.assign(n + 1, 0);
dfn.assign(n + 1, 0);
rnk.assign(n + 1, 0);
top.assign(n + 1, 0);
void hld(int root) {
    function<void(int)> dfs1 = [&](int t) {
        dep[t] = dep[fa[t]] + 1;
        siz[t] = 1;
        for (auto& [to, w] : graph[t]) {
            if (to == fa[t]) continue;
```

```
            fa[to] = t;
            dfs1(to);
            if (siz[son[t]] < siz[to]) son[t] = to;
            siz[t] += siz[to];
        }
    };
    dfs1(root);
    int dfn_tail = 0;
    for (int i = 1; i <= n; i++) top[i] = i;
    function<void(int)> dfs2 = [&](int t) {
        dfn[t] = ++dfn_tail;
        rnk[dfn_tail] = t;
        if (!son[t]) return;
        top[son[t]] = top[t];
        dfs2(son[t]);
        for (auto& [to, w] : graph[t]) {
            if (to == fa[t] || to == son[t]) continue;
            dfs2(to);
        }
    };
    dfs2(root);
}
```

2.3 强连通分量

```
void tarjan(Graph& g1, Graph& g2) {
    int dfn_tail = 0, cnt = 0;
    vector<int> dfn(g1.n + 1, 0), low(g1.n + 1, 0), exist(g1.n + 1, 0),
        belong(g1.n + 1, 0);
    stack<int> sta;
    function<void(int)> dfs = [&](int t) {
        dfn[t] = low[t] = ++dfn_tail;
        sta.push(t);
        exist[t] = 1;
        for (auto& [to] : g1.graph[t])
            if (!dfn[to]) {
                dfs(to);
                low[t] = min(low[t], low[to]);
            } else if (exist[to]) low[t] = min(low[t], dfn[to]);
        if (dfn[t] == low[t]) {
            cnt++;
            while (int temp = sta.top()) {
                belong[temp] = cnt;
                exist[temp] = 0;
                sta.pop();
                if (temp == t) break;
            }
        }
    };
    for (int i = 1; i <= g1.n; i++)
        if (!dfn[i]) dfs(i);
```

```
    g2 = Graph(cnt);
    for (int i = 1; i <= g1.n; i++) g2.w[belong[i]] += g1.w[i];
    for (int i = 1; i <= g1.n; i++)
        for (auto& [to] : g1.graph[i])
            if (belong[i] != belong[to]) g2.add(belong[i], belong[to]);
}
```

2.4 拓扑排序

```
void toposort(Graph& g, vector<int>& dis) {
    vector<int> in(g.n + 1, 0);
    for (int i = 1; i <= g.n; i++)
        for (auto& [to] : g.graph[i]) in[to]++;
    queue<int> que;
    for (int i = 1; i <= g.n; i++)
        if (!in[i]) {
            que.push(i);
            dis[i] = g.w[i];  // dp
        }
    while (!que.empty()) {
        int u = que.front();
        que.pop();
        for (auto& [to] : g.graph[u]) {
            in[to]--;
            dis[to] = max(dis[to], dis[u] + g.w[to]);  // dp
            if (!in[to]) que.push(to);
        }
    }
}
```

3 字符串

3.1 kmp

```
auto kmp(string& s) {
    vector next(s.size(), -1);
    for (int i = 1, j = -1; i < s.size(); i++) {
        while (j >= 0 && s[i] != s[j + 1]) j = next[j];
        if (s[i] == s[j + 1]) j++;
        next[i] = j;
    }
    // next 意为长度
    for (auto& i : next) i++;
    return next;
}
```

3.2 哈希

```
constexpr int N = 2e6;
constexpr ll mod[2] = {2000000011, 2000000033}, base[2] = {20011, 20033};
vector<array<ll, 2>> pow_base(N);

pow_base[0][0] = pow_base[0][1] = 1;
for (int i = 1; i < N; i++) {
    pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
    pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
}

struct Hash {
    int size;
    vector<array<ll, 2>> hash;
    Hash() {}
    Hash(const string& s) {
        size = s.size();
        hash.resize(size);
        hash[0][0] = hash[0][1] = s[0];
        for (int i = 1; i < size; i++) {
            hash[i][0] = (hash[i - 1][0] * base[0] + s[i]) % mod[0];
            hash[i][1] = (hash[i - 1][1] * base[1] + s[i]) % mod[1];
        }
    }
    array<ll, 2> operator[](const array<int, 2>& range) const {
        int l = range[0], r = range[1];
        if (l == 0) return hash[r];
        auto single_hash = [&](bool flag) {
            return (hash[r][flag] -
                    hash[l - 1][flag] * pow_base[r - l + 1][flag] % mod[flag] +
                    mod[flag]) %
                   mod[flag];
        };
        return {single_hash(0), single_hash(1)};
```

```
    }
};
```

3.3 manacher

```
void manacher(const string& _s, vector<int>& r) {
    string s(_s.size() * 2 + 1, '$');
    for (int i = 0; i < _s.size(); i++) s[2 * i + 1] = _s[i];
    r.resize(_s.size() * 2 + 1);
    for (int i = 0, maxr = 0, mid = 0; i < s.size(); i++) {
        if (i < maxr) r[i] = min(r[mid * 2 - i], maxr - i);
        while (i - r[i] - 1 >= 0 && i + r[i] + 1 < s.size() &&
               s[i - r[i] - 1] == s[i + r[i] + 1])
            ++r[i];
        if (i + r[i] > maxr) maxr = i + r[i], mid = i;
    }
}
```

4 数学

4.1 扩展欧几里得

需保证 $a, b >= 0$

$x = x + k * dx, y = y - k * dy$

若要求 $x \geq p$，$k \geq \lceil \frac{p-x}{dx} \rceil$

若要求 $x \leq q$，$k \leq \lfloor \frac{q-x}{dx} \rfloor$

若要求 $y \geq p$，$k \leq \lfloor \frac{y-p}{dy} \rfloor$

若要求 $y \leq q$，$k \geq \lceil \frac{y-q}{dy} \rceil$

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

array<int, 2> exgcd(int a, int b, int c) {
    int x, y;
    int g = __exgcd(a, b, x, y);
    if (c % g) return {INT_MAX, INT_MAX};
    int dx = b / g;
    int dy = a / g;
    x = c / g % dx * x % dx;
    if (x < 0) x += dx;
    y = (c - a * x) / b;
    return {x, y};
}
```

