图论

SCC

一般

```
struct SCC {
    int n, cnt = 0, cur = 0;
    vector<int> h, t, to;
    vector<int> dfn, low, stk, f;
    vector<bool> in;

    SCC(int n): n(n), h(n, -1), dfn(n, -1), low(n), f(n, -1), in(n, 0) {}

    void tarjan(int u) {
        dfn[u] = low[u] = cur ++;
        stk.push_back(u);
        in[u] = 1;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (dfn[v] == -1) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else if (in[v]) {
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            int t = cnt ++;
            do {
                in[stk.back()] = 0;
                f[stk.back()] = t;
                stk.pop_back();
            } while (in[u]);
        }
    }

    void addEdge(int u, int v) {
        t.push_back(h[u]), h[u] = to.size(), to.push_back(v);
    }

    vector<vector<int>> Graph() {
        vector<vector<int>> g(cnt);
        for (int u = 0; u < n; u += 1) {
            for (int i = h[u]; i != -1; i = t[i]) {
                int v = to[i];
                if (f[u] != f[v]) {
                    g[f[u]].push_back(f[v]);
                }
            }
        }
        return g;
    }
```

```
    vector<array<int, 2>> Edges() {
        vector<array<int, 2>> e;
        for (int u = 0; u < n; u += 1) {
            for (int i = h[u]; i != -1; i = t[i]) {
                int v = to[i];
                if (f[u] != f[v]) {
                    e.push_back({f[u], f[v]});
                }
            }
        }
        sort(e.begin(), e.end());
        e.erase(unique(e.begin(), e.end()), e.end());
        return e;
    }
};
```

割边

```
struct CutEdge {
    int n, tot = -1;
    vector<array<int, 2>> e;
    vector<int> h, t;
    vector<int> dfn, low, ans;

    CutEdge(int n) :n(n), dfn(n, -1), low(n, -1), h(n, -1) {};

    // in 表示从哪条边下来，如果为根，by为一个极大值
    void tarjan(int u, int in = 1E9) {
        dfn[u] = low[u] = ++tot;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = e[i][1];
            if (i == (in ^ 1)) {
                continue;
            }
            if (dfn[v] == -1) {
                tarjan(v, i);
                low[u] = min(low[u], low[v]);
                if (low[v] > dfn[u]) {
                    ans.push_back(i);
                }
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }

    void addEdge(int u, int v) {
        t.push_back(h[u]), h[u] = e.size(); e.push_back({u, v});
        t.push_back(h[v]), h[v] = e.size(); e.push_back({v, u});
    }

    vector<int> &answer() {
        for (int i = 0; i < n; i += 1) {
            if (dfn[i] == -1) {
```

```
                tarjan(i);
            }
        }
        return ans;
    }
};
```

边双

```
struct EDCC {
    int n, cur = -1, cnt = 0;
    vector<array<int, 2>> e;
    vector<int> h, t;
    vector<int> dfn, low, ans, stk, f;

    EDCC(int n) :n(n), dfn(n, -1), low(n, -1), h(n, -1), f(n, -1) {};

    // in 表示从哪条边下来，如果为根，by为一个极大值
    void tarjan(int u, int in = 1E9) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = e[i][1];
            if (i == (in ^ 1)) {
                continue;
            }
            if (dfn[v] == -1) {
                tarjan(v, i);
                low[u] = min(low[u], low[v]);
                if (low[v] > dfn[u]) {
                    ans.push_back(i);
                }
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            int t = cnt ++;
            int v;
            do {
                v = stk.back();
                stk.pop_back();
                f[v] = t;
            } while (v != u);
        }
    }

    void addEdge(int u, int v) {
        t.push_back(h[u]), h[u] = e.size(); e.push_back({u, v});
        t.push_back(h[v]), h[v] = e.size(); e.push_back({v, u});
    }

    vector<int> &answer() {
        for (int i = 0; i < n; i += 1) {
```

```
            if (dfn[i] == -1) {
                tarjan(i);
            }
        }
        return ans;
    }
};
```

割点

```
struct CutPoint {
    int n, cur = 0;
    vector<int> h, t, to, dfn, low;
    vector<bool> cut;

    CutPoint(int n): n(n), h(n, -1), dfn(n, -1), low(n, -1), cut(n) {};

    // 传参时 tarjan(u, u)
    void tarjan(int u, int r) {
        dfn[u] = low[u] = cur ++;
        int child = 0;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (dfn[v] == -1) {
                tarjan(v, r);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    ++child;
                    if (u != r || child > 1) {
                        cut[u] = true;
                    }
                }
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }

    void addEdge(int u, int v) {
        t.push_back(h[u]), h[u] = to.size(); to.push_back(v);
        t.push_back(h[v]), h[v] = to.size(); to.push_back(u);
    }
};
```

点双

```
struct VDCC {
    int n, cur = 0, cnt = 0;
    vector<int> h, t, to, dfn, low, stk;
    vector<bool> cut;
    vector<vector<int>> sorted;
```

```
    VDCC(int n): n(n), h(n, -1), dfn(n, -1), low(n, -1), cut(n) {};

    // 传参时 tarjan(u, u)
    void tarjan(int u, int r) {
        dfn[u] = low[u] = cur ++;
        if (h[u] == -1) {
            debug(u);
            int t = cnt ++;
            sorted.push_back({u});
            return;
        }
        stk.push_back(u);
        int child = 0;
        for (int i = h[u]; i != -1; i = t[i]) {
            int v = to[i];
            if (dfn[v] == -1) {
                tarjan(v, r);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    ++child;
                    if (u != r || child > 1) {
                        cut[u] = true;
                    }
                    int t = cnt ++;
                    sorted.emplace_back();
                    int x;
                    do {
                        int x = stk.back();
                        stk.pop_back();
                        sorted[t].push_back(x);
                    } while (x != v);
                    sorted[t].push_back(u);
                }
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }

    void addEdge(int u, int v) {
        t.push_back(h[u]), h[u] = to.size(); to.push_back(v);
        t.push_back(h[v]), h[v] = to.size(); to.push_back(u);
    }
};
```

