算法模板总结

图论

1.网络流

有源汇上下界最大/小流

```cpp
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 c1, T c2 = T{}) {
        g[u].push_back(e.size());
        e.emplace_back(v, c1);
        g[v].push_back(e.size());
        e.emplace_back(u, c2);
    }
    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;
    }
};

void solve() {
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    vector<int> w(n + 1);
    MaxFlow<int> g(n + 2);
    while (m -- ) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        g.addEdge(a, b, d - c);
        w[b] += c;
        w[a] -= c;
    }
    int S = 0, T = n + 1, tot = 0;
    for (int i = 1; i <= n; i ++ ) {
        if (w[i] > 0) {
            g.addEdge(S, i, w[i]);
            tot += w[i];
        } else if (w[i] < 0) {
            g.addEdge(i, T, -w[i]);
        }
    }
    g.addEdge(t, s, inf);
    if (g.flow(S, T) < tot) {
        cout << "No Solution\n";
    } else {
        auto v = g.edges();
        int res = v.back().flow;
        g.e.back().cap = g.e[g.e.size() - 2].cap = 0;
        // 最大
        cout << g.flow(s, t) + res << '\n';
        // 最小
        cout << res - g.flow(t, s) << '\n';
    }
}
```

无源汇上下界可行流

```
constexpr int inf = 1E9;
template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };
    int n;
    std::vector<_Edge> e, l;
    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();
        l.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
    }

    T dfs(int u, int t, T f) {
    }
    void addEdge(int u, int v, T c, T d) {
        g[u].push_back(e.size());
        e.emplace_back(v, d - c);
        l.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
        l.emplace_back(-1, -1);
    }
    T flow(int s, int t) {
    }

    std::vector<bool> minCut() {
    }

    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
        T down;
    };
    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;
            x.down = l[i].cap;
            a.push_back(x);
        }
        return a;
    }
};

void solve() {
    int n, m;
    cin >> n >> m;
    MaxFlow<int> g(n + 2);
    int s = 0, t = n + 1;
    vector<int> dflow(n + 1);
    while (m -- ) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        g.addEdge(a, b, c, d);
        dflow[b] += c;
        dflow[a] -= c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i ++ ) {
        if (dflow[i] > 0) {
            g.addEdge(s, i, 0, dflow[i]);
            tot += dflow[i];
        } else if (dflow[i] < 0) {
            g.addEdge(i, t, dflow[i], 0);
        }
    }
    if (tot != g.flow(s, t)) {
        return cout << "NO\n", void();
    }
    cout << "YES\n";
    auto v  = g.edges();
    for (auto x : v) {
        if (x.from != 0 && x.to != n + 1) {
            cout << x.flow + x.down << '\n';
        }
    }
}
```

最大权闭合子图

选x点就必须选y点

新建源点S,向正权点连容量为点权的边, 新建汇点T,负权点向T连容量为点权的相反数的边。图中原有的边容量改为正无穷。正权点点权和减去最小割即为答案

```
void solve() {
    int n, m;
    cin >> n >> m;
    MaxFlow<int> g(n + m + 2);
    int S = 0, T = n + m + 1;
    for (int i = 1; i <= n; i ++ ) {
        int p; cin >> p;
        g.addEdge(m + i, T, p);
    }
    int tot = 0;
    for (int i = 1; i <= m; i ++ ) {
        int a, b, c;
        cin >> a >> b >> c;
        g.addEdge(S, i, c);
        g.addEdge(i, m + a, inf);
        g.addEdge(i, m + b, inf);
        tot += c;
    }
    cout << tot - g.flow(S, T) << '\n';
}
```

最大密度子图

选第i条边, 就要选该边的两个端点

```
// 最大密度子图(一般都是无向图)
// 最大化 |E| + |V| - g|V|
// 原图的边 + (i -> T, U + 2 * g - E_i - 2 * V_i) + (S -> i, U)
// Max(|E| + |V| - g|V|) = (U * n - c[S, T]) / 2;
// g一般需要二分, 不过这道题只需要最大化|E| + |V|, 令g=0即可
void solve() {
    cin >> n >> m;
    MaxFlow<int> f(n + 2);
    int S = 0, T = n + 1;
    vector<int> p(n + 1);
    for (int i = 1; i <= n; i ++ ) {
        cin >> p[i];
        p[i] *= -1;
    }

    vector<int> deg(n + 1);
    for (int i = 1; i <= m; i ++ ) {
        int u, v, c;
        cin >> u >> v >> c;
        deg[u] += c, deg[v] += c;
        // 网络流中的无向边可以这样简化
        f.addEdge(u, v, c, c);
    }

    // 偏移量, 保证流量为正
    int U = 0;
    for (int i = 1; i <= n; i ++ ) {
        U = max(U, deg[i] + 2 * p[i]);
    }

    for (int i = 1; i <= n; i ++ ) {
        f.addEdge(S, i, U);
        f.addEdge(i, T, U - 2 * p[i] - deg[i]);
    }

    cout << (U * n - f.flow(S, T)) / 2 << '\n';
}
```

有源汇上下界最小费用流

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> a(m), b(n), w(n + m + 2);
    MinCostFlow<int> g(n + m + 4);
    int s = n + m, t = n + m + 1, S = t + 1, T = t + 2;

    for (int i = 0; i < m; i++) {
        cin >> a[i];
        g.addEdge(i + n, t, 0, 0);
        w[t] += a[i];
        w[i + n] -= a[i];
    }

    for (int i = 0; i < n; i++) {
        cin >> b[i];
    }

    for (int i = 0; i < n; i++) {
        int l, r;
        cin >> l >> r;
        g.addEdge(s, i, r - l, 0);
        w[i] += l;
        w[s] -= l;
    }

    for (int i = 0; i < n; i++) {
        int k; cin >> k;
        while (k--) {
            int x; cin >> x;
            x--;
            g.addEdge(i, x + n, inf, b[i]);
        }
    }

    int tot = 0;
    for (int i = 0; i < n + m + 2; i++) {
        if (w[i] > 0) {
            g.addEdge(S, i, w[i], 0);
            tot += w[i];
        } else if (w[i] < 0) {
            g.addEdge(i, T, -w[i], 0);
        }
    }
    g.addEdge(t, s, inf, 0);

    auto [res, ans] = g.flow(S, T);
    if (res != tot) {
        ans = -1;
    }
    cout << ans << '\n';
}
```

2.线段树优化建图

```
struct SegmentTreeGraph;

struct SegmentTree {
    int n;
    SegmentTreeGraph &stg;
    SegmentTree(int n_, SegmentTreeGraph &stg_) : n(n_), stg(stg_) {}
    virtual void build(int p, int l, int r) = 0;
    virtual void modify(int p, int l, int r, int x, int y, int k, int w) = 0;
};

struct SegmentTreeGraph {
    int n, m, idx;
    std::vector<std::vector<std::pair<int, int>>> adj;
    // num第i个点在线段树中的编号, pos线段树中第p个节点所对应的编号
    struct SegmentTreeIn : public SegmentTree {
        std::vector<int> num, pos;
        SegmentTreeIn(int n_, SegmentTreeGraph &stg_) : SegmentTree(n_, stg_) {
            num.assign(n_, 0);
            pos.assign(n_ << 2, 0);
        }
        void build(int p, int l, int r) {
            pos[p] = stg.idx++;
            if (r - l == 1) {
                num[l] = pos[p];
                return;
            }
            int m = l + r >> 1;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            stg.add(pos[2 * p], pos[p], 0);
            stg.add(pos[2 * p + 1], pos[p], 0);
        }
        void build() {
            build(1, 0, n);
        }
        void modify(int p, int l, int r, int x, int y, int k, int w) {
            if (l >= y || r <= x) {
                return;
            }
            if (l >= x && r <= y) {
                stg.add(pos[p], k, w);
                return;
            }
            int m = l + r >> 1;
            if (x < m) {
                modify(2 * p, l, m, x, y, k, w);
            }
            if (y >= m) {
                modify(2 * p + 1, m, r, x, y, k, w);
            }
        }
        void modify(int x, int y, int k, int w) {
            modify(1, 0, n, x, y, k, w);
        }
    } in_sg;
    struct SegmentTreeOut : public SegmentTree {
        std::vector<int> num, pos;
        SegmentTreeOut(int n_, SegmentTreeGraph &stg_) : SegmentTree(n_, stg_) {
            num.assign(n_, 0);
            pos.assign(n_ << 2, 0);
        }
        void build(int p, int l, int r) {
            pos[p] = stg.idx++;
            if (r - l == 1) {
                num[l] = pos[p];
                return;
            }
            int m = l + r >> 1;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            stg.add(pos[p], pos[2 * p], 0);
            stg.add(pos[p], pos[2 * p + 1], 0);
        }
        void build() {
            build(1, 0, n);
        }
        void modify(int p, int l, int r, int x, int y, int k, int w) {
            if (l >= y || r <= x) {
                return;
            }
            if (l >= x && r <= y) {
                stg.add(k, pos[p], w);
                return;
            }
            int m = l + r >> 1;
            if (x < m) {
                modify(2 * p, l, m, x, y, k, w);
            }
            if (y >= m) {
                modify(2 * p + 1, m, r, x, y, k, w);
            }
        }
        void modify(int x, int y, int k, int w) {
            modify(1, 0, n, x, y, k, w);
        }
    } out_sg;
    SegmentTreeGraph(int n_, int m_) : n(n_), m(m_), idx(0), in_sg(n_, *this),
out_sg(n_, *this) {
        adj.assign((8 << std::__lg(n)) + m, {});
        in_sg.build();
        out_sg.build();
        for (int i = 0; i < n; i ++ ) {
            add(out_sg.num[i], in_sg.num[i], 0);
            add(in_sg.num[i], out_sg.num[i], 0);
        }
    }
    void add(int u, int v, int w) {
        adj[u].emplace_back(v, w);
    }
    void insert(int l1, int r1, int l2, int r2, int w = 1) {
        int s = idx++;
        in_sg.modify(l1, r1, s, 0);
        out_sg.modify(l2, r2, s, w);
    }
    std::vector<std::vector<std::pair<int, int>>> graph() {
        return adj;
    }
};
```

3. 欧拉路径/回路

对于无向图:

1.存在欧拉路径的充要条件: 度数为奇数的点只能有0个或2个

2.存在欧拉回路的充要条件: 度数为奇数的点只能有0个

对于有向图:

1.存在欧拉路径的充要条件: 要么所有点的出度等于入度, 要么只有两个点不满足出度等于入度, 且这两个点一个点出度-入度=1, 另一个点入度-出度=1

2.存在欧拉回路的充要条件: 所有点的出度等于入度

输出欧拉路径/回路的合法方案