4.2 线性筛法

```cpp
constexpr int N = 10000000;
array<int, N + 1> min_prime;
vector<int> primes;
bool ok = []() {
    for (int i = 2; i <= N; i++) {
        if (min_prime[i] == 0) {
            min_prime[i] = i;
            primes.push_back(i);
        }
        for (auto& j : primes) {
            if (j > min_prime[i] || j > N / i) break;
            min_prime[j * i] = j;
        }
    }
    return 1;
```

```
}();
```

4.3 分解质因数

```
auto getprimes(int n) {
    vector<array<int, 2>> res;
    for (auto& i : primes) {
        if (i > n / i) break;
        if (n % i == 0) {
            res.push_back({i, 0});
            while (n % i == 0) {
                n /= i;
                res.back()[1]++;
            }
        }
    }
    if (n > 1) res.push_back({n, 1});
    return res;
}
```

4.4 pollard rho

```
using LL = __int128_t;

random_device rd;
mt19937 seed(rd());

ll power(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = (LL)res * a % mod;
        a = (LL)a * a % mod;
        b >>= 1;
    }
    return res;
}

bool isprime(ll n) {
    static array primes{2, 3, 5, 7, 11, 13, 17, 19, 23};
    static unordered_map<ll, bool> S;
    if (n < 2) return 0;
    if (S.count(n)) return S[n];
    ll d = n - 1, r = 0;
    while (!(d & 1)) {
        r++;
        d >>= 1;
    }
    for (auto& a : primes) {
        if (a == n) return S[n] = 1;
        ll x = power(a, d, n);
```

```
        if (x == 1 || x == n - 1) continue;
        for (int i = 0; i < r - 1; i++) {
            x = (LL)x * x % n;
            if (x == n - 1) break;
        }
        if (x != n - 1) return S[n] = 0;
    }
    return S[n] = 1;
}

ll pollard_rho(ll n) {
    ll s = 0, t = 0;
    ll c = seed() % (n - 1) + 1;
    ll val = 1;
    for (int goal = 1;; goal *= 2, s = t, val = 1) {
        for (int step = 1; step <= goal; step++) {
            t = ((LL)t * t + c) % n;
            val = (LL)val * abs(t - s) % n;
            if (step % 127 == 0) {
                ll g = gcd(val, n);
                if (g > 1) return g;
            }
        }
        ll g = gcd(val, n);
        if (g > 1) return g;
    }
}
auto getprimes(ll n) {
    unordered_set<ll> S;
    auto get = [&](auto self, ll n) {
        if (n < 2) return;
        if (isprime(n)) {
            S.insert(n);
            return;
        }
        ll mx = pollard_rho(n);
        self(self, n / mx);
        self(self, mx);
    };
    get(get, n);
    return S;
}
```

4.5 组合数

```
constexpr int N = 1e7;
array<modint, N + 1> fac, ifac;
auto _ = []() {
    fac[0] = 1;
    for (int i = 1; i <= N; i++) fac[i] = fac[i - 1] * i;
    ifac[N] = fac[N].inv();
```

```
    for (int i = N - 1; i >= 0; i--) ifac[i] = ifac[i + 1] * (i + 1);
    return true;
}();

modint C(int n, int m) {
    if (n < m) return 0;
    if (n <= mod) return fac[n] * ifac[m] * ifac[n - m];
    // n >= mod 时需要这个
    return C(n % mod, m % mod) * C(n / mod, m / mod);
}
```

4.6 数论分块

求解形如 $\sum_{i=1}^{n} f(i)g(\lfloor \frac{n}{i} \rfloor)$ 的合式

$s(n) = \sum_{i=1}^{n} f(i)$

```
modint sqrt_decomposition(int n) {
    auto s = [&](int x) { return x; };
    auto g = [&](int x) { return x; };
    modint res = 0;
    while (l <= R) {
        int r = n / (n / l);
        res = res + (s(r) - s(l - 1)) * g(n / l);
        l = r + 1;
    }
    return res;
}
```

4.7 积性函数

4.7.1 定义

函数 $f(n)$ 满足 $f(1) = 1$ 且 $\forall x, y \in \mathbf{N}^*, \gcd(x, y) = 1$ 都有 $f(xy) = f(x)f(y)$，则 $f(n)$ 为积性函数。

函数 $f(n)$ 满足 $f(1) = 1$ 且 $\forall x, y \in \mathbf{N}^*$ 都有 $f(xy) = f(x)f(y)$，则 $f(n)$ 为完全积性函数。

4.7.2 例子

- 单位函数：$\varepsilon(n) = [n = 1]$。(完全积性)
- 恒等函数：$\mathrm{id}_k(n) = n^k$。(完全积性)
- 常数函数：$1(n) = 1$。(完全积性)
- 除数函数：$\sigma_k(n) = \sum_{d|n} d^k$。$\sigma_0(n)$ 通常简记作 $d(n)$ 或 $\tau(n)$，$\sigma_1(n)$ 通常简记作 $\sigma(n)$。
- 欧拉函数：$\varphi(n) = \sum_{i=1}^{n} [\gcd(i, n) = 1]$。
- 莫比乌斯函数：$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 \mid n \\ (-1)^{\omega(n)} & \text{otherwise} \end{cases}$，其中 $\omega(n)$ 表示 n 的本质不同质因子个数，它是一个加性函数。

4.8 狄利克雷卷积

对于两个数论函数 $f(x)$ 和 $g(x)$，则它们的狄利克雷卷积得到的结果 $h(x)$ 定义为：
$h(x)=\sum_{d|x}f(d)g\left(\frac{x}{d}\right)=\sum_{ab=x}f(a)g(b)$
可以简记为：$h=f*g$。