倍增lca

```
template <class E>
struct csr {
    vector<int> s;
    vector<E> adj;
    csr() {}
    csr(int n, const vector<pair<int, E>>& g)
        : s(n + 1), adj(g.size()) {
        for (auto e : g) {
```

```
            s[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            s[i] += s[i - 1];
        }
        auto c = s;
        for (auto e : g) {
            adj[c[e.first]++] = e.second;
        }
    }
};

// static unweighted tree
template<int N>
struct multiTree {
    int n;
    int K;
    vector<array<int, __lg(N) + 1>> p;
    struct Edge {
        int v; i64 w;
    };
    vector<pair<int, int>> e;
    csr<int> g;
    vector<int> dep;
    multiTree(int n): n(n), dep(n), p(n), K(__lg(n) + 1) {}
    void addEdge(int u, int v) {
        e.push_back({u, v});
        e.push_back({v, u});
    }
    void work(int r = 0) {
        g = csr<int>(n, e);
        dfs(r, r);
    }
    void dfs(int u, int f) {
        p[u][0] = f;
        for (int i = 1; i < K; i += 1) {
            p[u][i] = p[p[u][i - 1]][i - 1];
        }
        for (int i = g.s[u]; i < g.s[u + 1]; i += 1) {
            auto v = g.adj[i];
            if (v == f) {
                continue;
            }
            dep[v] = dep[u] + 1;
            dfs(v, u);
        }
    }
    int jump(int u, int d) {
        for (int i = dep[u] - d; i > 0; i -= 1 << __lg(i)) {
            u = p[u][__lg(i)];
        }
        return u;
    }
    int lca(int u, int v) {
        if (dep[u] < dep[v]) {
            swap(u, v);
```

```
        }
        u = jump(u, dep[v]);
        if (u == v) {
            return u;
        }
        for (int i = __lg(dep[u]); i >= 0; i -= 1) {
            if (p[u][i] != p[v][i]) {
                u = p[u][i], v = p[v][i];
            }
        }
        return p[u][0];
    }
    int dis1(int u, int v) {
        int m = lca(u, v);
        return dep[u] + dep[v] - 2 * dep[m];
    }
};
```

带权

```
template <class E>
struct csr {
    vector<int> s;
    vector<E> adj;
    csr() {}
    csr(int n, const vector<pair<int, E>>& g)
        : s(n + 1), adj(g.size()) {
        for (auto e : g) {
            s[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            s[i] += s[i - 1];
        }
        auto c = s;
        for (auto e : g) {
            adj[c[e.first]++] = e.second;
        }
    }
};

// static weighted tree
template<int N = int(1E6)>
struct multiTree {
    int n;
    int K;
    vector<array<int, __lg(N) + 1>> p;
    struct Edge {
        int v; i64 w;
        friend ostream &operator<<(ostream &cout, Edge u) {
            return cout << "(" << u.v << ", " << u.w << ")";
        }
    };
    vector<pair<int, Edge>> e;
    csr<Edge> g;
    vector<int> dep;
```

```
    vector<i64> dep2;
    multiTree() {}
    multiTree(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        K = __lg(n) + 1;
        p.assign(n, {});
        dep.assign(n, 0);
        dep2.assign(n, 0);
    }
    void addEdge(int u, int v, auto w) {
        e.push_back({u, {v, w}});
        e.push_back({v, {u, w}});
    }
    void work(int r = 0) {
        g = csr<Edge>(n, e);
        dfs(r, r);
    }
    void dfs(int u, int f) {
        p[u][0] = f;
        for (int i = 1; i < K; i += 1) {
            p[u][i] = p[p[u][i - 1]][i - 1];
        }
        for (int i = g.s[u]; i < g.s[u + 1]; i += 1) {
            auto [v, w] = g.adj[i];
            if (v == f) {
                continue;
            }
            dep[v] = dep[u] + 1;
            dep2[v] = dep2[u] + w;
            dfs(v, u);
        }
    }
    int jump(int u, int d) {
        for (int i = dep[u] - d; i > 0; i -= 1 << __lg(i)) {
            u = p[u][__lg(i)];
        }
        return u;
    }
    int lca(int u, int v) {
        if (dep[u] < dep[v]) {
            swap(u, v);
        }
        u = jump(u, dep[v]);
        if (u == v) {
            return u;
        }
        for (int i = __lg(dep[u]); i >= 0; i -= 1) {
            if (p[u][i] != p[v][i]) {
                u = p[u][i], v = p[v][i];
            }
        }
        return p[u][0];
    }
```

```
    int dist(int u, int v) {
        int m = lca(u, v);
        return dep[u] + dep[v] - 2 * dep[m];
    }
    int dis2(int u, int v) {
        int m = lca(u, v);
        return dep2[u] + dep2[v] - 2 * dep[m];
    }
};
```

虚树

```
auto merge = [&] (int x, int y) {
    return std::max(x, y);
};
struct Tree {
    int n, cur;
    vector<vector<int>> adj;
    vector<int> dfn, dep, siz, fa, seq, dnp;
    SparseTable<int, decltype(::merge)> rmq;
    Tree() = default;
    Tree(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        dfn.assign(n, 0);
        dep.assign(n, 0);
        siz.assign(n, 1);
        fa.assign(n, 0);
        seq.assign(n, 0);
        dnp.assign(n, 0);
        cur = 0;
        dfs(root, root);
        rmq.init(dnp, ::merge);
    }
    void dfs(int now, int f) {
        dnp[cur] = dfn[f];
        seq[dfn[now] = cur ++] = now;
        dep[now] = dep[f] + 1;
        fa[now] = f;
        for (auto here : adj[now]) {
            if (here == f) continue;
            dfs(here, now);
            siz[now] += siz[here];
        }
```

```
    }
    int lca (int lhs, int rhs) {
        if (lhs == rhs) return lhs;
        if ((lhs = dfn[lhs]) > (rhs = dfn[rhs])) swap(lhs, rhs);
        return seq[rmq(lhs + 1, rhs + 1)];
    }
};

constexpr int maxTop = 3e5;
int stk[maxTop];
struct virTree : public Tree {
    virTree(int n) : vTree(n) {}
    vector<vector<int>> vTree;
    vector<vector<int>> &build_virtual_tree(vector<int> &key) {
        sort(key.begin(), key.end(), [&] (int x, int y) {return dfn[x] <
dfn[y];});
        int Top = 0;
        vTree[0].clear();
        for (int i = key[0] == 0; i < key.size(); i += 1) {
            int Lca = lca(key[i], stk[Top]);
            if (Lca != stk[Top]) {
                while(dfn[Lca] < dfn[stk[Top - 1]]) {
                    vTree[stk[Top - 1]].push_back(stk[Top --]);
                }
                if (Lca != stk[Top - 1]) {
                    vTree[Lca].clear();
                    vTree[Lca].push_back(stk[Top]); stk[Top] = Lca;
                } else {
                    vTree[Lca].push_back(stk[Top --]);
                }
            }
            vTree[key[i]].clear();
            stk[++ Top] = key[i];
        }
        for (int i = 0; i < Top; i += 1) {
            vTree[stk[i]].push_back(stk[i + 1]);
        }
        return vTree;
    }
};
```