```
vector<int> ans, st(n), cur(2 * n);
    auto dfs = [&](auto && self, int u) -> void {
        for (int &i = cur[u]; i < adj[u].size(); i ++ ) {
            auto [v, id] = adj[u][i];
            if (st[id]) {
                continue;
            }
            st[id] = 1;
            self(self, v);
            ans.push_back(id + 1);
        }
    };
    dfs(dfs, flg1);
```

动态规划

1.数位dp

```
int digit_dp(int l, int r) {
    string low = to_string(l), high = to_string(r);
    int n = high.size(), diff = n - low.size();
    unordered_map<i64, int> st; // 状态记录
    auto dfs = [&](auto && self, int i, int m, int s, bool lim_l, bool lim_h) ->
int {
        if (i == n) {
            return is_ok(m, s);
        }
        i64 msk = (i64)m << 32ll | i << 16ll | s; // dp数组，位运算，字符串叠加(用空
格分隔)(会很慢)，离散化
        if (!lim_l && !lim_h && st.count(msk)) {
            return st[msk];
        }
        int lo = lim_l && i >= diff ? low[i - diff] - '0' : 0;
        int hi = lim_h ? high[i] - '0' : 9;
        int res = 0, d = lo;
        if (lim_l && i < diff) {
            res = self(self, i + 1, 1, 0, 1, 0);
            d = 1;
        }
        for (; d <= hi; d++) {
            res += self(self, i + 1, m * d, s + d, lim_l && d == lo, lim_h && d
== hi);
        }
        if (!lim_l && !lim_h) {
            return st[msk] = res;
        }
        return res;
    };
    return dfs(dfs, 0, 1, 0, 1, 1);
```

2.dp套dp

```
void solve() {
    int n, m;
    cin >> n >> m;
    cin >> s;
    s = ' ' + s;
    int st = 1 << n;
    for (int i = 0, tmp[N], res[N]; i < st; i++) {
        res[0] = tmp[0] = 0;
        for (int j = 1, k = i; j <= n; j++, k >>= 1) {
            tmp[j] = tmp[j - 1] + (k & 1);
        }
        for (int k = 0, num; k < 26; k++) {
            num = 0;
            for (int j = 1; j <= n; j++) {
                res[j] = (s[j] == k + 'a') ? tmp[j - 1] + 1 : max(tmp[j], res[j -
1]);
                num += (1 << (j - 1)) * (res[j] - res[j - 1]);
            }
            nxt[i][k] = num;
        }
    }
    memset(f, 0, sizeof f);
    f[0][0] = 1;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < st; j++) {
            f[(i & 1) ^ 1][j] = 0;
        }
        for (int j = 0; j < st; j++) {
            for (int k = 0; k < 26; k++) {
                (f[(i & 1) ^ 1][nxt[j][k]] += f[i & 1][j]) %= p;
            }
        }
    }
    for (int i = 0; i <= n; i++) {
        ans[i] = 0;
    }
    for (int i = 0; i < st; i++) {
        (ans[__builtin_popcount(i)] += f[m & 1][i]) %= p;
    }
    for (int i = 0; i <= n; i++) {
        cout << ans[i] << " \n"[i == n];
    }
}
```

3.矩阵优化dp

对于初始矩阵A直接构造成n行1列, 转移矩阵M构造成n行n列即可

```
constexpr int N = 100;
struct Matrix {
    int g[N][N], m;
    Matrix(int m_ = N, int x = 1) {
        m = m_;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                g[i][j] = 0;
            }
            g[i][i] = x;
        }
    }
};

Matrix operator*(Matrix &a, Matrix &b) {
    Matrix c(b.m, 0);
    for (int i = 0; i < N; i++) {
        for (int k = 0; k < N; k++) {
            for (int j = 0; j < b.m; j++) {
                c.g[i][j] += a.g[i][k] * b.g[k][j] % P;
                c.g[i][j] %= P;
            }
        }
    }
    return c;
}
```

动态dp & 广义矩阵乘法

一般都是线性dp, 或者可以使用树链剖分拆成线性的树形dp

$$+\times\text{型}$$

$$\text{零矩阵}\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}$$

$$\text{单位矩阵}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

$$max+\text{型}$$

$$\text{零矩阵}\begin{bmatrix}-\infty&-\infty&-\infty\\-\infty&-\infty&-\infty\\-\infty&-\infty&-\infty\end{bmatrix}$$

$$\text{单位矩阵}\begin{bmatrix}0&-\infty&-\infty\\-\infty&0&-\infty\\-\infty&-\infty&0\end{bmatrix}$$

4.子集/超集DP

```
// 子集
for (int i = 0; i < N; i++) {
    for (int j = 0; j < 1 << N; j++) {
        if (j >> i & 1) {
            dp[j] += dp[j ^ (1 << i)];
        }
    }
}
// 超集
for (int i = 0; i < N; i++) {
    for (int j = 0; j < 1 << N; j++) {
        if (!(j >> i & 1)) {
            dp[j] += dp[j ^ (1 << i)];
        }
    }
}
```

5.bitset优化DP

形如

```
if (condition) {
    dp[i + x] |= dp[i];
}
```

的dp一般都能用bitset优化, 方式如下

```
bitset<N> mask, dp;
for (int i = 0; i < N; i++) {
    if (condition) {
        mask.set(i);
    }
}
dp |= (dp << x) & mask;
```

以上述的子集DP为例子, 不难发现等价于

$$dp_{j+(1<<i)} \mid = \ dp_j$$

优化方式如下

```
bitset<1 << N> mask[N], dp;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < 1 << N; j++) {
        if (j >> i & 1) {
            mask[i].set(j);
        }
    }
}
for (int i = 0; i < N; i++) {
    dp |= (dp << (1 << i)) & mask[i];
}
```

超集dp优化方式如下

```
bitset<1 << N> mask[N], dp;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < 1 << N; j++) {
        if (!(j >> i & 1)) {
            mask[i].set(j);
        }
    }
}
for (int i = 0; i < N; i++) {
    dp |= (dp >> (1 << i)) & mask[i];
}
```

数据结构

1.kruskal重构树

```
constexpr int N = 18, inf = -1E9;
template<class Cmp = less<int>>
struct KruskalRTree {
    struct Node {
        int u, v, w;
        bool operator<(const Node &t) {
            Cmp cmp;
            return cmp(w, t.w);
        }
    };

    vector<Node> e;
    vector<int> val;
    vector<vector<int>> dp;
    int n;
    HLD g;
    DSU d;

    void init(int n_) {
        n = n_;
        d.init(2 * n);
        g.init(2 * n - 1);
        val.assign(2 * n - 1, -1);
        dp.assign(N, vector<int>(2 * n - 1, inf));
    }

    KruskalRTree(int n_) {
        init(n_);
    }

    void addEdge(int u, int v, int w) {
        e.push_back({u, v, w});
    }

    i64 kruskal() {
        sort(e.begin(), e.end());
        int cnt = n;
        i64 res = 0;
        for (auto [u, v, w] : e) {
            int pu = d.find(u), pv = d.find(v);
            if (pu != pv) {
                g.addEdge(pu, cnt);
                g.addEdge(pv, cnt);
                d.f[u] = d.f[v] = cnt;
                val[cnt] = w;
                res += w;
                cnt++;
            }
        }
        return res;
    }

    i64 calc(i64 x, i64 y) {
        return max(x, y);
    }

    void bfs() {
        g.work(2 * n - 2);
        vector<int> st(2 * n - 1);
        queue<int> q;
        q.push(2 * n - 2);
        st[2 * n - 2] = 1;
        while (q.size()) {
            int u = q.front();
            q.pop();

            for (auto v : g.adj[u]) {
                if (!st[v]) {
                    st[v] = 1;
                    int p = g.jump(v, 1);
                    dp[0][v] = val[p];
                    for (int i = 1; i < N; i++) {
                        p = g.jump(v, 1 << i - 1);
                        if (p != -1) {
                            dp[i][v] = calc(dp[i - 1][v], dp[i - 1][p]);
                        }
                    }
                }
            }
        }
    }

    int jump(int u, i64 k) {
        for (int i = N - 1; i >= 0; i--) {
            int p = g.jump(u, 1 << i);
            if (p != -1 && dp[i][u] <= k) {
                u = p;
            }
        }
        return u;
    }