4.8.1 性质

交换律： $f*g=g*f$。
结合律： $(f*g)*h=f*(g*h)$。
分配律： $(f+g)*h=f*h+g*h$。
等式的性质： $f=g$ 的充要条件是 $f*h=g*h$，其中数论函数 $h(x)$ 要满足 $h(1)\neq 0$。

4.8.2 例子

- $\varepsilon=\mu*1 \iff \varepsilon(n)=\sum_{d|n}\mu(d)$
- $id=\varphi*1 \iff id(n)=\sum_{d|n}\varphi(d)$
- $d=1*1 \iff d(n)=\sum_{d|n}1$
- $\sigma=\mathrm{id}*1 \iff \sigma(n)=\sum_{d|n}d$
- $\varphi=\mu*\mathrm{id} \iff \varphi(n)=\sum_{d|n}d\cdot\mu(\frac{n}{d})$

4.9 欧拉函数

```cpp
array<int, N + 1> phi;
auto _ = []() {
    iota(phi.begin() + 1, phi.end(), 1);
    for (int i = 2; i <= N; i++) {
        if (phi[i] == i)
            for (int j = i; j <= N; j += i) phi[j] = phi[j] / i * (i - 1);
    }
    return true;
}();
```

4.10 莫比乌斯反演

4.10.1 莫比乌斯函数性质

- $\sum_{d|n}\mu(d)=\begin{cases}1 & n=1\\0 & n\neq 1\end{cases}$，即 $\sum_{d|n}\mu(d)=\varepsilon(n)$，$\mu*1=\varepsilon$
- $[\gcd(i,j)=1]=\sum\limits_{d|\gcd(i,j)}\mu(d)$

```cpp
array<int, N + 1> miu;
array<bool, N + 1> ispr;
auto _ = []() {
    miu.fill(1);
    ispr.fill(1);
```

```
    for (int i = 2; i <= N; i++) {
        if (!ispr[i]) continue;
        miu[i] = -1;
        for (int j = 2 * i; j <= N; j += i) {
            ispr[j] = 0;
            if ((j / i) % i == 0) miu[j] = 0;
            else miu[j] *= -1;
        }
    }
    return true;
}();
```

4.10.2 莫比乌斯变换/反演

$f(n)=\sum_{d|n}g(d)$，那么有 $g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})=\sum_{n|d}\mu(\frac{d}{n})f(d)$ 。
用狄利克雷卷积表示则为 $f=g*1$，有 $g=f*\mu$。
$f\to g$ 称为莫比乌斯反演，$g\to f$ 称为莫比乌斯反演。

4.11 杜教筛

杜教筛被用于处理一类数论函数的前缀和问题。对于数论函数 f，杜教筛可以在低于线性时间的复杂度内计算 $S(n)=\sum_{i=1}^{n}f(i)$。

$$S(n)=\frac{\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)}{g(1)}$$

可以构造恰当的数论函数 g 使得：

- 可以快速计算 $\sum_{i=1}^{n}(f*g)(i)$。
- 可以快速计算 g 的单点值，用数论分块求解 $\sum_{i=2}^{n}g(i)S\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$。

4.11.1 示例

```
ll sum_phi(ll n) {
    if (n <= N) return sp[n];
    if (sp2.count(n)) return sp2[n];
    ll res = 0, l = 2;
    while (l <= n) {
        ll r = n / (n / l);
        res = res + (r - l + 1) * sum_phi(n / l);
        l = r + 1;
    }
    return sp2[n] = (ll)n * (n + 1) / 2 - res;
}
ll sum_miu(ll n) {
    if (n <= N) return sm[n];
    if (sm2.count(n)) return sm2[n];
    ll res = 0, l = 2;
    while (l <= n) {
        ll r = n / (n / l);
        res = res + (r - l + 1) * sum_miu(n / l);
```

```
        l = r + 1;
    }
    return sm2[n] = 1 - res;
}
```

4.12 盒子与球

n 个球，m 个盒

球同	盒同	可空	公式
✓	✓	✓	$f_{n,m} = f_{n-1,m-1} + f_{n-m,m}$
✓	✓	×	$f_{n-m,m}$
×	✓	✓	$\Sigma_{i=1}^{m} g_{n,i}$
×	✓	×	$g_{n,m} = g_{n-1,m-1} + m * g_{n-1,m}$
✓	×	✓	C_{n+m-1}^{m-1}
✓	×	×	C_{n-1}^{m-1}
×	×	✓	m^n
×	×	×	$m! * g_{n,m}$

扩展：

n 个相同的球，m 个不同的盒，每个盒子超过 k 个球，问方案数？

可以考虑容斥，$f(d)$ 表示至少有 d 个盒子装了 $> k$ 个球方案数，总方案数则为 $f(0) - f(1) + f(2) - \ldots$

4.13 线性基

```
// 线性基
struct basis {
    int rnk = 0;
    array<ull, 64> p{};

    // 将x插入此线性基中
    void insert(ull x) {
        for (int i = 63; i >= 0; i--) {
            if ((x >> i) & 1) {
                if (p[i]) x ^= p[i];
                else {
                    for (int j = 0; j < i; j++)
                        if (x >> j & 1) x ^= p[j];
                    for (int j = i + 1; j <= 63; j++)
                        if (p[j] >> i & 1) p[j] ^= x;
                    p[i] = x;
                    rnk++;
                    break;
                }
            }
```

```
        }
    }

    // 将另一个线性基插入此线性基中
    void insert(basis other) {
        for (int i = 0; i <= 63; i++) {
            if (!other.p[i]) continue;
            insert(other.p[i]);
        }
    }

    // 最大异或值
    ull max_basis() {
        ull res = 0;
        for (int i = 63; i >= 0; i--)
            if ((res ^ p[i]) > res) res ^= p[i];
        return res;
    }
};
```