重链剖分

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

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
```

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

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

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

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

        int d = dep[u] - k;

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

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

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

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

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

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

重链剖分套线段树

```
template<typename Info, typename Tag>
struct HLD_Seg : public HLD, LazySegmentTree<Info, Tag> {
    using LazySegmentTree<Info, Tag>::rangeApply, LazySegmentTree<Info,
Tag>::rangeQuery;
    HLD_Seg(int n) {
        init(n);
    }
    void init(int n) {
        HLD::init(n);
    }
    void Work(int root, const vector<Info> &a) {
        HLD::work(root);
        vector<Info> b(n);
        for (int i = 0; i < n; i += 1) {
            b[in[i]] = a[i];
        }
        LazySegmentTree<Info, Tag>::init(b);
    }
    void LineApply(int u, int v, Tag t) {
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) {
                swap(u, v);
            }
            rangeApply(in[top[u]], in[u] + 1, t);
            u = parent[top[u]];
        }
        if (in[u] > in[v]) {
            swap(u, v);
        }
        rangeApply(in[u], in[v] + 1, t);
    }
    Info LineQuery(int u, int v) {
        Info ans = Info();
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) {
                swap(u, v);
            }
            ans = Info::merge(ans, rangeQuery(in[top[u]], in[u] + 1));
            u = parent[top[u]];
        }
        if (in[u] > in[v]) {
            swap(u, v);
        }
        ans = Info::merge(ans, rangeQuery(in[u], in[v] + 1));
        return ans;
    }
    void SubApply(int u, Tag t, int r = 0) {
        if (u == r) {
            rangeApply(0, n, t);
        } else if (isAncester(u, r)) {
            if (top[u] == top[r]) {
                r = seq[in[u] + 1];
            } else {
```

```
                while (top[parent[top[r]]] != top[u]) {
                    r = parent[top[r]];
                }
                r = top[r];
                if (parent[r] != u) {
                    r = seq[in[u] + 1];
                }
            }
            rangeApply(0, in[r], t);
            rangeApply(out[r], n, t);
        } else {
            rangeApply(in[u], out[u], t);
        }
    }
    Info SubQuery(int u, int r = 0) {
        Info ans = Info();
        if (u == r) {
            return ans = rangeQuery(0, n);
        } else if (isAncester(u, r)) {
            if (top[u] == top[r]) {
                r = seq[in[u] + 1];
            } else {
                while (top[parent[top[r]]] != top[u]) {
                    r = parent[top[r]];
                }
                r = top[r];
                if (parent[r] != u) {
                    r = seq[in[u] + 1];
                }
            }
            ans = Info::merge(rangeQuery(0, in[r]), rangeQuery(out[r], n));
        } else {
            return ans = rangeQuery(in[u], out[u]);
        }
        return ans;
    }
};
```

流

网络流

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

    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
```

```
    std::vector<int> cur, h;

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

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

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

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

```
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }

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

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

网络流前向星

```
template<class T>
struct MaxFlow {
    int n;
    vector<int> r, t, to, h, cur;
    vector<T> c;
    MaxFlow(int n, int m = 0) {
        init(n, m);
```

```
    }
    void init(int n, int m = 0) {
        this->n = n;
        r.assign(n, -1);
        h.assign(n, -1);
        cur.assign(n, 0);
        t.reserve(2 * m);
        to.reserve(2 * m);
        c.reserve(2 * m);
    }
    void addEdge(int u, int v, T cap) {
        t.push_back(r[u]), r[u] = to.size(), to.push_back(v), c.push_back(cap);
        t.push_back(r[v]), r[v] = to.size(), to.push_back(u), c.push_back(0);
    }
    bool bfs(int s, int e) {
        fill(h.begin(), h.end(), -1);
        queue<int> q;
        h[s] = 0;
        cur[s] = r[s];
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = r[u]; ~i; i = t[i]) {
                int v = to[i];
                T cap = c[i];
                if (cap > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    cur[v] = r[v];
                    if (v == e) {
                        return true;
                    }
                    q.push(v);
                }
            }
        }
        return false;
    }
    T dfs(int u, int e, T f) {
        if (u == e) {
            return f;
        }
        T r = f;
        for (int &i = cur[u]; ~i; i = t[i]) {
            int v = to[i];
            T cap = c[i];
            if (cap > 0 && h[v] == h[u] + 1) {
                T k = dfs(v, e, min(cap, r));
                if (k == 0) {
                    h[v] = -1;
                }
                c[i] -= k;
                c[i ^ 1] += k;
                r -= k;
                if (r == 0) {
                    return f;
```

```
                }
            }
        }
        return f - r;
    }
    T flow(int s, int e) {
        T ans = 0;
        while (bfs(s, e)) {
            ans += dfs(s, e, 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;
        friend ostream &operator<<(ostream &cout, Edge u) {
            return cout << '{' << u.from << ", " << u.to << ", " << u.cap << ", "
<< u.flow << "}";
        }
    };
    vector<Edge> edges() {
        vector<Edge> a;
        for (int i = 0; i < t.size(); i += 2) {
            Edge x;
            x.from = to[i + 1];
            x.to = to[i];
            x.cap = c[i] + c[i + 1];
            x.flow = c[i + 1];
            a.push_back(x);
        }
        return a;
    }
};
```