    int lca(int u, int v) {
        return g.lca(u, v);
    }
};
```

2.Splay

```
template<class Info, class Tag>
struct Splay {
    struct Node {
        array<int, 2> ch;
        int p;
        Info val;
        Tag v;
    };
    int root, idx;
    vector<Node> t;
    void pull(int x) {
        t[x].val = t[t[x].ch[0]].val + t[t[x].ch[1]].val;
    }
    void apply(int x, const Tag &v) {
        t[x].val.apply(v);
        t[x].v.apply(v);
    }
    void push(int x) {
        apply(t[x].ch[0], t[x].v);
        apply(t[x].ch[1], t[x].v);
        t[x].v = Tag();
    }
    void rotate(int x) {
        int y = t[x].p, z = t[y].p;
        int k = t[y].ch[1] == x;  // k=0表示x是y的左儿子；k=1表示x是y的右儿子
        t[z].ch[t[z].ch[1] == y] = x;
        t[x].p = z;
        t[y].ch[k] = t[x].ch[k ^ 1];
        t[t[x].ch[k ^ 1]].p = y;
        t[x].ch[k ^ 1] = y;
        t[y].p = x;
        pull(y);
        pull(x);
    }
    void splay(int x, int k) { //x转到k的子节点
        while (t[x].p != k) {
            int y = t[x].p, z = t[y].p;
            if (z != k) {
                if ((t[y].ch[1] == x) ^ (t[z].ch[1] == y)) {
                    rotate(x);
                } else {
                    rotate(y);
                }
            }
            rotate(x);
        }
        if (!k) {
            root = x;
        }
    }
    int find(int k) {
        int u = root;
        while (u) {
            push(u);
            if (t[t[u].ch[0]].val.sz >= k) {
                u = t[u].ch[0];
            } else if (t[t[u].ch[0]].val.sz + 1 == k) {
                return u;
            } else {
                k -= t[t[u].ch[0]].val.sz + 1;
                u = t[u].ch[1];
            }
        }
        return -1;
    }
};
struct Tag {
    int cnt;
    void apply(Tag t) {
        cnt += t.cnt;
    }
};
struct Info {
    int sum, sz, sz1;
    void apply(Tag t) {
        sz += t.cnt;
    }
};
Info operator+(const Info &a, const Info &b) {
    Info c;
    c.sz = a.sz + b.sz + 1;
    c.sz1 = a.sz1 + b.sz1 + 1;
    c.sum = a.sum + b.sum;
    return c;
}
```

LCT

```
struct LCT {
    struct Node {
        int ch[2] {};
        int p {};
        bool rev {};
    };

    vector<Node> t;

    LCT(int n) : t(n + 1) {}

    bool isRoot(int x) {
        int p = t[x].p;
        return p == 0 || (t[p].ch[0] != x && t[p].ch[1] != x);
    }

    void applyRev(int x) {
        if (!x) {
            return;
        }
        t[x].rev ^= 1;
        swap(t[x].ch[0], t[x].ch[1]);
    }

    void push(int x) {
        if (t[x].rev) {
            applyRev(t[x].ch[0]);
            applyRev(t[x].ch[1]);
            t[x].rev = false;
        }
    }

    void rotate(int x) {
        int p = t[x].p, g = t[p].p;
        int dx = (t[p].ch[1] == x);
        int w = t[x].ch[dx ^ 1];
        if (!isRoot(p)) {
            if (t[g].ch[0] == p) {
                t[g].ch[0] = x;
            } else {
                t[g].ch[1] = x;
            }
        }
        t[x].p = g;
        t[p].ch[dx] = w;
        if (w) {
            t[w].p = p;
        }
        t[x].ch[dx ^ 1] = p;
        t[p].p = x;
    }

    void pushAll(int x) {
        if (!isRoot(x)) {
            pushAll(t[x].p);
        }
        push(x);
    }

    void splay(int x) {
        pushAll(x);
        while (!isRoot(x)) {
            int p = t[x].p, g = t[p].p;
            if (!isRoot(p)) {
                bool xr = (t[g].ch[1] == p);
                bool pr = (t[p].ch[1] == x);
                if (xr == pr) {
                    rotate(p);
                } else {
                    rotate(x);
                }
            }
            rotate(x);
        }
    }

    void access(int x) {
        int last = 0;
        for (int y = x; y; y = t[y].p) {
            splay(y);
            t[y].ch[1] = last;
            last = y;
        }
        splay(x);
    }

    void change(int x) {
        access(x);
        applyRev(x);
    }

    int find(int x) {
        access(x);
        while (true) {
            push(x);
            if (!t[x].ch[0]) {
                break;
            }
            x = t[x].ch[0];
        }
        splay(x);
        return x;
    }

    bool connected(int u, int v) {
        if (u == v) {
            return true;
        }
        return find(u) == find(v);
    }

    void link(int u, int v) {
        change(u);
        if (find(v) != u) {
            t[u].p = v;
        }
    }

    void cut(int u, int v) {
        change(u);
        access(v);
        if (t[v].ch[0] == u && t[u].ch[1] == 0) {
            t[v].ch[0] = 0;
            t[u].p = 0;
        }
    }
};
```

3.Treap

```
struct Treap {
    struct Node {
        int l, r;
        int key, val;
        int cnt, size;
    };
    vector<Node> tr;
    int root, idx;

    Treap(int n) {
        tr.resize(n + 10 + q);
        root = 0, idx = 0;
    }

    void pushup(int p) {
        tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
    }

    int get_node(int key) {
        tr[ ++ idx ].key = key;
        tr[idx].val = rand();
        tr[idx].cnt = 1;
        tr[idx].size = 1;
        return idx;
    }

    void zig(int &p) {
        int q = tr[p].l;
        tr[p].l = tr[q].r;
        tr[q].r = p;
        p = q;
        pushup(p);
        pushup(tr[p].r);
    }

    void zag(int &p) {
        int q = tr[p].r;
        tr[p].r = tr[q].l;
        tr[q].l = p;
        p = q;
        pushup(p);
        pushup(tr[p].l);
    }

    void build() {
        get_node(-inf); get_node(inf);
        root = 1; tr[1].r = 2;
        pushup(root);
        if (tr[2].val > tr[1].val) {
            zag(root);
        }
    }

    void insert(int &p, int key) {
        if (!p) {
            p = get_node(key);
        } else if (tr[p].key == key) {
            tr[p].cnt ++;
        } else if (tr[p].key > key) {
            insert(tr[p].l, key);
            if (tr[p].val < tr[tr[p].l].val) {
                zig(p);
            }
        } else if (tr[p].key < key) {
            insert(tr[p].r, key);
            if (tr[p].val < tr[tr[p].r].val) {
                zag(p);
            }
        }
        pushup(p);
    }

    void pop(int &p, int key) {
        if (!p) {
            return;
        }
        if (tr[p].key == key) {
            if (tr[p].cnt > 1) {
                tr[p].cnt --;
```

```cpp
            } else {
                if (tr[p].l || tr[p].r) {
                    if (!tr[p].r || tr[p].val < tr[tr[p].l].val) {
                        zig(p);
                        pop(tr[p].r, key);
                    } else {
                        zag(p);
                        pop(tr[p].l, key);
                    }
                } else {
                    p = 0;
                }
            }
        } else if (tr[p].key > key) {
            pop(tr[p].l, key);
        } else pop(tr[p].r, key);
        pushup(p);
    }

    int get_rank(int p, int key) {
        if (!p) {
            return 0;
        }
        if (tr[p].key == key) {
            return tr[tr[p].l].size + 1;
        }
        if (tr[p].key > key) {
            return get_rank(tr[p].l, key);
        }
        if (tr[p].key < key) {
            return tr[tr[p].l].size + tr[p].cnt + get_rank(tr[p].r, key);
        }
    }

    int get_key(int p, int rank) {
        if (!p) {
            return inf;
        }
        if (tr[tr[p].l].size >= rank) {
            return get_key(tr[p].l, rank);
        } else if (tr[tr[p].l].size + tr[p].cnt >= rank) {
            return tr[p].key;
        } else {
            return get_key(tr[p].r, rank - tr[tr[p].l].size - tr[p].cnt);
        }
    }

    int get_pre(int p, int key) {
        if (!p) {
            return -inf;
        }
        if (tr[p].key >= key) {
            return get_pre(tr[p].l, key);
        } else {
            return max(tr[p].key, get_pre(tr[p].r, key));
        }
    }

    int get_next(int p, int key) {
        if (!p) {
            return inf;
        }
        if (tr[p].key <= key) {
            return get_next(tr[p].r, key);
        } else {
            return min(tr[p].key, get_next(tr[p].l, key));
        }
    }
};
```

4.Trie

