图论

SCC

```
struct SCC {
    using i64 = long long ;
    i64 n , cnt = 0 , tot = -1 ;
    vector < vector < i64 > > mp ;
    vector < i64 > d , id , stack , tag ;
    vector < bool > instack ;

    explicit SCC ( ll n , vector < vector < i64 > >& mp ) :
        n ( n ) , mp ( mp ) , d ( vector < i64 > ( n , -1 ) ) , id ( vector < i64 > ( n ) ) ,
            tag ( vector < i64 > ( n , -1 ) ) , instack ( vector < bool > ( n , 0 ) ) {}
private:
    void __scc ( ll now ) {
//      ps ;
        d [ now ] = id [ now ] = ++ tot ;
        stack.push_back ( now ) ; instack [ now ] = 1 ;
        for ( auto u : mp [ now ] ) {
            if ( !~d [ u ] ) {
                __scc ( u ) ;
                id [ now ] = min ( id [ now ] , id [ u ] ) ;
            }
            else if ( instack [ u ] ) {
                id [ now ] = min ( id [ now ] , id [ u ] ) ;
            }
        }
        if ( d [ now ] == id [ now ] ) {
            ++ cnt ;
            do {
                instack [ stack.back () ] = 0 ;
                tag [ stack.back () ] = cnt ;
                stack.pop_back () ;
            } while ( instack [ now ] ) ;
        }
    }
public:
    void scc ( ll now ) {
        -- cnt ;
        __scc ( now ) ;
        ++ cnt ;
    }
};
```

Lca、dfn、虚树

```
template<class T,
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - 1);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};

struct DFN {
    int n;
    vector<int> dfn, dep, sz, fa;
    RMQ<array<int, 2>> rmq;
    DFN() = default;
    template<class T>
    DFN(const std::vector<std::vector<T>> &adj, T root = 0) {
        init(adj, root);
    }
    template<class T>
    void init(const std::vector<std::vector<T>> &adj, T root = 0) {
        n = adj.size();
        dfn.assign(n, 0);
        dep.assign(n, 0);
        sz.assign(n, 0);
        fa.assign(n, 0);
        virtual_tree.assign(n, {});
        vector<array<int, 2>> inrmq(n);
        int tot = 0;
        auto &pa = fa;
        auto dfs = [&] (auto&&dfs, int now, int fa) -> void {
            dfn[now] = tot ++;
            dep[now] = dep[fa] + 1;
            pa[now] = fa;
            for (auto here : adj[now]) {
                if (here == fa) continue;
                dfs(dfs, here, now);
                sz[now] += sz[here];
            }
            sz[now] += 1;
        };
        dfs(dfs, root, root);
        for (int i = 0; i < n; i += 1) {
            inrmq[dfn[i]] = {dep[i], i};
        }
        rmq.init(inrmq);
    }
    int lca (int lhs, int rhs) {
        if (lhs == rhs) return lhs;
        if (dfn[lhs] > dfn[rhs]) swap(lhs, rhs);
        return fa[rmq(dfn[lhs] + 1, dfn[rhs] + 1)[1]];
    }
    std::vector<std::vector<int>> virtual_tree;
    std::vector<int> real_key;
    template<class T>
    std::vector<std::vector<int>> &build_virtual_tree(std::vector<T> key) {
        for (auto u : real_key) {
            virtual_tree[u].clear();
        }
        real_key.clear();
        sort(key.begin(), key.end(), [&] (T x, T y) {return dfn[x] < dfn[y];});
        for (int i = 0; i < int(key.size()) - 1; i += 1) {
            real_key.push_back(key[i]);
            real_key.push_back(lca(key[i], key[i + 1]));
        }
        real_key.push_back(key.back());
        sort(real_key.begin(), real_key.end(), [&] (T x, T y) {return dfn[x] <
dfn[y];});
        real_key.erase(std::unique(real_key.begin(), real_key.end()),
real_key.end());
        for (int i = 0; i < int(real_key.size()) - 1; i += 1 ){
            int Lca = lca(real_key[i], real_key[i + 1]);
            virtual_tree[Lca].push_back(real_key[i + 1]);
            virtual_tree[real_key[i + 1]].push_back(Lca);
        }
        return virtual_tree;
    }
};
```

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);
    }
};
```

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;
    }
};
```

费用流

```
template < typename T >
struct Min_Cost_Flow {
    using i64 = int64_t ;
    struct info { T v , f , c ; info ( T v , T f , T c ): v ( v ) , f ( f ) , c ( c ) {}};

    i64 n ;
    vector < info > e ;
    vector < vector < T > > g ;
    std::vector < i64 > dis ,h ;
    std::vector < T > pre ;

    Min_Cost_Flow ( i64 n ): n ( n ) , g ( n ) {}
    void add ( T u , T v , T f , T c ) {
        if ( c < 0 ) {
            g [ u ].push_back ( e.size () ) ;
            e.emplace_back ( v , 0 , c ) ;
            g [ v ].push_back ( e.size () ) ;
            e.emplace_back ( u , f , -c ) ;
        } else {
            g [ u ].push_back ( e.size () ) ;
            e.emplace_back ( v , f , c ) ;
            g [ v ].push_back ( e.size () ) ;
            e.emplace_back ( u , 0 , -c ) ;
        }
    }
    bool dijkstra ( i64 s , i64 t ) {
        dis.assign ( n , std::numeric_limits < i64 >::max () ) ;
        pre.assign ( n , -1 ) ;
        priority_queue < pair < i64 , i64 > , std::vector < pair < i64 , i64 > >
                , std::greater < pair < i64 , i64 > > > que ;
        dis [ s ] = 0 ;
        que.emplace ( 0 , s ) ;
        while ( !que.empty () ) {
            auto [ d , u ] = que.top () ;
            que.pop () ;
            if ( dis [ u ] < d ) continue ;
            for ( i64 i : g [ u ] ) {
                auto [ v , f , c ] = e [ i ] ;
                if ( f > 0 && dis [ v ] > d + h [ u ] - h [ v ] + c ) {
                    dis [ v ] = d + h [ u ] - h [ v ] + c ;
                    pre [ v ] = i ;
                    que.emplace ( dis [ v ] , v ) ;
                }
            }
        }
        return dis [ t ] != std::numeric_limits < i64 >::max () ;
    }

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

    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].v;
            x.to = e[i].v;
            x.cap = e[i].f + e[i + 1].f;
            x.flow = e[i + 1].f;
            a.push_back(x);
        }
        return a;
    }
};
```