网络流未封装

```
using T = int;

constexpr int N = 2e5 + 2, M = 2 * N + 1e5;
int head[N], nxt[2 * M], to[2 * M];
T cap[2 * M];
int cur = 0;
int _n = 0;
int h[N], now[N];
```

```
void init(int n) {
    fill(head, head + n, -1);
    _n = n;
    cur = -1;
}

void addEdge(int u, int v, T c) {
    nxt[++cur] = head[u], to[cur] = v, cap[cur] = c, head[u] = cur;
    nxt[++cur] = head[v], to[cur] = u, cap[cur] = 0, head[v] = cur;
}


bool bfs(int s, int t) {
    fill(h, h + _n, -1);
    queue<int> q;
    h[s] = 0;
    now[s] = head[s];
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = nxt[i]) {
            int v = to[i];
            T c = cap[i];
            if (c > 0 && h[v] == -1) {
                h[v] = h[u] + 1;
                now[v] = head[v];
                if (v == t) {
                    return true;
                }
                q.push(v);
            }
        }
    }
    return false;
}

T dfs(int u, int t, T f) {
    if (u == t) {
        return f;
    }
    T r = f;
    for (int &i = now[u]; ~i; i = nxt[i]) {
        int v = to[i];
        T c = cap[i];
        if (c > 0 && h[v] == h[u] + 1) {
            T k = dfs(v, t, min(c, r));
            if (k == 0) {
                h[v] = -1;
            }
            cap[i] -= k;
            cap[i ^ 1] += k;
            r -= k;
            if (r == 0) {
                return f;
```

```
            }
        }
    }
    return f - r;
}

T flow(int s, int t) {
    T ans = 0;
    while (bfs(s, t)) {
        ans += dfs(s, t, std::numeric_limits<T>::max());
    }
    return ans;
}

struct Edge {
    int from;
    int to;
    T cap;
    T flow;
};

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

vector<Edge> Edges() {
    vector<Edge> a;
    for (int i = 0; i < cur; i += 2) {
        Edge x;
        x.from = to[i + 1];
        x.to = to[i];
        x.cap = cap[i] + cap[i + 1];
        x.flow = cap[i + 1];
        a.push_back(x);
    }
    return a;
}
```

费用流

```
struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<i64> h, dis;
```

```
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<i64>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64,
int>>, std::greater<std::pair<i64, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            i64 d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] < d) continue;
            for (int i : g[u]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<i64>::max();
    }
    MCFGraph(int n) : n(n), g(n) {}
    void addEdge(int u, int v, int c, int f) {
        g[u].push_back(e.size());
        e.emplace_back(v, c, f);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -f);
    }
    std::pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            int aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug,
e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
        }
        return std::make_pair(flow, cost);
    }
};
```

费用流前向星

```
template<typename T>
using min_heap = priority_queue<T, vector<T>, greater<T>>;

struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
        template<class ostream>
        friend ostream& operator<<(ostream& cout, Edge e) {
            return cout << "{" << e.v << ", " << e.c << ", " << e.f << "}";
        }
    };
    int n;
    vector<Edge> e;
    vector<int> r, t, pre;
    vector<i64> h, dis;
    bool dijkstra(int S, int T) {
        dis.assign(n, numeric_limits<i64>::max());
        pre.assign(n, -1);
        min_heap<pair<i64, int>> q;
        dis[S] = 0;
        q.emplace(0, S);
        while (!q.empty()) {
            i64 d = q.top().first;
            int u = q.top().second;
            q.pop();
            if (dis[u] < d) continue;
            for (int i = r[u]; ~i; i = t[i]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    q.emplace(dis[v], v);
                }
            }
        }
        return dis[T] != numeric_limits<i64>::max();
    }
    MCFGraph(int n, int m = 0) : n(n), r(n, -1) {
        t.reserve(2 * m), e.reserve(2 * m);
    }
    void addEdge(int u, int v, int c, int f) {
        // cerr << u << ' ' << v << ' ' << c << '-' << f << '\n';
        t.push_back(r[u]), r[u] = e.size(), e.emplace_back(v, c, f);
        t.push_back(r[v]), r[v] = e.size(), e.emplace_back(u, 0, -f);
    }
    pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
```

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

    struct _Edge {
        int from;
        int to;
        int cap;
        int flow;
        int cost;
    };

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

pair<bool, i64> MCFF(vector<array<int, 5>> &e, int n) {
    int N = n + 2;
    int s = N - 2, t = s + 1;
    vector<int> d(n);
    MCFGraph g(N);
    for (auto [u, v, L, U, c] : e) {
        g.addEdge(u, v, U - L, c);
        d[u] -= L;
        d[v] += L;
    }
    for (int i = 0; i < n; i += 1) {
        if (d[i] > 0) {
            g.addEdge(s, i, d[i], 0);
        } else {
            g.addEdge(i, t, -d[i], 0);
        }
    }
    auto [flow, cost] = g.flow(s, t);
    bool ok = 1;
```

```
    for (int i = g.r[s]; ~i; i = g.t[i]) {
        ok &= g.e[i].c == 0;
    }
    for (int i = g.r[t]; ~i; i = g.t[i]) {
        ok &= g.e[i ^ 1].c == 0;
    }
    return {ok, cost};
}
```

费用流多类型EK

```
template<typename T>
using min_heap = priority_queue<T, vector<T>, greater<T>>;

template<typename T>
struct Ceil {
    constexpr static T max() {
        return numeric_limits<T>::max();
    }
};

using f64 = double;
template<>
struct Ceil<f64> {
    constexpr static f64 max() {
        return 1e9;
    }
};

template<typename Cap, typename Cost>
struct MCFGraph {
    struct Edge {
        int v; Cap c; Cost f;
        Edge(int v, Cap c, Cost f) : v(v), c(c), f(f) {}
        template<class ostream>
        friend ostream& operator<<(ostream& cout, Edge e) {
            return cout << "{" << e.v << ", " << e.c << ", " << e.f << "}";
        }
    };
    int n;
    vector<Edge> e;
    vector<int> r, t, pre;
    vector<Cost> h, dis;
    bool dijkstra(int S, int T) {
        dis.assign(n, Ceil<Cost>::max());
        pre.assign(n, -1);
        min_heap<pair<Cost, Cap>> q;
        dis[S] = 0;
        q.emplace(0, S);
        while (!q.empty()) {
            Cost d = q.top().first;
            Cap u = q.top().second;
            q.pop();
```