4.14 矩阵快速幂

```
constexpr ll mod = 2147493647;
struct Mat {
    int n, m;
    vector<vector<ll>> mat;
    Mat(int n, int m) : n(n), m(n), mat(n, vector<ll>(m, 0)) {}
    Mat(vector<vector<ll>> mat) : n(mat.size()), m(mat[0].size()), mat(mat) {}
    Mat operator*(const Mat& other) {
        assert(m == other.n);
        Mat res(n, other.m);
        for (int i = 0; i < res.n; i++)
            for (int j = 0; j < res.m; j++)
                for (int k = 0; k < m; k++)
                    res.mat[i][j] =
                        (res.mat[i][j] + mat[i][k] * other.mat[k][j] % mod) %
                        mod;
        return res;
    }
};
Mat ksm(Mat a, ll b) {
    assert(a.n == a.m);
    Mat res(a.n, a.m);
    for (int i = 0; i < res.n; i++) res.mat[i][i] = 1;
    while (b) {
        if (b & 1) res = res * a;
        b >>= 1;
        a = a * a;
    }
    return res;
}
```

5 计算几何

5.1 整数

```
const double PI = acos(-1);
constexpr double eps = 1e-8;

// 向量
struct vec {
    static bool cmp(const vec &a, const vec &b) {
        return tie(a.x, a.y) < tie(b.x, b.y);
    }

    ll x, y;
    vec(ll _x = 0, ll _y = 0) : x(_x), y(_y) {}

    // 模
    ll len2() const { return x * x + y * y; }
    double len() const { return sqrt(x * x + y * y); }

    // 是否在上半轴
    bool up() const { return y > 0 || y == 0 && x >= 0; }

    bool operator==(const vec &b) const { return tie(x, y) == tie(b.x, b.y); }

    // 极角排序
    bool operator<(const vec &b) const {
        if (up() != b.up()) return up() > b.up();
        ll tmp = (*this) ^ b;
        return tmp ? tmp > 0 : cmp(*this, b);
    }

    vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
    vec operator-() const { return {-x, -y}; }
    vec operator-(const vec &b) const { return -b + (*this); }
    vec operator*(ll b) const { return {x * b, y * b}; }
    ll operator*(const vec &b) const { return x * b.x + y * b.y; }

    // 叉积 结果大于0，a到b为逆时针，小于0，a到b顺时针，
    // 等于0共线，可能同向或反向，结果绝对值表示 a b 形成的平行四边行的面积
    ll operator^(const vec &other) const { return x * other.y - y * other.x; }

    friend istream &operator>>(istream &in, vec &data) {
        in >> data.x >> data.y;
        return in;
    }
    friend ostream &operator<<(ostream &out, const vec &data) {
        out << fixed << setprecision(6);
        out << data.x << " " << data.y;
        return out;
    }
};
```

```

ll cross(const vec &a, const vec &b, const vec &c) { return (a - c) ^ (b - c); }

// 判断点是否在凸包内
bool in_polygon(const vec &a, vector<vec> &p) {
    int n = p.size();
    if (n == 0) return 0;
    if (n == 1) return a == p[0];
    if (cross(a, p[1], p[0]) > 0 || cross(p.back(), a, p[0]) > 0) return 0;
    auto cmp = [&p](vec &x, const vec &y) { return ((x - p[0]) ^ y) >= 0; };
    int i = lower_bound(p.begin() + 2, p.end(), a - p[0], cmp) - p.begin() - 1;
    return cross(p[(i + 1) % n], a, p[i]) >= 0;
}

// 多边形的面积
double polygon_area(vector<vec> &p) {
    ll area = 0;
    for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
    area += p.back() ^ p[0];
    return abs(area / 2.0);
}

// 多边形的周长
double polygon_length(vector<vec> &p) {
    double len = 0;
    for (int i = 1; i < p.size(); i++) len += (p[i - 1] - p[i]).len();
    len += (p.back() - p[0]).len();
    return len;
}

// 以整点为顶点的线段上的整点个数
ll count(const vec &a, const vec &b) {
    vec c = a - b;
    return gcd(abs(c.x), abs(c.y)) + 1;
}

// 以整点为顶点的多边形边上整点个数
ll count(vector<vec> &p) {
    ll cnt = 0;
    for (int i = 1; i < p.size(); i++) cnt += count(p[i - 1], p[i]);
    cnt += count(p.back(), p[0]);
    return cnt - p.size();
}

// 凸包直径的两个端点
auto polygon_dia(vector<vec> &p) {
    int n = p.size();
    array<vec, 2> res{};
    if (n == 1) return res;
    if (n == 2) return res = {p[0], p[1]};
    ll mx = 0;
    for (int i = 0, j = 2; i < n; i++) {
```

```
        while (abs(cross(p[i], p[(i + 1) % n], p[j])) <=
               abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n])))
            j = (j + 1) % n;
        ll tmp = (p[i] - p[j]).len2();
        if (tmp > mx) {
            mx = tmp;
            res = {p[i], p[j]};
        }
        tmp = (p[(i + 1) % n] - p[j]).len2();
        if (tmp > mx) {
            mx = tmp;
            res = {p[(i + 1) % n], p[j]};
        }
    }
    return res;
}

// 凸包
auto convex_hull(vector<vec> &p) {
    sort(p.begin(), p.end(), vec::cmp);
    int n = p.size();
    vector sta(n + 1, 0);
    vector v(n, false);
    int tp = -1;
    sta[++tp] = 0;
    auto update = [&](int lim, int i) {
        while (tp > lim && cross(p[i], p[sta[tp]], p[sta[tp - 1]]) >= 0)
            v[sta[tp--]] = 0;
        sta[++tp] = i;
        v[i] = 1;
    };
    for (int i = 1; i < n; i++) update(0, i);
    int cnt = tp;
    for (int i = n - 1; i >= 0; i--) {
        if (v[i]) continue;
        update(cnt, i);
    }
    vector<vec> res(tp);
    for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
    return res;
}

// 闵可夫斯基和，两个点集的和构成一个凸包
auto minkowski(vector<vec> &a, vector<vec> &b) {
    rotate(a.begin(), min_element(a.begin(), a.end(), vec::cmp), a.end());
    rotate(b.begin(), min_element(b.begin(), b.end(), vec::cmp), b.end());
    int n = a.size(), m = b.size();
    vector<vec> c{a[0] + b[0]};
    c.reserve(n + m);
    int i = 0, j = 0;
    while (i < n && j < m) {
        vec x = a[(i + 1) % n] - a[i];
```

```
        vec y = b[(j + 1) % m] - b[j];
        c.push_back(c.back() + ((x ^ y) >= 0 ? (i++, x) : (j++, y)));
    }
    while (i + 1 < n) {
        c.push_back(c.back() + a[(i + 1) % n] - a[i]);
        i++;
    }
    while (j + 1 < m) {
        c.push_back(c.back() + b[(j + 1) % m] - b[j]);
        j++;
    }
    return c;
}

// 过凸多边形外一点求凸多边形的切线，返回切点下标
auto tangent(const vec &a, vector<vec> &p) {
    int n = p.size();
    int l = -1, r = -1;
    for (int i = 0; i < n; i++) {
        ll tmp1 = cross(p[i], p[(i - 1 + n) % n], a);
        ll tmp2 = cross(p[i], p[(i + 1) % n], a);
        if (l == -1 && tmp1 <= 0 && tmp2 <= 0) l = i;
        else if (r == -1 && tmp1 >= 0 && tmp2 >= 0) r = i;
    }
    return array{l, r};
}

// 直线
struct line {
    vec p, d;
    line(const vec &_p = vec(), const vec &_d = vec()) : p(_p), d(_d) {}
};

// 点到直线距离
double dis(const vec &a, const line &b) {
    return abs((b.p - a) ^ (b.p + b.d - a)) / b.d.len();
}

// 点在直线哪边，大于0在左边，等于0在线上，小于0在右边
ll side_line(const vec &a, const line &b) { return b.d ^ (a - b.p); }

// 两直线是否垂直
bool perpen(const line &a, const line &b) { return a.d * b.d == 0; }

// 两直线是否平行
bool parallel(const line &a, const line &b) { return (a.d ^ b.d) == 0; }

// 点的垂线是否与线段有交点
bool perpen(const vec &a, const line &b) {
    vec p(-b.d.y, b.d.x);
    bool cross1 = (p ^ (b.p - a)) > 0;
    bool cross2 = (p ^ (b.p + b.d - a)) > 0;
```

```
    return cross1 != cross2;
}

// 点到线段距离
double dis_seg(const vec &a, const line &b) {
    if (perpen(a, b)) return dis(a, b);
    return min(dis(a, b.p), dis(a, b.p + b.d));
}

// 两直线交点
vec intersection(ll A, ll B, ll C, ll D, ll E, ll F) {
    return {(B * F - C * E) / (A * E - B * D),
            (C * D - A * F) / (A * E - B * D)};
}

// 两直线交点
vec intersection(const line &a, const line &b) {
    return intersection(a.d.y, -a.d.x, a.d.x * a.p.y - a.d.y * a.p.x, b.d.y,
                        -b.d.x, b.d.x * b.p.y - b.d.y * b.p.x);
}
```