```
template <class T, class Info, size_t V = 1>
struct Trie {
    int n, idx;
    vector<Info> info;
    void init(int n_) {
        idx = 0;
        info.assign(2 * V * n_, Info {});
    }
    Trie(int n_) : n(n_) {
        init(n_);
    }
    string unified(const T &v) {
        string nv;
        if constexpr (V != 1) {
            for (int i = V; i >= 0; --i) {
                nv += char((v >> i & 1) + 'a');
            }
        } else {
            nv = v;
        }
        return nv;
    }
    void insert(const T &v, const Info &x = Info{}) {
        string nv = unified(v);
        int p = 0;
        for (auto x : nv) {
            int u = x - 'a';
            if (!info[p].ch[u]) {
                info[p].ch[u] = ++ idx;
            }
            p = info[p].ch[u];
        }
        info[p].apply(x);
    }
    Info query(const T &v) {
        string nv = unified(v);
        int p = 0;
        for (auto x : nv) {
            int u = x - 'a';
            if (!info[p].ch[u]) {
                return Info{};
            }
            p = info[p].ch[u];
        }
        return info[p];
    }
    i64 find(const T &v) {
        if constexpr (V != 1) {
            int p = 0;
            i64 ans = 0;
            for (int i = V; i >= 0; --i) {
                int u = v >> i & 1;
                if (info[p].ch[u ^ 1]) {
                    ans |= (1LL << i);
                    p = info[p].ch[u ^ 1];
                } else {
                    p = info[p].ch[u];
                }
            }
            return ans;
        }
    }
};

template <size_t N>
struct Info {
    array<int, N> ch;
    void apply(const Info &x) {

    }
};
```

可持久化01Trie

```
template <class Info, size_t V = 30>
struct Persistent01Trie {
    struct Node {
        array<int, 2> ch;
        Info info =Info{};
    };
    int n, idx;
    vector<Node> tr;
    vector<int> root;
    void init(int n_) {
        idx = 0;
        tr.assign(V * n_ * 2, Node{});
        root.assign(n_ + 1, 0);
    }
    Persistent01Trie(int n_) : n(n_) {
        init(n_);
    }
    void pull(int q) {
        tr[q] = tr[tr[q].ch[0]].info + tr[tr[q].ch[1]].info;
    }
    void insert(int k, int p, int &q, int x, const Info &v) {
        q = ++ idx;
        tr[q] = tr[p];
        if (k < 0) {
            tr[q].info.apply(v);
            return;
        }
        int u = x >> k & 1;
        tr[q].ch[u ^ 1] = tr[p].ch[u ^ 1];
        insert(k - 1, tr[p].ch[u], tr[q].ch[u], x, v);
        pull(q);
    }
    void insert(int i, int x, const Info &v = Info{}) {
        insert(V, root[i], root[i + 1], x, v);
    }
    template <class F>
    i64 find(int k, int p, int q, int x, F pred) {
        if (k < 0) {
            return 0ll;
        }
        int u = x >> k & 1;
        if (pred(tr[tr[p].ch[u ^ 1]].info, tr[tr[q].ch[u ^ 1]].info)) {
            return find(k - 1, tr[p].ch[u ^ 1], tr[q].ch[u ^ 1], x, pred) + (1ll
<< k);
        } else {
            return find(k - 1, tr[p].ch[u], tr[q].ch[u], x, pred);
        }
    }
    template <class F>
    i64 find(int l, int r, int x, F pred) {
        return find(V, root[l + 1], root[r + 1], x, pred);
    }
};

struct Info {
    int val;
    void apply(const Info &v) {
    }
};

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

5.线段树

主席树

```
constexpr int inf = 1E9;
template <class Info>
struct PersistentSegmentTree {
    int n, idx;
    struct Node {
        int l = 0, r = 0;
        Info info = Info{};
    };
    vector<Node> tr;
    vector<int> root;
    void init(int n_) {
        idx = 0;
        n = n_;
        tr.assign(n_ << 5, Node{});
        root.assign(n_ + 1, 0);
    }
    PersistentSegmentTree(int n_) {
        init (n_);
    }
    void insert(int p, int &q, int l, int r, int x, const Info &v) {
        q = ++idx;
        tr[q] = tr[p];
        if (r - l == 1) {
            tr[q].info.apply(x, v);
            return;
        }
        int m = l + r >> 1;
        if (x < m) {
            insert(tr[p].l, tr[q].l, l, m, x, v);
        } else {
            insert(tr[p].r, tr[q].r, m, r, x, v);
        }
        pull(q);
    }
    void insert(int i, int x, const Info &v = Info{}) {
        insert(root[i], root[i + 1], -inf, inf + 1, x, v);
    }
    void pull(int q) {
        tr[q].info = tr[tr[q].l].info + tr[tr[q].r].info;
    }
    Info rangeQuery(int p, int q, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info{};
        }
        if (l >= x && r <= y) {
            return tr[q].info - tr[p].info;
        }
        int m = l + r >> 1;
        return rangequery(tr[p].l, tr[q].l, l, m, x, y) + rangequery(tr[p].r,
tr[q].r, m, r, x, y);
    }
    Info rangeQuery(int l, int r, int x, int y) {
        return rangeQuery(root[l + 1], root[r + 1], -inf, inf + 1, x, y);
    }
    template <class F>
    int findFirst(int p, int q, int l, int r, int x, F pred) {
        if (r - l == 1) {
            return l;
        }
        int m = l + r >> 1;
        if (pred(tr[tr[p].l].info, tr[tr[q].l].info, x)) {
            return findFirst(tr[p].l, tr[q].l, l, m, x, pred);
        } else {
            return findFirst(tr[p].r, tr[q].r, m, r, x, pred);
        }
    }
    template <class F>
    int findFirst(int l, int r, int x, F pred) {
        return findFirst(root[l + 1], root[r + 1], -inf, inf + 1, x, pred);
    }
    template <class F>
    int findLast(int p, int q, int l, int r, int x, F pred) {
        if (r - l == 1) {
            return l;
        }
        int m = l + r >> 1;
        if (pred(tr[tr[p].r].info, tr[tr[q].r].info, x)) {
            return findFirst(tr[p].r, tr[q].r, m, r, x, pred);
        } else {
            return findFirst(tr[p].l, tr[q].l, l, m, x, pred);
        }
    }
    template <class F>
    int findLast(int l, int r, int x, F pred) {
        return findLast(root[l + 1], root[r + 1], -inf, inf + 1, x, pred);
    }
};
struct Info {
    int val = 0;
    void apply (int x, const Info &v) {
        val += v.val;
    }
};
Info operator+ (const Info &a, const Info &b) {
    return {a.val + b.val};
}
Info operator- (const Info &a, const Info &b) {

}
```

势能线段树(区间&x + 区间|x)

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

```cpp
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeAnd(int p, int l, int r, int x, int y, const Info &v) {
        i64 vor = info[p].vor, vand = info[p].vand;
        if (r <= x || l >= y || (vor & v.vor) == vor) {
            return;
        }
        i64 mask = vor ^ vand;
        if (l >= x && r <= y && (mask | v.vor) == v.vor) {
            i64 val = (vand & v.vor) - vand;
            apply(p, {val});
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeAnd(2 * p, l, m, x, y, v);
        rangeAnd(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeAnd(int l, int r, const Info &v) {
        rangeAnd(1, 0, n, l, r, v);
    }
    void rangeOr(int p, int l, int r, int x, int y, const Info &v) {
        i64 vor = info[p].vor, vand = info[p].vand;
        if (r <= x || l >= y || (vand | v.vor) == vand) {
            return;
        }
        i64 mask = vor ^ vand;
        if (l >= x && r <= y && !(mask & v.vor)) {
            i64 val = (vand | v.vor) - vand;
            apply(p, {val});
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeOr(2 * p, l, m, x, y, v);
        rangeOr(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeOr(int l, int r, const Info &v) {
        rangeOr(1, 0, n, l, r, v);
    }
};

struct Tag {
    i64 x = 0;
    void apply(Tag t) {
        x += t.x;
    }
};

struct Info {
    i64 vor = 0, vand = 0;
    void apply(Tag t) {
        vor += t.x;
        vand += t.x;
    }
};
Info operator+(Info a, Info b) {
    Info c;
    c.vor  = a.vor | b.vor;
    c.vand = a.vand & b.vand;
    return c;
}
```

扫描线

```
vector<double> ys;
int find(double y) {
    return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
}

template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(8 * n + 10, Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            info[p] = {l, r, 0, 0};
            if (l != r) {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m + 1, r);
            }
        };
        build(1, 0, n);
    }
    void pull(int p) {
        Info &c = info[p], &a = info[2 * p], &b = info[2 * p + 1];
        if (c.cnt) c.len = ys[c.r + 1] - ys[c.l];
        else if (c.l != c.r) c.len = a.len + b.len;
        else c.len = 0;
    }
    void modify(int p, int l, int r, int x) {
        if (info[p].l >= l && info[p].r <= r) {
            info[p].cnt += x;
```

```cpp
        } else {
            int mid = info[p].l + info[p].r >> 1;
            if (l <= mid) modify(2 * p, l, r, x);
            if (r > mid) modify(2 * p + 1, l, r, x);
        }
        pull(p);
    }
};

struct Info {
    int l, r, cnt;
    double len;
};

struct Seg {
    double x, y1, y2;
    int d;
    bool operator< (const Seg &tmp) const {
        return x < tmp.x;
    }
};

int main() {
    int t = 1, n;
    while (cin >> n, n) {
        ys.clear();
        vector<Seg> segs;
        for (int i = 0; i < n; i ++ ) {
            double x1, y1, x2, y2;
            cin >> x1 >> y1 >> x2 >> y2;
            segs.push_back({x1, y1, y2, 1});
            segs.push_back({x2, y1, y2, -1});
            ys.push_back(y1), ys.push_back(y2);
        }
        sort(segs.begin(), segs.end());
        sort(ys.begin(), ys.end());
        ys.erase(unique(ys.begin(), ys.end()), ys.end());
        int m = segs.size();
        SegmentTree<Info> sg((int)ys.size() - 2);
        double res = 0;
        for (int i = 0; i < m; i ++ ) {
            if (i) {
                res += sg.info[1].len * (segs[i].x - segs[i - 1].x);
            }
            sg.modify(1, find(segs[i].y1), find(segs[i].y2) - 1, segs[i].d);
        }
    }
    return 0;
}
```