```
            if (dis[u] < d) continue;
            for (int i = r[u]; ~i; i = t[i]) {
                int v = e[i].v;
                Cap c = e[i].c;
                Cost f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    q.emplace(dis[v], v);
                }
            }
        }
        return dis[T] != Ceil<Cost>::max();
    }
    MCFGraph(int n, int m = 0) : n(n), r(n, -1) {
        t.reserve(2 * m), e.reserve(2 * m);
    }
    void addEdge(int u, int v, Cap c, Cost f) {
        // cerr << u << ' ' << v << ' ' << c << '-' << f << '\n';
        t.push_back(r[u]), r[u] = e.size(), e.emplace_back(v, c, f);
        t.push_back(r[v]), r[v] = e.size(), e.emplace_back(u, 0, -f);
    }
    pair<Cap, Cost> flow(int s, int t) {
        Cap flow = 0;
        Cost cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            Cap aug = Ceil<Cap>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug,
e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return make_pair(flow, cost);
    }

    struct _Edge {
        int from;
        int to;
        Cap cap;
        Cap flow;
        Cost cost;
    };

    std::vector<_Edge> edges() {
        std::vector<_Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            _Edge x;
            x.from = e[i + 1].v;
            x.to = e[i].v;
            x.cap = e[i].c + e[i + 1].c;
```

```
            x.flow = e[i + 1].c;
            x.cost = e[i].f;
            a.push_back(x);
        }
        return a;
    }
};
```

费用流原始对偶

```
template<class E>
struct csr {
    vector<int> h;
    vector<E> e;

    csr(int n, const vector<pair<int, E>> &edges)
            : h(n + 1), e(edges.size()) {
        for (auto u : edges)
            h[u.first + 1]++;
        for (int i = 1; i <= n; i++)
            h[i] += h[i - 1];
        auto c = h;
        for (auto u : edges)
            e[c[u.first]++] = u.second;
    }
};


struct MCFGraph {
    MCFGraph() {}

    MCFGraph(int n) : n(n) {}

    void addEdge(int u, int to, int c, i64 p) {
        E.push_back({u, to, c, 0, p});
    }

    struct Edge {
        int u, v;
        int c, f;
        i64 p;
    };

    vector<Edge> Edges() { return E; }
    static constexpr int inf = numeric_limits<int>::max();
    using Ans = pair<int, i64>;

    Ans flow(int s, int t, int f = inf) {
        return slope(s, t, f).back();
    }

    vector<Ans> slope(int s, int t, int f = inf) {
        int m = E.size();
        vector<int> id(m);
```

```
        auto g = [&]() {
            vector<int> d(n), rid(m);
            vector<pair<int, _Edge>> elist;
            elist.reserve(2 * m);
            for (int i = 0; i < m; i++) {
                auto e = E[i];
                id[i] = d[e.u]++;
                rid[i] = d[e.v]++;
                elist.push_back({e.u, {e.v, -1, e.c - e.f, e.p}});
                elist.push_back({e.v, {e.u, -1, e.f, -e.p}});
            }
            auto g = csr<_Edge>(n, elist);
            for (int i = 0; i < m; i++) {
                auto e = E[i];
                id[i] += g.h[e.u];
                rid[i] += g.h[e.v];
                g.e[id[i]].rev = rid[i];
                g.e[rid[i]].rev = id[i];
            }
            return g;
        }();

        auto ans = slope(g, s, t, f);

        for (int i = 0; i < m; i++) {
            E[i].f = E[i].c - g.e[id[i]].c;
        }

        return ans;
    }

    int n;
    vector<Edge> E;

    struct _Edge {
        int v, rev;
        int c;
        i64 p;
    };

    vector<Ans> slope(csr<_Edge> &g, int s, int t, int f) {
        vector<array<i64, 2>> d(n);
        vector<int> pre(n), qm;
        vector<bool> vis(n);
        vector<pair<i64, int>> q;
        auto cmp = greater<pair<i64, int>>();
        auto ref = [&]() {
            for (int i = 0; i < n; i++) {
                d[i][1] = numeric_limits<i64>::max();
            }
            fill(vis.begin(), vis.end(), false);
            qm.clear();
            q.clear();

            int r = 0;
```

```
            d[s][1] = 0;
            qm.push_back(s);
            while (!qm.empty() || !q.empty()) {
                int v;
                if (!qm.empty()) {
                    v = qm.back();
                    qm.pop_back();
                } else {
                    while (r < q.size()) {
                        r++;
                        push_heap(q.begin(), q.begin() + r, cmp);
                    }
                    v = q.front().second;
                    pop_heap(q.begin(), q.end(), cmp);
                    q.pop_back();
                    r--;
                }
                if (vis[v]) continue;
                vis[v] = true;
                if (v == t) break;
                i64 u = d[v][0], dis = d[v][1];
                for (int i = g.h[v]; i < g.h[v + 1]; i++) {
                    auto e = g.e[i];
                    if (!e.c) continue;
                    i64 p = e.p - d[e.v][0] + u;
                    if (d[e.v][1] - dis > p) {
                        i64 to = dis + p;
                        d[e.v][1] = to;
                        pre[e.v] = e.rev;
                        if (to == dis) {
                            qm.push_back(e.v);
                        } else {
                            q.push_back({to, e.v});
                        }
                    }
                }
            }
            if (!vis[t]) {
                return false;
            }

            for (int v = 0; v < n; v++) {
                if (!vis[v]) continue;
                d[v][0] -= d[t][1] - d[v][1];
            }
            return true;
        };
        int r = 0;
        i64 p = 0, cf = -1;
        vector<Ans> ans(1);
        while (r < f) {
            if (!ref()) break;
            int c = f - r;
            for (int v = t; v != s; v = g.e[pre[v]].v) {
                c = min(c, g.e[g.e[pre[v]].rev].c);
            }
```