5.2 浮点数

```
constexpr double eps = 1e-8;
const double PI = acos(-1);

int sgn(double a, double b) {
    double c = a - b;
    return c < -eps ? -1 : c < eps ? 0 : 1;
}

// 向量
struct vec {
    static bool cmp(const vec &a, const vec &b) {
        return sgn(a.x, b.x) ? a.x < b.x : sgn(a.y, b.y) < 0;
    }

    double x, y;
    vec(double _x = 0, double _y = 0) : x(_x), y(_y) {}

    // 模
    double len2() const { return x * x + y * y; }
    double len() const { return sqrt(x * x + y * y); }

    // 与x轴正方向的夹角
    double angle() const {
        double angle = atan2(y, x);
        if (angle < 0) angle += 2 * PI;
        return angle;
    }

```

```cpp
    // 逆时针旋转
    vec rotate(const double &theta) {
        return {x * cos(theta) - y * sin(theta),
                y * cos(theta) + x * sin(theta)};
    }

    bool operator==(const vec &other) const {
        return sgn(x, other.x) == 0 && sgn(y, other.y) == 0;
    }

    // 是否在上半轴
    bool up() const {
        return sgn(y, 0) > 0 || sgn(y, 0) == 0 && sgn(x, 0) >= 0;
    }

    // 极角排序
    bool operator<(const vec &b) const {
        if (up() != b.up()) return up() > b.up();
        double tmp = (*this) ^ b;
        return sgn(tmp, 0) ? tmp > 0 : cmp(*this, b);
    }

    vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
    vec operator-() const { return {-x, -y}; }
    vec operator-(const vec &b) const { return -b + (*this); }
    vec operator*(double b) const { return {x * b, y * b}; }
    vec operator/(double b) const { return {x / b, y / b}; }
    double operator*(const vec &b) const { return x * b.x + y * b.y; }

    // 叉积 结果大于0，a到b为逆时针，小于0，a到b顺时针，
    // 等于0共线，可能同向或反向，结果绝对值表示 a b 形成的平行四边行的面积
    double operator^(const vec &b) const { return x * b.y - y * b.x; }

    friend istream &operator>>(istream &in, vec &data) {
        in >> data.x >> data.y;
        return in;
    }
    friend ostream &operator<<(ostream &out, const vec &data) {
        out << fixed << setprecision(6);
        out << data.x << " " << data.y;
        return out;
    }
};

double cross(const vec &a, const vec &b, const vec &c) {
    return (a - c) ^ (b - c);
}

// 判断点是否在凸包内
bool in_polygon(const vec &a, vector<vec> &p) {
    int n = p.size();
    if (n == 1) return a == p[0];
```

```cpp
    if (sgn(cross(a, p[1], p[0]), 0) > 0 ||
        sgn(cross(p.back(), a, p[0]), 0) > 0)
        return 0;
    auto cmp = [&p](vec &x, const vec &y) {
        return sgn((x - p[0]) ^ y, 0) >= 0;
    };
    int i = lower_bound(p.begin() + 2, p.end(), a - p[0], cmp) - p.begin() - 1;
    return sgn(cross(p[(i + 1) % n], a, p[i]), 0) >= 0;
}

// 多边形的面积
double polygon_area(vector<vec> &p) {
    double area = 0;
    for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
    area += p.back() ^ p[0];
    return abs(area / 2.0);
}

// 多边形的周长
double polygon_length(vector<vec> &p) {
    double len = 0;
    for (int i = 1; i < p.size(); i++) len += (p[i - 1] - p[i]).len();
    len += (p.back() - p[0]).len();
    return len;
}

// 凸包直径的两个端点
auto polygon_dia(vector<vec> &p) {
    int n = p.size();
    array<vec, 2> res{};
    if (n == 1) return res;
    if (n == 2) return res = {p[0], p[1]};
    double mx = 0;
    for (int i = 0, j = 2; i < n; i++) {
        while (sgn(abs(cross(p[i], p[(i + 1) % n], p[j])),
                   abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n]))) <= 0)
            j = (j + 1) % n;
        double tmp = (p[i] - p[j]).len();
        if (tmp > mx) {
            mx = tmp;
            res = {p[i], p[j]};
        }
        tmp = (p[(i + 1) % n] - p[j]).len();
        if (tmp > mx) {
            mx = tmp;
            res = {p[(i + 1) % n], p[j]};
        }
    }
    return res;
}

// 凸包
```

```cpp
auto convex_hull(vector<vec> &p) {