线段树分治

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

```
template<class T>
struct Info {
    vector<T> ch;
    void apply(const T &v) {
        ch.push_back(v);
    }
};
```

线段树套平衡树

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
        };
        build(1, 0, n);
    }
    void modify(int p, int l, int r, int x, int v1, int v2) {
        auto &S = info[p].s;
        if (S.find(v1) != S.end()) {
            S.erase(v1);
        }
        S.insert(v2);
        if (r - l == 1) {
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v1, v2);
        } else {
            modify(2 * p + 1, m, r, x, v1, v2);
        }
    }
    void modify(int p, int v1, int v2) {
        modify(1, 0, n, p, v1, v2);
    }
    pair<int, int> rangeQuery(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x) {
            return {0, 0};
        }
        if (l >= x && r <= y) {
            auto &S = info[p].s;
            int res1 = 0, res2 = 0;
            auto it = S.lower_bound(v);
            res1 = distance(S.begin(), it);
            it = S.upper_bound(v);
            if (it != S.end()) {
                res2 = distance(it, S.end());
            }
            return {res1, res2};
        }
        int m = (l + r) / 2;
        auto [x1, y1] = rangeQuery(2 * p, l, m, x, y, v);
        auto [x2, y2] = rangeQuery(2 * p + 1, m, r, x, y, v);
        return {x1 + x2, y1 + y2};
    }
    pair<int, int> rangeQuery(int l, int r, int v) {
        return rangeQuery(1, 0, n, l, r, v);
    }
};
struct Info {
    set<int> s;
};
```

6. 并查集

可撤销并查集

```
struct DSU {
    std::vector<std::pair<int &, int>> his;

    int n;
    std::vector<int> f, g;

    DSU(int n_) : n(n_), f(n, -1), g(n) {}

    std::pair<int, int> find(int x) {
        if (f[x] < 0) {
            return {x, 0};
        }
        auto [u, v] = find(f[x]);
        return {u, v ^ g[x]};
    }

    void set(int &a, int b) {
        his.emplace_back(a, a);
        a = b;
    }

    void merge(int a, int b, int &ans) {
        auto [u, xa] = find(a);
        auto [v, xb] = find(b);
        int w = xa ^ xb ^ 1;
        if (u == v) {
            return;
        }
        if (f[u] > f[v]) {
            std::swap(u, v);
        }
        set(f[u], f[u] + f[v]);
        set(f[v], u);
        set(g[v], w);
        //如果需要更新ans，就直接set(ans, 更新后的ans)
    }

    int timeStamp() {
        return his.size();
    }

    void rollback(int t, int &ans) {
        while (his.size() > t) {
            auto [x, y] = his.back();
            x = y;
            his.pop_back();
        }
    }
};
```

带权并查集

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

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

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

    int find(int x) {
        if (f[x] != x) {
            int t = find(f[x]);
            d[x] ^= d[f[x]];
            f[x] = t;
        }
        return f[x];
    }

    bool same(int x, int y) {
        int px = find(x), py = find(y);
        if (px == py) {
            return !(d[x] ^ d[y]);
        }
    }

    //t == 1不同类, t == 0 同一类
    bool merge(int x, int y, int t) {
        int px = find(x);
        int py = find(y);
        if (px == py) {
            return (d[x] ^ d[y]) == t;
        }
        siz[px] += siz[py];
        f[py] = px;
        d[py] = d[x] ^ d[y] ^ t;
        return true;
    }

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

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

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

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

    int find(int x) {
        if (f[x] != x) {
            int t = find(f[x]);
            d[x] += d[f[x]];
            f[x] = t;
        }
        return f[x];
    }

    void merge(int x, int y, int t) {
        int px = find(x);
        int py = find(y);
        if (px == py) {
            return;
        }
        f[py] = px;
        d[py] = d[x] - d[y] - t;
    }

    int dist(int u, int v) {
        return d[u] - d[v];
    }
};
```

7. 数组模拟双向链表

```
constexpr int N = 5005;
template<class T>
struct List {
    int e[N], l[N], r[N];
    int idx;

    void init() {
        idx = 2;
        r[0] = 1;
        l[1] = 0;
    }

    List() {
        init();
    }

    void insert(int k, const T& x) {
        e[idx] = x;
        l[idx] = k;
        r[idx] = r[k];
        l[r[k]] = idx;
        r[k] = idx++;
    }

    void remove(int k) {
        l[r[k]] = l[k];
        r[l[k]] = r[k];
    }
};
```

8. 珂朵莉树

```
struct ODT {
    map<int, int> odt;

    ODT() {
        odt[-1] = 0;
    }

    void split(int x) {
        auto it = prev(odt.upper_bound(x));
        odt[x] = it->second;
    }
    void assign(int l, int r, int v) {
        split(l);
        split(r);
        auto it = odt.find(l);
        while (it->first != r) {
            it = odt.erase(it);
        }
        odt[l] = v;
    }
};
```

9. 笛卡尔树

```
// 第一关键字满足中序遍历按cmp1序, 第二关键字满足小根堆(less)/大根堆(greater)
struct Node {
    int l = -1, r = -1;
};
template<class T1, class T2, class Cmp1 = less<T1>, class Cmp2 = less<T2>>
struct CartesianTree {
    vector<T1> a;
    vector<T2> b;
    vector<int> p;
    vector<Node> t;
    int n;

    void init(const vector<pair<T1, T2>> &q) {
        n = q.size();
        for (auto [x, y] : q) {
            a.push_back(x);
            b.push_back(y);
        }
        p.resize(n);
        iota(p.begin(), p.end(), 0);
        Cmp1 cmp1;
        sort(p.begin(), p.end(), [&](int i, int j) {
            return cmp1(a[i], a[j]);
        });
        t.assign(n, Node{});
        vector<int> stk;
        Cmp2 cmp2;
        for (int i = 0; i < n; i++) {
            int x = p[i], lst = -1;
            while (stk.size() && cmp2(b[x], b[stk.back()])) {
                lst = stk.back();
                stk.pop_back();
            }
            if (stk.size()) {
                t[stk.back()].r = x;
            }
            t[x].l = lst;
            stk.push_back(x);
        }
    }
    CartesianTree(const vector<pair<T1, T2>> &q) {
        init(q);
    }

    vector<Node> work() {
        return t;
    }
};
```

字符串

1. 字符串哈希

```
template<u32 P = 131>
struct StringHash {
    vector<u32> p, h;
    int n;
    StringHash(int n_ = 1) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        p.resize(n + 1);
        h.resize(n + 1);
        p[0] = 1;
        for (int i = 1; i <= n; i++) {
            p[i] = p[i - 1] * P;
        }
    }

    void setString(string s) {
        if (s.size() > n) {
            n = s.size();
            init(n);
        }
        for (int i = 1; i <= n; i++) {
            h[i] = h[i - 1] * P + s[i - 1];
        }
    }

    u32 operator()(int l, int r) {
        return h[r + 1] - h[l] * p[r - l + 1];
    }
};
```

数论

1. CRT(中国剩余定理)

$$\text{求解形如}\begin{cases} x & \equiv & a_1(mod\ n_1) \\ x & \equiv & a_2(mod\ n_2) \\ \dots \\ x & \equiv & a_k(mod\ n_k) \end{cases}$$

$$\text{其中}n_1, \dots, n_k\text{两两互质}$$

```
template<u32 P>
struct CRT {
    vector<i64> fac;

    CRT() {
        i64 x = P;
        for (int i = 2; 1U * i * i <= x; i++) {
            if (x % i == 0) {
                i64 res = 1;
                while (x % i == 0) {
                    x /= i;
                    res *= i;
                }
                fac.push_back(res);
            }
        }
        if (x > 1) {
            fac.push_back(x);
        }
    }

    i64 exgcd(i64 a, i64 b) {
        i64 u = 1, v = 0, p = b;
        while (b) {
            i64 t = a / b;
            a -= t * b; swap(a, b);
            u -= t * v; swap(u, v);
        }
        return (u + p) % p;
    }

    template<typename T>
    i64 crt(vector<T> &a) {
        for (int i = 0; i < fac.size(); i++) {
            for (int j = i + 1; j < fac.size(); j++) {
                if (gcd(fac[i], fac[j]) > 1) {
                    return 0LL;
                }
            }
        }
        i64 ans = 0;
        for (int i = 0; i < fac.size(); i ++ ) {
            i64 t1 = P / fac[i], t2 = exgcd(t1, fac[i]);
            ans = (ans + 1LL * t1 * t2 % P * a[i] % P) % P;
        }
        return ans % P;
    }
};
```

