```cpp
#include <bits/stdc++.h>
using namespace std;
// 割边缩点
struct EDCC{
    int n, now, cnt;
    vector<vector<int>> ver;
    vector<int> dfn, low, col, S;
    set<array<int, 2>> bridge;
    EDCC(int n) : n(n), ver(n + 1), low(n + 1){
        dfn.resize(n + 1, -1);
        col.resize(n + 1, -1);
        now = cnt = 0;
    }
    void add(int x, int y){
        ver[x].push_back(y);
        ver[y].push_back(x);
    }
    void tarjan(int x, int fa)
    {
        dfn[x] = low[x] = now++;
        S.push_back(x);
        for (auto y : ver[x]){
            if (y == fa) continue;
            if (dfn[y] == -1){
                bridge.insert({x, y}); // 储存割边
                tarjan(y, x);
                low[x] = min(low[x], low[y]);
            }
            else if (col[y] == -1 && dfn[y] < dfn[x]){
                bridge.insert({x, y});
                low[x] = min(low[x], dfn[y]);
            }
        }
        if (dfn[x] == low[x]){
            int pre;
            cnt++;
            do{
                pre = S.back();
                col[pre] = cnt;
                S.pop_back();
            }while (pre != x);
        }
    }
    auto work(){ // [cnt 新图的顶点数量， bridge 全部割边]
        for (int i = 1; i <= n; i++) { // 避免图不连通
            if (dfn[i] == -1) {
                tarjan(i, 0);
            }
        }
        vector<int> siz(cnt + 1); // siz 每个边双中点的数量
        vector<vector<int>> adj(cnt + 1); // adj 新图
        for (int i = 1; i <= n; i++) {
```

```
            siz[col[i]]++;
            for (auto j : ver[i]) {
                int x = col[i], y = col[j];
                if (x != y) {
                    adj[x].push_back(y);
                }
            }
        }
        return tuple{cnt, adj, col, siz};
    }
};
```

```
#include <bits/stdc++.h>

using namespace std;
// 有向图缩点
struct SCC
{
    int n, now, cnt;
    vector<vector<int>> ver;
    vector<int> dfn, low, col, S;

    SCC(int n) : n(n), ver(n + 1), low(n + 1)
    {
        dfn.resize(n + 1, -1);
        col.resize(n + 1, -1);
        now = cnt = 0;
    }

    void add(int x, int y)
    {
        ver[x].push_back(y);
    }
    void tarjan(int x)
    {
        dfn[x] = low[x] = now++;
        S.push_back(x);
        for (auto y : ver[x])
        {
            if (dfn[y] == -1)
            {
                tarjan(y);
                low[x] = min(low[x], low[y]);
            }
            else if (col[y] == -1)
            {
                low[x] = min(low[x], dfn[y]);
            }
        }
        if (dfn[x] == low[x])
        {
            int pre;
```

```
            cnt++;
            do{
                pre = S.back();
                col[pre] = cnt;
                S.pop_back();
            }while (pre != x);
        }
    }
    tuple<int, vector<vector<int>>, vector<int>, vector<int>> work()
    {
        for (int i = 1; i <= n; i++)
        {
            if (dfn[i] == -1)
            {
                tarjan(i);
            }
        }

        vector<int> siz(cnt + 1);
        vector<vector<int>> adj(cnt + 1);
        for (int i = 1 ; i <= n; i++)
        {
            siz[col[i]]++;
            for (auto j : ver[i])
            {
                int x = col[i], y = col[j];
                if (x != y)
                {
                    adj[x].push_back(y);
                }
            }
        }
        // 新图的顶点数， 新图的边， 节点属于哪个个SCC， SCC的大小
        // C++17及以上可用 : auto {} = scc.work();
        // C++11 / 14 可用: result = scc.work(); vector<int> cnt = get<0>(result);
vector<vector<int>> adj = get<1>(result); vector<int> col = get<2>(result);
vector<int> siz = get<3>(result);
        return {cnt, adj, col, siz};
    }
};
```

```
#include <bits/stdc++.h>
using namespace std;
// 割点缩边
struct V_DCC
{
    int n;
    vector<vector<int>> ver, col;
    vector<int> dfn, low, S;
    int now, cnt;
    vector<bool> point; // 记录是否为割点
    V_DCC(int n) : n(n)
```

```
    {
        ver.resize(n + 1);
        dfn.resize(n + 1);
        low.resize(n + 1);
        col.resize(2 * n + 1);
        point.resize(n + 1);
        S.clear();
        cnt = now = 0;
    }
    void add(int x, int y)
    {
        if (x == y)
            return; // 手动去除重边
        ver[x].push_back(y);
        ver[y].push_back(x);
    }
    void tarjan(int x, int root)
    {
        low[x] = dfn[x] = now++;
        S.push_back(x);
        if (x == root && !ver[x].size())
        { // 特判孤立点
            ++cnt;
            col[cnt].push_back(x);
            return;
        }
        int flag = 0;
        for (auto y : ver[x])
        {
            if (!dfn[y])
            {
                tarjan(y, root);
                low[x] = min(low[x], low[y]);
                if (dfn[x] <= low[y])
                {
                    flag++;
                    if (x != root || flag > 1)
                    {
                        point[x] = true; // 标记为割点
                    }
                    int pre = 0;
                    cnt++;
                    do
                    {
                        pre = S.back();
                        col[cnt].push_back(pre);
                        S.pop_back();
                    } while (pre != y);
                    col[cnt].push_back(x);
                }
            }
            else
            {
                low[x] = min(low[x], dfn[y]);
```

```
            }
        }
    }
    pair<int, vector<vector<int>>> rebuild()
    { // [新图的顶点数量, 新图]
        work();
        vector<vector<int>> adj(cnt + 1);
        for (int i = 1; i <= cnt; i++)
        {
            if (!col[i].size())
            { // 注意，孤立点也是 V-DCC
                continue;
            }
            for (auto j : col[i])
            {
                if (point[j])
                { // 如果 j 是割点
                    adj[i].push_back(point[j]);
                    adj[point[j]].push_back(i);
                }
            }
        }
        return {cnt, adj};
    }
    void work()
    {
        for (int i = 1; i <= n; ++i)
        { // 避免图不连通
            if (!dfn[i])
            {
                tarjan(i, i);
            }
        }
    }
};
```

```
template <class T>
struct Fenwick
{
    int n;
    vector<T> t;
    Fenwick(T n) : n(n) { t.assign(n + 1, T{}); }
    void add(int x, const T &v)
    {
        for (int i = x; i <= n; i += i & -i)
        {
            t[i] += v;
        }
    }
    T sum(int x)
    {
        assert(x >= 0);
```

```
        int res = 0;
        for (int i = x; i; i -= i & -i)
        {
            res += t[i];
        }
        return res;
    }
    T range(int l, int r)
    {
        return sum(r) - sum(l - 1);
    }
    int select(const T &k)
    {
        int x = 0;
        T cur{};
        for (int i = 1 << __lg(n); i; i /= 2)
        {
            if (x + i <= n && cur + t[x + i] <= k)
            {
                x += i;
                cur += t[x];
            }
        }
        return x;
    }
};
```