```
            for (int v = t; v != s; v = g.e[pre[v]].v) {
                auto &e = g.e[pre[v]];
                e.c += c;
                g.e[e.rev].c -= c;
            }
            i64 D = -d[s][0];
            r += c;
            p += c * D;
            if (cf == D) {
                ans.pop_back();
            }
            ans.push_back({r, p});
            cf = D;
        }
        return ans;
    }
};
```

费用流多类型原始对偶

```
template<class E>
struct csr {
    vector<int> h;
    vector<E> e;

    csr(int n, const vector<pair<int, E>> &edges)
            : h(n + 1), e(edges.size()) {
        for (auto u : edges) {
            h[u.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            h[i] += h[i - 1];
        }
        auto c = h;
        for (auto u : edges) {
            e[c[u.first]++] = u.second;
        }
    }
};


template<typename Cap, typename Cost>
struct MCFGraph {
    MCFGraph() {}

    MCFGraph(int n) : n(n) {}

    void addEdge(int u, int to, Cap c, Cost p) {
        E.push_back({u, to, c, 0, p});
    }

    struct Edge {
        int u, v;
        Cap c, f;
        Cost p;
```

```
};

vector<Edge> Edges() { return E; }
static constexpr Cap inf = numeric_limits<Cap>::max();
using Ans = pair<Cap, Cost>;

Ans flow(int s, int t, Cap f = inf) {
    return slope(s, t, f).back();
}

vector<Ans> slope(int s, int t, Cap f = inf) {
    int m = E.size();
    vector<int> id(m);

    auto g = [&]() {
        vector<int> d(n), rid(m);
        vector<pair<int, _Edge>> elist;
        elist.reserve(2 * m);
        for (int i = 0; i < m; i++) {
            auto e = E[i];
            id[i] = d[e.u]++;
            rid[i] = d[e.v]++;
            elist.push_back({e.u, {e.v, -1, e.c - e.f, e.p}});
            elist.push_back({e.v, {e.u, -1, e.f, -e.p}});
        }
        auto g = csr<_Edge>(n, elist);
        for (int i = 0; i < m; i++) {
            auto e = E[i];
            id[i] += g.h[e.u];
            rid[i] += g.h[e.v];
            g.e[id[i]].rev = rid[i];
            g.e[rid[i]].rev = id[i];
        }
        return g;
    }();

    auto ans = slope(g, s, t, f);

    for (int i = 0; i < m; i++) {
        E[i].f = E[i].c - g.e[id[i]].c;
    }

    return ans;
}

int n;
vector<Edge> E;

struct _Edge {
    int v, rev;
    Cap c;
    Cost p;
};

vector<Ans> slope(csr<_Edge> &g, int s, int t, Cap f) {
    vector<array<Cost, 2>> d(n);
```

```
        vector<int> pre(n), qm;
        vector<bool> vis(n);
        vector<pair<Cost, int>> q;
        auto cmp = greater<pair<Cost, int>>();
        auto ref = [&]() {
            for (int i = 0; i < n; i++) {
                d[i][1] = numeric_limits<Cost>::max();
            }
            fill(vis.begin(), vis.end(), false);
            qm.clear();
            q.clear();

            size_t r = 0;

            d[s][1] = 0;
            qm.push_back(s);
            while (!qm.empty() || !q.empty()) {
                int v;
                if (!qm.empty()) {
                    v = qm.back();
                    qm.pop_back();
                } else {
                    while (r < q.size()) {
                        r++;
                        push_heap(q.begin(), q.begin() + r, cmp);
                    }
                    v = q.front().second;
                    pop_heap(q.begin(), q.end(), cmp);
                    q.pop_back();
                    r--;
                }
                if (vis[v]) continue;
                vis[v] = true;
                if (v == t) break;
                Cost u = d[v][0], dis = d[v][1];
                for (int i = g.h[v]; i < g.h[v + 1]; i++) {
                    auto e = g.e[i];
                    if (!e.c) continue;
                    Cost p = e.p - d[e.v][0] + u;
                    if (d[e.v][1] - dis > p) {
                        Cost to = dis + p;
                        d[e.v][1] = to;
                        pre[e.v] = e.rev;
                        if (to == dis) {
                            qm.push_back(e.v);
                        } else {
                            q.push_back({to, e.v});
                        }
                    }
                }
            }
            if (!vis[t]) {
                return false;
            }

            for (int v = 0; v < n; v++) {
```

```
                if (!vis[v]) continue;
                d[v][0] -= d[t][1] - d[v][1];
            }
            return true;
        };
        Cap r = 0;
        Cost p = 0, cf = -1;
        vector<Ans> ans(1);
        while (r < f) {
            if (!ref()) break;
            Cap c = f - r;
            for (int v = t; v != s; v = g.e[pre[v]].v) {
                c = min(c, g.e[g.e[pre[v]].rev].c);
            }
            for (int v = t; v != s; v = g.e[pre[v]].v) {
                auto &e = g.e[pre[v]];
                e.c += c;
                g.e[e.rev].c -= c;
            }
            Cost D = -d[s][0];
            r += c;
            p += c * D;
            if (cf == D) {
                ans.pop_back();
            }
            ans.push_back({r, p});
            cf = D;
        }
        return ans;
    }
};
```

单纯形

```
struct MCFGraph {
    struct Edge {
        int u, v, nxt;
        i64 f, w;
    };
    static constexpr int inf  = numeric_limits<int>::max();
    vector<Edge> E;
    vector<int> fa, fe, cir, tag, H;
    vector<i64> pre;

    MCFGraph(int n, int m = 0): fa(n), fe(n), cir(n), tag(n), H(n), pre(n), E(2)
{
        E.reserve(2 * m + 4);
    }

    int tot = 1;

    void addEdge(int u, int v, i64 f, i64 w) {
        E.push_back({u, v, H[u], f, + w}), H[u] = ++ tot;
```