2. 卷积

FFT(快速傅里叶变换)

$$求解C_k = \sum_{i+j=k} a_i \times b_j$$

```
template<class T>
struct FFT {
    vector<complex<T>> a, b;
    int n, m, k;
    vector<int> rev;
    T pi = acos(-1);

    void init(const vector<int> &p, const vector<int> &q) {
        n = p.size() + q.size() - 1;
        k = 1;
        while ((1 << k) < n) {
            k++;
        }
        m = 1 << k;
        a.assign(m, 0);
        b.assign(m, 0);
        rev.assign(m, 0);
        for (int i = 0; i < m; i++) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << k - 1);
        }
        for (int i = 0; i < p.size(); i++) {
            a[i] = {p[i], T{}};
        }
        for (int i = 0; i < q.size(); i++) {
            b[i] = {q[i], T{}};
        }
    }

    FFT(const vector<int> &p, const vector<int> &q) {
        init(p, q);
    }

    void fft(vector<complex<T>> &a, int inv = 1) {
        for (int i = 0; i < m; i++) {
            if (i < rev[i]) {
                swap(a[i], a[rev[i]]);
            }
        }

        for (int len = 1; len < m; len <<= 1) {
            complex<T> w1{cos(pi / len), inv * sin(pi / len)};
            for (int i = 0; i < m; i += (len << 1)) {
                complex<T> wk{1, 0};
                for (int j = 0; j < len; j++, wk = wk * w1) {
                    auto l = a[i + j], r = wk * a[i + j + len];
                    a[i + j] = l + r;
                    a[i + j + len] = l - r;
                }
            }
        }
    }

    void work() {
        fft(a);
        fft(b);
        for (int i = 0; i < m; i++) {
            a[i] = a[i] * b[i];
        }
        fft(a, -1);
        for (int i = 0; i < n; i++) {
            a[i] = a[i].real() / m + 0.5;
        }
    }
};
```

NTT(模意义下的快速傅里叶变换/快速数论变换)

```
template<u32 P, u32 g>
struct NTT {
    using Z = ModInt<P>;
    vector<Z> a, b;
    int n, m, k;
    vector<int> rev;
    Z invg, invm;

    void init(const vector<int> &p, const vector<int> &q) {
        n = p.size() + q.size() - 1;
        k = 1;
        while ((1 << k) < n) {
            k++;
        }
        m = 1 << k;
        invg = Z(1) / g;
        invm = Z(1) / m;
        a.assign(m, 0);
        b.assign(m, 0);
        rev.assign(m, 0);
        for (int i = 0; i < m; i++) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << k - 1);
        }
        for (int i = 0; i < p.size(); i++) {
            a[i] = Z(p[i]);
        }
        for (int i = 0; i < q.size(); i++) {
            b[i] = Z(q[i]);
        }
    }

    NTT() {}

    NTT(const vector<int> &p, const vector<int> &q) {
        init(p, q);
    }


    void ntt(vector<Z> &a, int inv = 1) {
        for (int i = 0; i < m; i++) {
            if (i < rev[i]) {
                swap(a[i], a[rev[i]]);
            }
        }

        for (int len = 1; len < m; len <<= 1) {
            Z w1 = power(inv == 1 ? g : invg, (P - 1) / (len << 1));
            for (int i = 0; i < m; i += (len << 1)) {
                Z wk = 1;
                for (int j = 0; j < len; j++, wk = wk * w1) {
                    Z l = a[i + j], r = wk * a[i + j + len];
                    a[i + j] = l + r;
                    a[i + j + len] = l - r;
                }
            }
        }
    }

    void work() {
        ntt(a);
        ntt(b);
        for (int i = 0; i < m; i++) {
            a[i] = a[i] * b[i];
        }
        ntt(a, -1);
        for (int i = 0; i < n; i++) {
            a[i] = a[i] * invm;
        }
    }
};
```

MTT(任意模数下的快速傅里叶变换)

```
constexpr u32 P1 = 469762049;
constexpr u32 P2 = 998244353;
constexpr u32 P3 = 1004535809;
struct MTT {
    NTT<P1, 3> g1;
    NTT<P2, 3> g2;
    NTT<P3, 3> g3;
    vector<vector<i64>> a;
    int n;
    u32 P;
    void init(const vector<int> &p, const vector<int> &q, u32 P_) {
        n = p.size() + q.size() - 1;
        P = P_;
        g1.init(p, q);
        g2.init(p, q);
        g3.init(p, q);
        g1.work();
        g2.work();
        g3.work();
        a.assign(3, vector<i64>(n));
        for (int j = 0; j < n; j++) {
            a[0][j] = g1.a[j].val();
            a[1][j] = g2.a[j].val();
            a[2][j] = g3.a[j].val();
        }
    }

    MTT(const vector<int> &p, const vector<int> &q, u32 P_) {
        init(p, q, P_);
    }

    i64 power(i64 a, i64 b, u32 P) {
        i64 res = 1;
        for (; b; b >>= 1, a = a * a % P) {
            if (b & 1) {
                res = res * a % P;
            }
        }
        return res;
    }

    i64 inv(i64 a, u32 P) {
        return power(a, P - 2, P);
    }

    void work() {
        for (int i = 0; i < n; i++) {
            i64 t = a[0][i] + (a[1][i] - a[0][i] + P2) % P2 * inv(P1, P2) % P2 *
P1;
            i64 ans = (t + (a[2][i] - t % P3 + P3) % P3 * inv(1ll * P1 * P2 % P3,
P3) % P3 * P1 % P * P2 % P) % P;
            a[0][i] = ans;
        }
    }
};
```

FWT(快速莫比乌斯变换)

$$求解C_k = \sum_{i位运算j=k} a_i \times b_j$$

```
class Or {
```

```cpp
public:
    void operator()(auto &a, auto inv) {
        int m = a.size();
        for (int len = 1; len < m; len <<= 1) {
            for (int i = 0; i < m; i += (len << 1)) {
                for (int j = 0; j < len; j++) {
                    a[i + j + len] += a[i + j] * inv;
                }
            }
        }
    }
};

class And {
public:
    void operator()(auto &a, auto inv) {
        int m = a.size();
        for (int len = 1; len < m; len <<= 1) {
            for (int i = 0; i < m; i += (len << 1)) {
                for (int j = 0; j < len; j++) {
                    a[i + j] += a[i + j + len] * inv;
                }
            }
        }
    }
};

class Xor {
public:
    void operator()(auto &a, auto inv) {
        int m = a.size();
        for (int len = 1; len < m; len <<= 1) {
            for (int i = 0; i < m; i += (len << 1)) {
                for (int j = 0; j < len; j++) {
                    a[i + j] += a[i + j + len];
                    a[i + j + len] = a[i + j] - 2 * a[i + j + len];
                    a[i + j] *= inv;
                    a[i + j + len] *= inv;
                }
            }
        }
    }
};

template<u32 P, class FWT_>
struct FWT {
    using Z = ModInt<P>;
    vector<Z> a, b;
    int n, m, k;
    Z inv2;

    void init(const vector<int> &p, const vector<int> &q) {
        n = p.size() + q.size() - 1;
        k = 1;
        inv2 = Z(1) / 2;
        while ((1 << k) < n) {
            k++;
        }
        m = 1 << k;
        a.assign(m, 0);
        b.assign(m, 0);
        for (int i = 0; i < p.size(); i++) {
            a[i] = Z(p[i]);
        }
        for (int i = 0; i < q.size(); i++) {
            b[i] = Z(q[i]);
        }
    }

    FWT() {}

    FWT(const vector<int> &p, const vector<int> &q) {
        init(p, q);
    }

    void work() {
        FWT_ fwt;
        fwt(a, Z(1));
        fwt(b, Z(1));
        for (int i = 0; i < m; i++) {
            a[i] = a[i] * b[i];
        }
        if (std::is_same<FWT_, Xor>::value) {
            fwt(a, inv2);
        } else {
            fwt(a, Z(-1));
        }
    }
};
```

循环卷积

$$\text{求解形如以下形式的问题：}$$

$$\text{对于}k \in [0, m), \text{快速求解}C_k = \sum_{i=0}^{n-1} a_i \times b_{(i+k) \bmod m}$$

$$\text{反转}b\text{数组并且与}a\text{数组卷积，对于卷积后的数组}A\text{，}C_k = A_k + A_{k+m}$$

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> b(m), cnt(m);
    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        b[x]++;
    }
    vector<int> val{1, 0, 0, 0, 1, 0, 1, 0, 2, 1};
    for (int i = 0; i < m; i++) {
        int x = m - i - 1;
        if (!x) {
            cnt[i]++;
        }
        while (x) {
            cnt[i] += val[x % 10];
            x /= 10;
        }
    }

    NTT<998244353, 3> g(b, cnt);
    g.work();
    int mx = 0;
    for (int i = 0; i < m; i++) {
        mx = max<int>(mx, g.a[i].val() + g.a[i + m].val());
    }
    cout << mx << '\n';
}
```

差卷积

常用于快速求解二项式反演, 所以这里以二项式反演为基础讲解

$$\text{令}f_m\text{为恰好选}m\text{个物品的方案数，}g_m\text{为至少选}m\text{个物品的方案数}$$

$$\begin{aligned}
f_m &= \sum_{i=m}^{n}(-1)^{i-m}\binom{i}{m}g_i \\
&= \sum_{i=m}^{n}(-1)^{i-m}\frac{i!}{m!(i-m)!}g_i \\
f_m\cdot m! &= \sum_{i=m}^{n}(i!\cdot g_i)\cdot\frac{(-1)^{i-m}}{(i-m)!} \\
\text{令}a_i &= i!\cdot g_i\text{，}b_i=\frac{(-1)^{n-i}}{(n-i)!} \\
f_m\cdot m! &= \sum_{i=m}^{n}a_i\cdot b_{n+m-i} \\
&= C_{n+m}(\text{卷积后的第}n+m\text{项})
\end{aligned}$$