    sort(p.begin(), p.end(), vec::cmp);
    int n = p.size();
    vector sta(n + 1, 0);
    vector v(n, false);
    int tp = -1;
    sta[++tp] = 0;
    auto update = [&](int lim, int i) {
        while (tp > lim && sgn(cross(p[i], p[sta[tp]], p[sta[tp - 1]]), 0) >= 0)
            v[sta[tp--]] = 0;
        sta[++tp] = i;
        v[i] = 1;
    };
    for (int i = 1; i < n; i++) update(0, i);
    int cnt = tp;
    for (int i = n - 1; i >= 0; i--) {
        if (v[i]) continue;
        update(cnt, i);
    }
    vector<vec> res(tp);
    for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
    return res;
}

// 闵可夫斯基和 两个点集的和构成一个凸包
auto minkowski(vector<vec> &a, vector<vec> &b) {
    rotate(a.begin(), min_element(a.begin(), a.end(), vec::cmp), a.end());
    rotate(b.begin(), min_element(b.begin(), b.end(), vec::cmp), b.end());
    int n = a.size(), m = b.size();
    vector<vec> c{a[0] + b[0]};
    c.reserve(n + m);
    int i = 0, j = 0;
    while (i < n && j < m) {
        vec x = a[(i + 1) % n] - a[i];
        vec y = b[(j + 1) % m] - b[j];
        c.push_back(c.back() + (sgn(x ^ y, 0) >= 0 ? (i++, x) : (j++, y)));
    }
    while (i + 1 < n) {
        c.push_back(c.back() + a[(i + 1) % n] - a[i]);
        i++;
    }
    while (j + 1 < m) {
        c.push_back(c.back() + b[(j + 1) % m] - b[j]);
        j++;
    }
    return c;
}

// 过凸多边形外一点求凸多边形的切线，返回切点下标
auto tangent(const vec &a, vector<vec> &p) {
    int n = p.size();
    int l = -1, r = -1;
```

```
    for (int i = 0; i < n; i++) {
        double tmp1 = cross(p[i], p[(i - 1 + n) % n], a);
        double tmp2 = cross(p[i], p[(i + 1) % n], a);
        if (l == -1 && sgn(tmp1, 0) <= 0 && sgn(tmp2, 0) <= 0) l = i;
        else if (r == -1 && sgn(tmp1, 0) >= 0 && sgn(tmp2, 0) >= 0) r = i;
    }
    return array{l, r};
}

// 直线
struct line {
    vec p, d;
    line(const vec &_p = vec(), const vec &_d = vec()) : p(_p), d(_d) {}
};

// 点到直线距离
double dis(const vec &a, const line &b) {
    return abs((b.p - a) ^ (b.p + b.d - a)) / b.d.len();
}

// 判断点在直线哪边，大于0在左边，等于0在线上，小于0在右边
int side_line(const vec &a, const line &b) { return sgn(b.d ^ (a - b.p), 0); }

// 两直线是否垂直
bool perpen(const line &a, const line &b) { return sgn(a.d * b.d, 0) == 0; }

// 两直线是否平行
bool parallel(const line &a, const line &b) { return sgn(a.d ^ b.d, 0) == 0; }

// 点的垂线是否与线段有交点
bool perpen(const vec &a, const line &b) {
    vec p(-b.d.y, b.d.x);
    bool cross1 = (p ^ (b.p - a)) > 0;
    bool cross2 = (p ^ (b.p + b.d - a)) > 0;
    return cross1 != cross2;
}

// 点到线段距离
double disseg(const vec &a, const line &b) {
    if (perpen(a, b)) return dis(a, b);
    return min(dis(a, b.p), dis(a, b.p + b.d));
}

// 两直线交点
vec intersection(double A, double B, double C, double D, double E, double F) {
    return {(B * F - C * E) / (A * E - B * D),
            (C * D - A * F) / (A * E - B * D)};
}

// 两直线交点
vec intersection(const line &a, const line &b) {
    return intersection(a.d.y, -a.d.x, a.d.x * a.p.y - a.d.y * a.p.x, b.d.y,
```

```
                        -b.d.x, b.d.x * b.p.y - b.d.y * b.p.x);
}

struct circle {
    vec o;
    double r;
    circle(const vec &_o, double _r) : o(_o), r(_r){};
    // 点与圆的关系 -1在圆内，0在圆上，1在圆外
    int relation(const vec &a) const {
        double len = (a - o).len();
        return sgn(len, r);
    }
    double area() { return PI * r * r; }
};

// 圆与直线交点
auto intersection(const circle &c, const line &l) {
    double d = dis(c.o, l);
    vector<vec> res;
    double len = l.d.len();
    vec mid = l.p + l.d * ((c.o - l.p) * l.d / len);
    if (sgn(d, c.r) == 0) res.push_back(mid);
    else if (sgn(d, c.r) < 0) {
        d = sqrt(c.r * c.r - d * d) / len;
        res.push_back(mid + l.d * d);
        res.push_back(mid - l.d * d);
    }
    return res;
}
```