```
    E.push_back({v, u, H[v], 0, - w}), H[v] = ++ tot;
}

int now = 0;

void InitZCT(int x, int e, int nod = 1) {
    fa[x] = E[fe[x] = e].u, tag[x] = nod;
    for (int i = H[x]; i; i = E[i].nxt)
        if(tag[E[i].v] != nod && E[i].f)
            InitZCT(E[i].v, i, nod);
}

i64 sum(int x) {
    if(tag[x] == now) return pre[x];
    return tag[x] = now, pre[x] = sum(fa[x]) + E[fe[x]].w;
}

i64 PushFlow(int x) {
    int rt = E[x].u, lca = E[x].v, p = 2, del = 0, cnt = 0;
    ++ now;
    while(rt) tag[rt] = now, rt = fa[rt];
    while(tag[lca] != now) tag[lca] = now, lca = fa[lca];

    i64 f = E[x].f, cost = 0;

    for (int u = E[x].u; u != lca; u = fa[u]) {
        cir[++ cnt] = fe[u];
        if(E[fe[u]].f < f) del = u, p = 0, f = E[fe[u]].f;
    }

    for (int u = E[x].v; u != lca; u = fa[u]) {
        cir[++ cnt] = fe[u] ^ 1;
        if(E[fe[u] ^ 1].f < f) del = u, p = 1, f = E[fe[u] ^ 1].f;
    }

    cir[++ cnt] = x;

    for (int i = 1; i <= cnt; ++ i)
        cost += E[cir[i]].w * f, E[cir[i]].f -= f, E[cir[i] ^ 1].f += f;

    if(p == 2) return cost;
    int u = E[x].u, v = E[x].v;
    if(p == 1) std::swap(u, v);
    int le = x ^ p, lu = v, tmp;

    while(lu != del) {
        le ^= 1, -- tag[u], std::swap(fe[u], le);
        tmp = fa[u], fa[u] = lu, lu = u, u = tmp;
    }

    return cost;
}


pair<i64, i64> flow(int S, int T) {
    addEdge(T, S, inf, - inf);
```

```
        InitZCT(T, 0, ++ now);

        tag[T] = ++ now, fa[T] = 0;

        bool Run = 1;
        i64 MinC = 0;

        while(Run) {
            Run = 0;
            for (int i = 2; i <= tot; ++ i)
                if(E[i].f && E[i].w + sum(E[i].u) - sum(E[i].v) < 0)
                    MinC += PushFlow(i), Run = 1;
        }

        MinC += E[tot].f * inf;

        return {E[tot].f, MinC};
    }
};
```

单纯形未封装

```
using T = int;

constexpr int N = 2e5 + 2, M = 2 * N + 1e5;
int head[N], nxt[2 * M], to[2 * M];
T cap[2 * M];
int cur = 0;
int _n = 0;
int h[N], now[N];

void init(int n) {
    fill(head, head + n, -1);
    _n = n;
    cur = -1;
}

void addEdge(int u, int v, T c) {
    nxt[++cur] = head[u], to[cur] = v, cap[cur] = c, head[u] = cur;
    nxt[++cur] = head[v], to[cur] = u, cap[cur] = 0, head[v] = cur;
}


bool bfs(int s, int t) {
    fill(h, h + _n, -1);
    queue<int> q;
    h[s] = 0;
    now[s] = head[s];
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = nxt[i]) {
            int v = to[i];
            T c = cap[i];
```

```
            if (c > 0 && h[v] == -1) {
                h[v] = h[u] + 1;
                now[v] = head[v];
                if (v == t) {
                    return true;
                }
                q.push(v);
            }
        }
    }
    return false;
}

T dfs(int u, int t, T f) {
    if (u == t) {
        return f;
    }
    T r = f;
    for (int &i = now[u]; ~i; i = nxt[i]) {
        int v = to[i];
        T c = cap[i];
        if (c > 0 && h[v] == h[u] + 1) {
            T k = dfs(v, t, min(c, r));
            if (k == 0) {
                h[v] = -1;
            }
            cap[i] -= k;
            cap[i ^ 1] += k;
            r -= k;
            if (r == 0) {
                return f;
            }
        }
    }
    return f - r;
}

T flow(int s, int t) {
    T ans = 0;
    while (bfs(s, t)) {
        ans += dfs(s, t, std::numeric_limits<T>::max());
    }
    return ans;
}

struct Edge {
    int from;
    int to;
    T cap;
    T flow;
};

vector<bool> minCut() {
    vector<bool> c(_n);
    for (int i = 0; i < _n; i ++) {
        c[i] = (h[i] != -1);
```

```
    }
    return c;
}

vector<Edge> Edges() {
    vector<Edge> a;
    for (int i = 0; i < cur; i += 2) {
        Edge x;
        x.from = to[i + 1];
        x.to = to[i];
        x.cap = cap[i] + cap[i + 1];
        x.flow = cap[i + 1];
        a.push_back(x);
    }
    return a;
}
```

笛卡尔树

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

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

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

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

板题实现

欧拉图

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

# ifdef LOCAL
    # include "C:\Users\Kevin\Desktop\demo\save\debug.h"
# else
# define debug(...) 114514
# define ps 114514
# endif

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

void solve() {
    int n, m; cin >> n >> m;
    vector<vector<int>> a(n);
    vector<int> in(n);
    for (int i = 0; i < m; i += 1) {
        int u, v; cin >> u >> v; -- u, -- v;
        a[u].push_back(v);
        in[v] ++;
    }
    array<int, 2> cnt{}; int s = 0;
    for (int i = 0; i < n; i += 1) {
        if (a[i].size() != in[i]) {
            if (int(a[i].size()) - in[i] == -1) {
                cnt[0] += 1;
            } else if (int(a[i].size()) - in[i] == 1) {
                cnt[1] += 1; s = i;
            } else {
                cout << "No" << endl;
                return;
            }
        }
    }
    for (auto i : {0, 1}) {
        if (cnt[i] > 1) {
            cout << "No" << endl;
            return;
        }
    }
    vector<int> cur(n);
    vector<int> seq;
    auto dfs = [&] (auto&&dfs, int now) -> void {
        // ps;
        if (cur[now] == 0) sort(a[now].begin() , a[now].end());
        for (int &i = cur[now]; i < a[now].size();) {
            dfs(dfs, a[now][i ++]);
        }
        seq.push_back(now);
    };
```

```
    dfs(dfs, s);
    reverse(seq.begin(), seq.end());
    for (auto u : seq) {
        cout << u + 1 << ' ';
    }
    cout << endl;
}

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

    i64 _ = 1 ;
    // cin >> _ ;
    while (_ --) {
        // debug(_);
        solve ();
    }
    return 0 ;
}
```