```
void solve() {
    int n, q;
    cin >> n >> q;
    vector g(n + 1, vector<Z>());
    for (int i = 0; i < n; i++) {
        g[0].push_back(comb.C(n, i) * comb.C(2 * n - 2 - i, n - 1 - i));
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 0; k * i <= n - 1; i++) {
            g[k].push_back(comb.C(n, i) * comb.C(n - 1 - k * i + n - 1 - i, n - 1
- i));
        }
    }

    vector f(n + 1, vector<Z>());
    for (int k = 0; k <= n; k++) {
        vector<Z> a, b;
        int m = g[k].size();
        // m-1对应上述推导式子中的n(不是输入的n)
        for (int i = 0; i < g[k].size(); i++) {
            a.push_back(comb.fac(i) * g[k][i]);
            b.push_back(power(Z(-1), m - 1 - i) / comb.fac(m - 1 - i));
        }
        NTT<P, 3> c(a, b);
        c.work();
        for (int i = m - 1; i < c.n; i++) {
            f[k].push_back(c.a[i] / n);
        }
    }

    if (n == 1) {
        f[0].push_back(1);
    }

    while (q--) {
        int m, k;
        cin >> m >> k;
        if (m >= f[k].size()) {
            cout << "0\n";
        } else {
            cout << f[k][m] / comb.fac(m) << '\n';
        }
    }
}
```

3. 拉格朗日插值

仅适用于多项式, 求解当x=k时多项式的近似值

```
auto Lagrange = [&](int k) {
    Z ans = 0;
    for (int i = 0; i < n; i++) {
        Z res = y[i];
        for (int j = 0; j < n; j++) {
            if (i == j) {
                continue;
            }
            res *= (k - x[j]) / (x[i] - x[j]);
        }
        ans += res;
    }
    return ans;
};
```

4. 类欧几里得

$$f(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$$

1. $a \ge c \ \vee \ b \ge c$

$$\begin{aligned}
f(a,b,c,n)&=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor\\
&=\sum_{i=0}^{n}\lfloor\frac{(c\lfloor\frac{a}{c}\rfloor+a\bmod c)i\ +\ (c\lfloor\frac{b}{c}\rfloor+b\bmod c)}{c}\rfloor\\
&=\sum_{i=0}^{n}\lfloor\frac{a}{c}\rfloor i+\sum_{i=0}^{n}\lfloor\frac{b}{c}\rfloor\ +\ \sum_{i=0}^{n}\lfloor\frac{(a\bmod c)i\ +\ (b\bmod c)}{c}\rfloor\\
&=\lfloor\frac{a}{c}\rfloor\frac{n(n+1)}{2}\ +\ \lfloor\frac{b}{c}\rfloor(n+1)\ +\ f(a\bmod c,b\bmod c,c,n)
\end{aligned}$$

2. $a < c \ \wedge \ b < c$

$$\begin{aligned}
m&=\lfloor\frac{an+b}{c}\rfloor\\
f(a,b,c,n)&=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor=\sum_{i=0}^{n}\sum_{j=0}^{\lfloor\frac{ai+b}{c}\rfloor-1}1\\
&=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[j<\lfloor\frac{ai+b}{c}\rfloor]=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[j<\frac{ai+b}{c}]=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[j+1\le\frac{ai+b}{c}]\\
&=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[cj+c-b-1<ai]=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[\frac{cj+c-b-1}{a}<i]\\
&=\sum_{j=0}^{m-1}\sum_{i=0}^{n}[\lfloor\frac{cj+c-b-1}{a}\rfloor<i]=\sum_{j=0}^{m-1}(n-\lfloor\frac{cj+c-b-1}{a}\rfloor)\\
&=nm\ -\ f(c,c-b-1,a,m-1)
\end{aligned}$$

$$g(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor i$$

$$g(a,b,c,n) = \begin{cases} \lfloor \frac{a}{c} \rfloor \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \frac{n(n+1)}{2} + g(a \bmod c, b \bmod c, c, n) \ (a \geq c \ \lor \ b \geq c) \\ \\ \frac{1}{2}(mn(n+1) - f(c, c-b-1, a, m-1) - \\ \\ h(c, c-b-1, a, m-1)) \ (a < c \ \land \ b < c) \end{cases}$$

$$h(a,b,c,n) = \sum_{i=0}^{n} (\lfloor \frac{ai+b}{c} \rfloor)^2$$

$$h(a,b,c,n) = \begin{cases} h(a \bmod c, b \bmod c, c, n) + 2\lfloor \frac{b}{c} \rfloor f(a \bmod c, b \bmod c, c, n) + \\ \\ 2\lfloor \frac{a}{c} \rfloor g(a \bmod c, b \bmod c, c, n) + \lfloor \frac{a}{c} \rfloor^2 \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor^2 (n+1) + \\ \\ \lfloor \frac{a}{c} \rfloor \lfloor \frac{b}{c} \rfloor n(n+1) \ (a \geq c \ \lor \ b \geq c) \\ \\ \\ nm(m+1) - 2g(c, c-b-1, a, m-1) - \\ \\ 2f(c, c-b-1, a, m-1) - f(a,b,c,n) \ (a < c \ \land \ b < c) \end{cases}$$

```cpp
template<class T>
struct EuclideanLike {
    struct Data {
        T f = T(), g = T(), h = T();
    };
    // Z i2 = Z(1) / 2, i6 = Z(1) / 6;
    Data work(i64 a, i64 b, i64 c, i64 n) {
        Data ans;
        i64 m = (a * n + b) / c, ac = a / c, bc = b / c;

        if (!a) {
            ans.f = T(n + 1) * bc;
            ans.g = T(n) * (n + 1) / 2 * bc;
            ans.h = T(n + 1) * bc * bc;
            return ans;
        }

        if (a >= c || b >= c) {
            ans.f = T(n) * (n + 1) / 2 * ac + T(n + 1) * bc;
            ans.g = T(n) * (n + 1) * (2 * n + 1) / 6 * ac + T(n) * (n + 1) / 2 *
bc;
            ans.h = T(n) * (n + 1) * (2 * n + 1) / 6 * ac * ac + T(n + 1) * bc *
bc + T(n) * (n + 1) * ac * bc;

            Data d = work(a % c, b % c, c, n);

            ans.f += d.f;
            ans.g += d.g;
            ans.h += d.h + 2 * bc * d.f + 2 * ac * d.g;
            return ans;
        }
        Data d = work(c, c - b - 1, a, m - 1);
        ans.f = T(n) * m - d.f;
        ans.g = T(n) * m * (n + 1) / 2 - (d.h + d.f) / 2;
        ans.h = T(n) * m * (m + 1) - 2 * d.g - 2 * d.f - ans.f;
        return ans;
    }
};
```

5. 线性基

```
constexpr i64 N = 63, inf = 1ll << N;
struct Basis {
    bitset<N> a[N];
    vector<int> t;

    Basis() {
        t.assign(N, -1);
    }

    bool insert(bitset<N> x, int tm = 1E9) {
        for (int k = N - 1; k >= 0; k--) {
            if (x[k] == 1) {
                if (tm > t[k]) {
                    swap(x, a[k]);
                    swap(tm, t[k]);
                }
                x ^= a[k];
            }
        }
        return x.to_ullong() == 0;
    }

    i64 query(bitset<N> x, int tm = 0) {
        for (int k = N - 1; k >= 0; k--) {
            if (t[k] >= tm) {
                // 求最小值
                // if (x[k] == 1) {
                //     x ^= a[k];
                // }
                // 求最大值
                if (x[k] == 0) {
                    x ^= a[k];
                }
            }
        }
        return x.to_ullong();
    }

    i64 queryK(bitset<N> x, int n, i64 id = inf, int tm = 0) {
        int cnt = 0;
        for (int k = N - 1; k >= 0; k--) {
            if (t[k] >= tm) {
                cnt += (a[k] != 0);
            }
        }

        // 如果求第k大
        // id = (1ll << cnt) - id + 1;
        // if (id <= 0) {
        //     return -1;
        // }

        // 如果必须要选
        //if (cnt < n) {
        //    id--;
        //}
        // 否则
        // id--;
        if (id >= (1ll << cnt)) {
            return -1;
        }

        for (int k = N - 1, i = cnt; k >= 0; k--) {
            if (t[k] >= tm && a[k] != 0) {
                i--;
                if (id >> i & 1) {
                    if (x[k] == 0) {
                        x ^= a[k];
                    }
                } else {
                    if (x[k] == 1) {
                        x ^= a[k];
                    }
                }
            }
        }
        return x.to_ullong();
    }
};
```

6. 组合数

Lucas定理

$$\text{快速求解}\binom{n}{m} mod\, P$$

```
template <class T>
struct CombLucas {
    T C(int n, int m) {
        if (n < m || m < 0) {
            return 0;
        }
        if (n == m) {
            return 1;
        }
        T res1 = 1, res2 = 1;
        for (int i = 1, j = n; i <= m; i++, j--) {
            res1 *= j;
            res2 *= i;
        }
        return res1 / res2;
    }

    T Lucas(i64 n, i64 m) {
        if (n < P && m < P) {
            return C(n, m);
        }
        return C(n % P, m % P) * Lucas(n / P, m / P);
    }