5.3 扫描线

```
#define ls (pos << 1)
#define rs (ls | 1)
#define mid ((tree[pos].l + tree[pos].r) >> 1)
struct Rectangle {
    ll x_l, y_l, x_r, y_r;
};
ll area(vector<Rectangle>& rec) {
    struct Line {
        ll x, y_up, y_down;
        int pd;
    };
    vector<Line> line(rec.size() * 2);
    vector<ll> y_set(rec.size() * 2);
    for (int i = 0; i < rec.size(); i++) {
        y_set[i * 2] = rec[i].y_l;
        y_set[i * 2 + 1] = rec[i].y_r;
        line[i * 2] = {rec[i].x_l, rec[i].y_r, rec[i].y_l, 1};
        line[i * 2 + 1] = {rec[i].x_r, rec[i].y_r, rec[i].y_l, -1};
    }
```

```
    sort(y_set.begin(), y_set.end());
    y_set.erase(unique(y_set.begin(), y_set.end()), y_set.end());
    sort(line.begin(), line.end(), [](Line a, Line b) { return a.x < b.x; });
    struct Data {
        int l, r;
        ll len, cnt, raw_len;
    };
    vector<Data> tree(4 * y_set.size());
    function<void(int, int, int)> build = [&](int pos, int l, int r) {
        tree[pos].l = l;
        tree[pos].r = r;
        if (l == r) {
            tree[pos].raw_len = y_set[r + 1] - y_set[l];
            tree[pos].cnt = tree[pos].len = 0;
            return;
        }
        build(ls, l, mid);
        build(rs, mid + 1, r);
        tree[pos].raw_len = tree[ls].raw_len + tree[rs].raw_len;
    };
    function<void(int, int, int, int)> update = [&](int pos, int l, int r,
                                                    int num) {
        if (l <= tree[pos].l && tree[pos].r <= r) {
            tree[pos].cnt += num;
            tree[pos].len = tree[pos].cnt ? tree[pos].raw_len
                            : tree[pos].l == tree[pos].r
                                ? 0
                                : tree[ls].len + tree[rs].len;
            return;
        }
        if (l <= mid) update(ls, l, r, num);
        if (r > mid) update(rs, l, r, num);
        tree[pos].len =
            tree[pos].cnt ? tree[pos].raw_len : tree[ls].len + tree[rs].len;
    };
    build(1, 0, y_set.size() - 2);
    auto find_pos = [&](ll num) {
        return lower_bound(y_set.begin(), y_set.end(), num) - y_set.begin();
    };
    ll res = 0;
    for (int i = 0; i < line.size() - 1; i++) {
        update(1, find_pos(line[i].y_down), find_pos(line[i].y_up) - 1,
               line[i].pd);
        res += (line[i + 1].x - line[i].x) * tree[1].len;
    }
    return res;
}
```

6 杂项

6.1 高精度

```
struct bignum {
    string num;

    bignum() : num("0") {}
    bignum(const string& num) : num(num) {
        reverse(this->num.begin(), this->num.end());
    }
    bignum(ll num) : num(to_string(num)) {
        reverse(this->num.begin(), this->num.end());
    }

    bignum operator+(const bignum& other) {
        bignum res;
        res.num.pop_back();
        res.num.reserve(max(num.size(), other.num.size()) + 1);
        for (int i = 0, j = 0, x; i < num.size() || i < other.num.size() || j;
             i++) {
            x = j;
            j = 0;
            if (i < num.size()) x += num[i] - '0';
            if (i < other.num.size()) x += other.num[i] - '0';
            if (x >= 10) j = 1, x -= 10;
            res.num.push_back(x + '0');
        }
        res.num.capacity();
        return res;
    }

    bignum operator*(const bignum& other) {
        vector<int> res(num.size() + other.num.size() - 1, 0);
        for (int i = 0; i < num.size(); i++)
            for (int j = 0; j < other.num.size(); j++)
                res[i + j] += (num[i] - '0') * (other.num[j] - '0');
        int g = 0;
        for (int i = 0; i < res.size(); i++) {
            res[i] += g;
            g = res[i] / 10;
            res[i] %= 10;
        }
        while (g) {
            res.push_back(g % 10);
            g /= 10;
        }
        int lim = res.size();
        while (lim > 1 && res[lim - 1] == 0) lim--;
        bignum res2;
        res2.num.resize(lim);
        for (int i = 0; i < lim; i++) res2.num[i] = res[i] + '0';
```

```
        return res2;
    }

    bool operator<(const bignum& other) {
        if (num.size() == other.num.size())
            for (int i = num.size() - 1; i >= 0; i--)
                if (num[i] == other.num[i]) continue;
                else return num[i] < other.num[i];
        return num.size() < other.num.size();
    }

    friend istream& operator>>(istream& in, bignum& a) {
        in >> a.num;
        reverse(a.num.begin(), a.num.end());
        return in;
    }
    friend ostream& operator<<(ostream& out, bignum a) {
        reverse(a.num.begin(), a.num.end());
        return out << a.num;
    }
};
```

6.2 模运算

```
struct modint {
    int x;
    modint(ll _x = 0) : x(_x % mod) {}
    modint inv() const { return power(*this, mod - 2); }
    modint operator+(const modint& b) { return {x + b.x}; }
    modint operator-() const { return {-x}; }
    modint operator-(const modint& b) { return {-b + *this}; }
    modint operator*(const modint& b) { return {(ll)x * b.x}; }
    modint operator/(const modint& b) { return *this * b.inv(); }
    friend istream& operator>>(istream& is, modint& other) {
        ll _x;
        is >> _x;
        other = modint(_x);
        return is;
    }
    friend ostream& operator<<(ostream& os, modint other) {
        other.x = (other.x + mod) % mod;
        return os << other.x;
    }
};
```