静态圆方树

```
template <class E>
struct csr {
    vector<int> s;
    vector<E> adj;
    csr() {}
    csr(int n, const vector<pair<int, E>>& g)
        : s(n + 1), adj(g.size()) {
        for (auto e : g) {
            s[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            s[i] += s[i - 1];
        }
        auto c = s;
        for (auto e : g) {
            adj[c[e.first]++] = e.second;
        }
    }
};

// static weighted tree
template<int N = int(1E6)>
struct multiTree {
    int n;
    int K;
    vector<array<int, __lg(N) + 1>> p;
    struct Edge {
        int v; i64 w;
        friend ostream &operator<<(ostream &cout, Edge u) {
```

```
        return cout << "(" << u.v << ", " << u.w << ")";
    }
};
vector<pair<int, Edge>> e;
csr<Edge> g;
vector<int> dep;
vector<i64> dep2;
multiTree() {}
multiTree(int n) {
    init(n);
}
void init(int n) {
    this->n = n;
    K = __lg(n) + 1;
    p.assign(n, {});
    dep.assign(n, 0);
    dep2.assign(n, 0);
}
void addEdge(int u, int v, auto w) {
    e.push_back({u, {v, w}});
    e.push_back({v, {u, w}});
}
void work(int r = 0) {
    g = csr<Edge>(n, e);
    dfs(r, r);
}
void dfs(int u, int f) {
    p[u][0] = f;
    for (int i = 1; i < K; i += 1) {
        p[u][i] = p[p[u][i - 1]][i - 1];
    }
    for (int i = g.s[u]; i < g.s[u + 1]; i += 1) {
        auto [v, w] = g.adj[i];
        if (v == f) {
            continue;
        }
        dep[v] = dep[u] + 1;
        dep2[v] = dep2[u] + w;
        dfs(v, u);
    }
}
int jump(int u, int d) {
    for (int i = dep[u] - d; i > 0; i -= 1 << __lg(i)) {
        u = p[u][__lg(i)];
    }
    return u;
}
int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        swap(u, v);
    }
    u = jump(u, dep[v]);
    if (u == v) {
        return u;
    }
    for (int i = __lg(dep[u]); i >= 0; i -= 1) {
```

```
            if (p[u][i] != p[v][i]) {
                u = p[u][i], v = p[v][i];
            }
        }
        return p[u][0];
    }
    int dist(int u, int v) {
        int m = lca(u, v);
        return dep[u] + dep[v] - 2 * dep[m];
    }
    int dis2(int u, int v) {
        int m = lca(u, v);
        return dep2[u] + dep2[v] - 2 * dep[m];
    }
};

template<class E>
struct star {
    vector<int> s, t;
    star() {}
    star(int n, const vector<pair<int, E>> &e): s(n, -1) {
        // debug(e);
        t.reserve(e.size());
        for (int i = 0; i < e.size(); i += 1) {
            int u = e[i].first;
            t.push_back(s[u]), s[u] = i;
        }
    }
};

struct RST : public multiTree<> {
    using multiTree<>::operator=; // 导入基类的所有成员
    int n, N;
    struct Edge {
        int v; i64 w;
        friend ostream &operator<<(ostream &cout, Edge u) {
            return cout << "(" << u.v << ", " << u.w << ")";
        }
    };
    vector<pair<int, Edge>> e;

    vector<i64> s, cir;
    RST(int n): n(n), N(n), s(n), cir(n) {}
    void tarjan() {
        star<Edge> g(n, e);
        int cur = 0;
        vector<int> dfn(n, -1), low(n, -1), fe(n, -1);
        auto tarjan = [&] (auto &&tarjan, int u, int E) -> void {
            dfn[u] = low[u] = cur ++;
            for (int i = g.s[u]; i != -1; i = g.t[i]) {
                auto [v, w] = e[i].second;
                if (dfn[v] == -1) {
                    fe[v] = i;
                    tarjan(tarjan, v, i);
                    low[u] = min(low[u], low[v]);
                    if (dfn[u] < low[v]) {
```

```
                    addEdge2(u, v, w);
                }
            } else if (i != (E ^ 1)) {
                low[u] = min(low[u], low[v]);
            }
        }
        for (int i = g.s[u]; i != -1; i = g.t[i]) {
            auto [v, w] = e[i].second;
            if (dfn[u] < dfn[v] && fe[v] != i) {
                int t = N ++;
                i64 sum = w;
                for (int j = v; j != u; j = e[fe[j]].first) {
                    s[j] = sum;
                    sum += e[fe[j]].second.w;
                }
                for (int j = v; j != u; j = e[fe[j]].first) {
                    addEdge2(t, j, std::min(s[j], sum - s[j]));
                    cir[j] = sum;
                }
                addEdge2(u, t, 0);
            }
        }
    };
    tarjan(tarjan, 0, 1E9);
    init(N);
    work();
}
void addEdge(int u, int v, i64 w) {
    e.push_back({u, {v, w}});
    e.push_back({v, {u, w}});
}
void addEdge2(int u, int v, i64 w) {
    multiTree::addEdge(u, v, w);
}
i64 dis(int u, int v) {
    int m = lca(u, v);
    if (m < n) {
        return dep2[u] + dep2[v] - 2 * dep2[m];
    }
    int d = dep[m];
    int x = jump(u, d + 1), y = jump(v, d + 1);
    i64 delta = std::abs(s[x] - s[y]);
    return dep2[u] + dep2[v] - dep2[x] - dep2[y] + std::min(delta, cir[x] -
delta);
}
};
```

存图

稀疏行压缩

```
template <class E>
struct csr {
    vector<int> s;
    vector<E> adj;
    csr(int n, const vector<pair<int, E>>& g)
        : s(n + 1), adj(g.size()) {
        for (auto e : g) {
            s[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            s[i] += s[i - 1];
        }
        auto c = s;
        for (auto e : g) {
            adj[c[e.first]++] = e.second;
        }
    }
};
```