};
CombLucas<Z> comb1;
```

组合数

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

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb (int n) : Comb() {
        init(n);
    }

    void init (int m) {
        if (m <= n) {
            return;
        }
        _fac.resize (m + 1);
        _invfac.resize (m + 1);
        _inv.resize (m + 1);

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

    T fac (int m) {
        if (m > n) {
            init(2 * m);
        }
        return _fac[m];
    }
    T invfac (int m) {
        if (m > n) {
            init(2 * m);
        }
        return _invfac[m];
    }
    T inv (int m) {
        if (m > n) {
            init(2 * m);
        }
        return _inv[m];
    }
    T C (int n, int m) {
        if (n < m || m < 0) {
            return 0;
        }
        return fac(n) * invfac(m) * invfac(n - m);
    }
    T A (int n, int m) {
        if (n < m || m < 0) {
            return 0;
        }
        return fac(n) * invfac(n - m);
    }
    // C(m, m) + C(m + 1, m) + ... + C(m + n, m) = C(m + 1 + n, m + 1)
    T Csum (int n, int m) {
        return C(m + 1 + n, m + 1);
    }

    T Catalan (int n) {
        return C(2 * n, n) - C(2 * n, n - 1);
    }
};
Comb<Z> comb;
```

组合数常用结论

二项式反演

$$\text{令} f_n \text{为恰好选} n \text{件物品的方案数，} g_n \text{为至多选} n \text{件物品的方案数}$$

$$f_n = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} g_i$$

$$\text{令} f_n \text{为恰好选} n \text{件物品的方案数，} g_n \text{为至少选} n \text{件物品的方案数}$$

$$f_n = \sum_{i=n}^{m} (-1)^{i-n} \binom{i}{n} g_i$$

判断组合数奇偶

$$\text{当} n \wedge m = m, \ \binom{n}{m} \text{为奇数}$$

7. 行列式

```
Z det(std::vector<std::vector<Z>> a) {
    int n = a.size();
    Z ans = 1;
    for (int i = 0; i < n; i++) {
        int j = i;
        while (j < n && a[j][i] == 0) {
            j++;
        }
        if (j == n) return 0;
        if (i != j) {
            std::swap(a[i], a[j]);
            ans *= -1;
        }
        ans *= a[i][i];
        auto v = Z(a[i][i]).inv();
        for (int j = i; j < n; j++) {
            a[i][j] *= v;
        }
        for (int j = i + 1; j < n; j++) {
            Z v = a[j][i];
            for (int k = i; k < n; k++) {
                a[j][k] -= a[i][k] * v;
            }
        }
    }
    return ans;
}
```

7. 莫比乌斯反演

$$F(n)=\sum_{d|n}f(d),\ \ f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$$

$$F(n)=\sum_{n|d}f(d),\ \ f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$$

常用来求解gcd = d的个数, 方法如下:

$$\text{令}F(n)\text{为}gcd\text{是}n\text{的倍数的个数，}f(n)\text{为}gcd\text{是}n\text{的个数，那么有}$$
$$F(n)=\sum_{n|d}f(d)$$
$$\text{通过莫比乌斯反演则有}$$
$$f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$$

求解莫比乌斯函数

```
void init() {
    mu[1] = 1;
    for (int i = 2; i < N; i++) {
        if (!st[i]) {
            primes.push_back(i);
            mu[i] = -1;
        }
        for (int j = 0; i * primes[j] < N; j++) {
            st[i * primes[j]] = true;
            if (i % primes[j] == 0) {
                break;
            }
            mu[i * primes[j]] = -mu[i];
        }
    }
}
```

相关结论

$$令d(x)为x的约数个数，则有d(i \cdot j) = \sum_{x|i} \sum_{y|j} [gcd(x,y) = 1]$$

8. 积性函数

$$对于任意互质的整数a,\ b,\ 都有f(a \cdot b) = f(a) \cdot f(b)$$

9. 斯特林数

第一类斯特林数

```
f[0][0] = 1;
for (int i = 1; i < n; i++) {
    for (int j = 1; j < m; j++) {
        f[i][j] = f[i - 1][j - 1] + (i - 1) * f[i - 1][j];
    }
}
// 1
// 0 1
// 0 1 1
// 0 2 3 1
// 0 6 11 6 1
// 0 24 50 35 10 1
// 0 120 274 225 85 15 1
// 0 720 1764 1624 735 175 21 1
// 0 5040 13068 13132 6769 1960 322 28 1
// 0 40320 109584 118124 67284 22449 4536 546 36 1
```

第二类斯特林数

```
f[0][0] = 1;
for (int i = 1; i < n; i++) {
    for (int j = 1; j < m; j++) {
        f[i][j] = f[i - 1][j - 1] + j * f[i - 1][j];
    }
}
// 1
// 0 1
// 0 1 1
// 0 1 3 1
// 0 1 7 6 1
// 0 1 15 25 10 1
// 0 1 31 90 65 15 1
// 0 1 63 301 350 140 21 1
// 0 1 127 966 1701 1050 266 28 1
// 0 1 255 3025 7770 6951 2646 462 36 1
```

杂项

1. __int128

```
std::ostream &operator<<(std::ostream &os, i128 n) {
    std::string s;
    if (!n) s = "0";
    int flg = 0;
    if (n < 0) {
        flg = 1;
        n *= -1;
    }
    while (n) {
        s += '0' + n % 10;
        n /= 10;
    }
    if (flg) {
        s += '-';
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

2. 常用函数

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

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
__gnu_pbds::tree<pair<int, int>, __gnu_pbds::null_type, less<pair<int, int>>,
           __gnu_pbds::rb_tree_tag,
           __gnu_pbds::tree_order_statistics_node_update>
           rbt;
// order_of_key(x) 返回严格小于x的元素的个数
// find_by_order(x) 返回排名为x的元素的迭代器

// c++位操作内置函数
// __builtin_ffs(x), 与__lg(lowbit(x)) + 1的值相等
// __builtin_clz(x), 返回前缀0的个数
// __builtin_ctz(x), 返回后缀0的个数
// __builtin_popcount(x), x的二进制表示中1的个数
// __builtin_parity(x), x的二进制表示中1的个数的奇偶

#define int long long
#define pb push_back
```

```
#define eb emplace_back
#define printv(a, x) for (int i = x; i < a.size(); i ++ ) \
                      cout << a[i] << " \n"[i == (int)a.size() - 1]
#define printvv(a, x) for (int i = x; i < a.size(); i ++ ) \
                      for (int j = x; j < a[i].size(); j ++ ) \
                        cout << a[i][j] << " \n"[j == (int)a[i].size() - 1]
#define all(x) (x).begin(), (x).end()
#define pq priority_queue
#define umap unordered_map
#define uset unordered_set
#define printd(x, d) cout << fixed << setprecision(d) << (x) << '\n'
#define ne_per(a) next_permutation((a).begin(), (a).end())

using i64 = long long;
using u64 = unsigned long long;
using u32 = unsigned int;
using i128 = __int128;

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

void solve() {
}

signed main() {
    ios_base::sync_with_stdio (false);
    cin.tie (nullptr); cout.tie (nullptr);
    int t = 1;
    // cin >> t;
    while (t -- ) {
        solve();
    }
    return 0;
}
```

3. 求逆序对

```
int tmp[N];
int merge_sort(vector<int> &q, int l, int r) {
    if (l >= r) {
        return 0;
    }
    int mid = l + r >> 1;
    int res = merge_sort(q, l, mid) + merge_sort(q, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r) {
        if (q[i] <= q[j]) {
            tmp[k++] = q[i++];
        } else {
            res += mid - i + 1;
            tmp[k++] = q[j++];
        }
    }
    while (i <= mid) {
        tmp[k++] = q[i++];
    }
    while (j <= r) {
        tmp[k++] = q[j++];
    }
    for (i = l, j = 0; i <= r; i++, j++) {
        q[i] = tmp[j];
    }
    return res;
}
```

计算几何

1. 静态凸包

```
struct Point {
    i64 x;
    i64 y;
    Point() : x{0}, y{0} {}
    Point(i64 x_, i64 y_) : x{x_}, y{y_} {}
};

i64 dot(Point a, Point b) {
    return a.x * b.x + a.y * b.y;
}

i64 cross(Point a, Point b) {
    return a.x * b.y - a.y * b.x;
}

Point operator+(Point a, Point b) {
    return Point(a.x + b.x, a.y + b.y);
}

Point operator-(Point a, Point b) {
    return Point(a.x - b.x, a.y - b.y);
}

auto getHull(std::vector<Point> p) {
    std::sort(p.begin(), p.end(),
    [&](auto a, auto b) {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    });

    std::vector<Point> hi, lo;
    for (auto p : p) {
        while (hi.size() > 1 && cross(hi.back() - hi[hi.size() - 2], p -
hi.back()) >= 0) {
            hi.pop_back();
        }
        while (!hi.empty() && hi.back().x == p.x) {
            hi.pop_back();
        }
        hi.push_back(p);
        while (lo.size() > 1 && cross(lo.back() - lo[lo.size() - 2], p -
lo.back()) <= 0) {
```

```
            lo.pop_back();
        }
        if (lo.empty() || lo.back().x < p.x) {
            lo.push_back(p);
        }
    }

    std::vector<Point> hull;
    int n = p.size();
    for (int i = 0; i < n; i ++ ) {
        while (hull.size() > 1 && cross(hull.back() - hull[hull.size() - 2], p[i]
- hull.back()) <= 0) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }
    int m = hull.size();
    for (int i = n - 1; i >= 0; i -- ) {
        while (hull.size() > m && cross(hull.back() - hull[hull.size() - 2], p[i]
- hull.back()) <= 0) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }
    return std::make_pair(hi, lo);
}

const double inf = INFINITY;
```