6.3 分数

```
struct frac {
    ll a, b;
    frac() : a(0), b(1) {}
```

```cpp
    frac(ll _a, ll _b) : a(_a), b(_b) {
        assert(b);
        if (a) {
            int tmp = gcd(a, b);
            a /= tmp;
            b /= tmp;
        } else *this = frac();
    }
    frac operator+(const frac& other) {
        return frac(a * other.b + other.a * b, b * other.b);
    }
    frac operator-() const {
        frac res = *this;
        res.a = -res.a;
        return res;
    }
    frac operator-(const frac& other) const { return -other + *this; }
    frac operator*(const frac& other) const {
        return frac(a * other.a, b * other.b);
    }
    frac operator/(const frac& other) const {
        assert(other.a);
        return *this * frac(other.b, other.a);
    }
    bool operator<(const frac& other) const { return (*this - other).a < 0; }
    bool operator<=(const frac& other) const { return (*this - other).a <= 0; }
    bool operator>=(const frac& other) const { return (*this - other).a >= 0; }
    bool operator>(const frac& other) const { return (*this - other).a > 0; }
    bool operator==(const frac& other) const {
        return a == other.a && b == other.b;
    }
    bool operator!=(const frac& other) const { return !(*this == other); }
};
```

6.4 表达式求值

```cpp
// 格式化表达式
string format(const string& s1) {
    stringstream ss(s1);
    string s2;
    char ch;
    while ((ch = ss.get()) != EOF) {
        if (ch == ' ') continue;
        if (isdigit(ch)) s2 += ch;
        else {
            if (s2.back() != ' ') s2 += ' ';
            s2 += ch;
            s2 += ' ';
        }
    }
    return s2;
```

```
}

// 中缀表达式转后缀表达式
string convert(const string& s1) {
    unordered_map<char, int> rank{
        {'+', 2}, {'-', 2}, {'*', 1}, {'/', 1}, {'^', 0}};
    stringstream ss(s1);
    string s2, temp;
    stack<char> op;
    while (ss >> temp) {
        if (isdigit(temp[0])) s2 += temp + ' ';
        else if (temp[0] == '(') op.push('(');
        else if (temp[0] == ')') {
            while (op.top() != '(') {
                s2 += op.top();
                s2 += ' ';
                op.pop();
            }
            op.pop();
        } else {
            while (!op.empty() && op.top() != '(' &&
                   (temp[0] != '^' && rank[op.top()] <= rank[temp[0]] ||
                    rank[op.top()] < rank[temp[0]])) {
                s2 += op.top();
                s2 += ' ';
                op.pop();
            }
            op.push(temp[0]);
        }
    }
    while (!op.empty()) {
        s2 += op.top();
        s2 += ' ';
        op.pop();
    }
    return s2;
}

// 计算后缀表达式
int calc(const string& s) {
    stack<int> num;
    stringstream ss(s);
    string temp;
    while (ss >> temp) {
        if (isdigit(temp[0])) num.push(stoi(temp));
        else {
            int b = num.top();
            num.pop();
            int a = num.top();
            num.pop();
            if (temp[0] == '+') a += b;
            else if (temp[0] == '-') a -= b;
```

```
            else if (temp[0] == '*') a *= b;
            else if (temp[0] == '/') a /= b;
            else if (temp[0] == '^') a = ksm(a, b);
            num.push(a);
        }
    }
    return num.top();
}
```

6.5 日期

```
int month[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int pre[13];
vector<int> leap;
struct Date {
    int y, m, d;
    bool operator<(const Date& other) const {
        return array<int, 3>{y, m, d} <
               array<int, 3>{other.y, other.m, other.d};
    }
    Date(const string& s) {
        stringstream ss(s);
        char ch;
        ss >> y >> ch >> m >> ch >> d;
    }
    int dis() const {
        int yd = (y - 1) * 365 +
                 (upper_bound(leap.begin(), leap.end(), y - 1) - leap.begin());
        int md =
            pre[m - 1] + (m > 2 && (y % 4 == 0 && y % 100 || y % 400 == 0));
        return yd + md + d;
    }
    int dis(const Date& other) const { return other.dis() - dis(); }
};
for (int i = 1; i <= 12; i++) pre[i] = pre[i - 1] + month[2];
for (int i = 1; i <= 1000000; i++)
    if (i % 4 == 0 && i % 100 || i % 400 == 0) leap.push_back(i);
```

6.6 对拍

linux/Mac

```
g++ a.cpp -o program/a -O2 -std=c++17
g++ b.cpp -o program/b -O2 -std=c++17
g++ suiji.cpp -o program/suiji -O2 -std=c++17

cnt=0

while true; do
    let cnt++
    echo TEST:$cnt
```

```

    ./program/suiji > in
    ./program/a < in > out.a
    ./program/b < in > out.b

    diff out.a out.b
    if [ $? -ne 0 ];then break;fi
done
```

windows

```
@echo off

g++ a.cpp -o program/a -O2 -std=c++17
g++ b.cpp -o program/b -O2 -std=c++17
g++ suiji.cpp -o program/suiji -O2 -std=c++17

set cnt=0

:again
    set /a cnt=cnt+1
    echo TEST:%cnt%
    .\program\suiji > in
    .\program\a < in > out.a
    .\program\b < in > out.b

    fc output.a output.b
if not errorlevel 1 goto again
```

6.7 编译常用选项

```
-Wall -Woverflow -Wextra -Wpedantic -Wfloat-equal -Wshadow -fsanitize=address,undefined
```

6.8 开栈

不同的编译器可能命令不一样

```
-Wl,--stack=0x10000000
-Wl,-stack_size -Wl,0x10000000
-Wl,-z,stack-size=0x10000000
```

6.9 clang-format

```
BasedOnStyle: Google
IndentWidth: 4
ColumnLimit: 80
AllowShortIfStatementsOnASingleLine: AllIfsAndElse
AccessModifierOffset: -4
EmptyLineBeforeAccessModifier: Leave
RemoveBracesLLVM: true
```