奇奇怪怪的连通分量

点双联通（无向图）

- 点双联通分量不存在割点
- 任意一对点之间存在两条简单路径，并且经过的点的并集为空集
- 去掉图中的所有割点，可以将图分成各个点双联通分量
- 两两点双联通分量之间的公共点一定是唯一的，一定是割点（两个**相邻的**点双连通分量之间有且仅有一个公共割点）
- 割点连接至少两个点双连通分量
- 任取一个双连通分量为起点，开始走，不经过相同的割点，则最后必然无路可走（即走到一个只有一个割点的双联通分量）
- 连接一个点双连通分量内部两个点的**简单**路径不可能经过这个点双外部的点。（如果经过显然也可以把那个点加入点双）
- **对于点双中的任意一对点，连接它们的简单路径所经过点的并集一定就是这个点双连通分量本身**。（证明不是很懂）

圆方树（无向图）

原理

将原图中的所有双联通分量视为一个个**方点**，将原有的点视为**圆点**，然后把原图的边都删除，将每个圆点与其相关联的双连通分量所代表的方点连边。

显然，一个割点会向多个方点连边，而由此，除了根节点以外，所有的**非割点都是叶子节点**。圆点和圆点之间不会互相连边，方点和方点之间也不会互相连边。

一般情况下，只有两个点的分量也看成点双

性质

- 树上的一条路径一定是圆点方点交替。
- **非割点都是叶子节点**
- 叶子节点都是圆点（原图点数大于1）（存疑）

用途

用于一些简单路径查询相关问题，以及连通性的问题，以及仙人掌图问题。

关键词：无向图、简单路径、不经过相同的点

例题

APOI2018铁人两项

题意：给定一个无向图，问有多少个三元组(x,y,z)□满足从x到z有一条简单路径经过y

题解：双联通建立圆方树，任取两个圆点，答案就是两个圆点路径的点权和,方点点权为双连通分量大小，圆点点权为-1（-1是为了减去任取的两个圆点x,z,同时为了去掉割点的同时计算）。利用双连通分量两个点之间的路径必然经过分量中的所有点

构造完圆方树后，问题转化为，统计树中任意两点路径的权值和，这个可以用换根dp来做，但是很麻烦。考虑每个点权的贡献，其对答案的贡献 = val*经过的路径条数

```
/*
  题意：给定一个无向图，问有多少个三元组(x,y,z)满足从x到z有一条简单路径经过y
  题解：双联通建立圆方树，任取两个圆点，答案就是两个圆点路径的点权和,方点点权为双连通分量大小，圆点点权为-1
   (-1是为了减去任取的两个圆点x,z,同时为了去掉割点的同时计算)
  利用双连通分量两个点之间的路径必然经过分量中的所有点
*/
#include <cstring>
#include <iostream>
#include<cstdio>
#include<algorithm>
#include<vector>
#include<stack>
using namespace std;
typedef long long  ll;
const int MAXN = 1e5 + 20;
int n, m;
vector<int> pic[MAXN], tree[MAXN << 1];
stack<int> stk;
int dfn[MAXN],low[MAXN], max_dfn,size_cc,tot_cc;
int val[MAXN << 1];
ll ans;
int read()
{
    int x=0,flag=1; char c=getchar();
    while((c>'9' || c<'0') && c!='-') c=getchar();
    if(c=='-') flag=0,c=getchar();
    while(c<='9' && c>='0') {x=(x<<3)+(x<<1)+c-'0';c=getchar();}
    return flag?x:-x;
}
void tarjan(int u)//构造圆方树
{
    low[u] = dfn[u] = ++max_dfn;
    stk.push(u);
    ++size_cc;
    for(auto v : pic[u])
    {
        if (!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == dfn[u])
            {
                tot_cc++;//总的双联通分量数
```

```
                val[tot_cc + n] = 0; //方点
                while(!stk.empty())
                {
                    int x = stk.top();
                    stk.pop();
                    tree[tot_cc + n].push_back(x);
                    tree[x].push_back(tot_cc + n);
                    val[tot_cc + n]++;
                    if (x == v)
                        break;
                }
                tree[tot_cc + n].push_back(u);
                tree[u].push_back(tot_cc + n);
                val[tot_cc + n]++;
            }
        }
        else
            low[u] = min(low[u], dfn[v]);
    }
}
/*
   构造完圆方树后，问题转化为，统计树中任意两点路径的权值和，
  这个可以用换根dp来做，但是很麻烦。
   考虑每个点权的贡献，其对答案的贡献 = val*经过的路径条数
*/
int vztd[MAXN << 1], siz[MAXN << 1];
void dfs(int u,int fa)
{
    vztd[u] = 1;
    siz[u] = (u <= n);//统计圆点数量，编号小于n的一定是原点
    for(auto v : tree[u])
        if (v != fa)
        {
            dfs(v, u);
            ans += 2ll * val[u] * siz[u] * siz[v];
            //子树各个点配对，包括了当前节点
            siz[u] += siz[v];
        }
    ans += 2ll * val[u] * siz[u] * (size_cc - siz[u]);
    //祖先节点和子节点配对，包含了祖先节点和当前节点的配对
}
int main()
{
    n = read();
    m = read();
    memset(val, -1, sizeof(val));
    for (int i = 1; i <= m;i++)
    {
        int x = read(), y = read();
        pic[x].push_back(y);
        pic[y].push_back(x);
```

```
    }
    for (int i = 1; i <= n;i++)
        if (!dfn[i])
        {
            while(!stk.empty())
                stk.pop();
            size_cc = 0;
            tarjan(i);
            dfs(i, 0);
        }
    cout << ans << endl;
    return 0;
}
```

CF487E

题意：

给定一个无向图，每个点有点权□1e5个询问，每次查询两点之间简单路径（可能有多条）上的权值最小的点的权值,支持修改点权

题解：

双联通建立圆方树，方点维护双联通分量中点权最小的点，那么在不考虑修改的情况下，原询问等价于静态树上路径最小点权，树剖即可。

考虑暴力修改点权，每次更改更新其周围的方点，但是菊花图会卡。

重点：

将方点维护的权值改为其所有子节点（一定是圆点）的权值最小值，那么更改权值的时候只要修改一次，但是如何保证查询的正确性？

考虑如果依旧按照原本的方式查询路径最小点权，其在哪种情况下不正确，当然是在更新方点最小值的时候，改变割点的权值却只更新了一个方点的数据

如何解决：

如果查询的两个节点的lca是个方点，那么最后答案还要和fa[lca(u,v)]的点权相比较。

为何？

假设lca不是方点，那么对于这条路径上的方点的父亲，也一定在这条路径上，所以即使方点没有被更新好，其父亲一定是更新好的，这样答案是正确的

如果lca是方点，那么它的父亲一定不在路径上，如果父亲的权值更新，这个lca不会被更新，所以要和父亲再比较一番。

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <set>
```

```
const int MN = 100005;
const int MS = 524288;
const int Inf = 0x7fffffff;

int N, M, Q, cnt;
int w[MN * 2];
std::vector<int> G[MN], T[MN * 2];
std::multiset<int> S[MN * 2];

int dfn[MN * 2], low[MN], dfc;
int stk[MN], tp;

void Tarjan(int u) {
    low[u] = dfn[u] = ++dfc;
    stk[++tp] = u;
    for (auto v : G[u]) {
        if (!dfn[v]) {
            Tarjan(v);
            low[u] = std::min(low[u], low[v]);
            if (low[v] == dfn[u]) {
                ++cnt;
                for (int x = 0; x != v; --tp) {
                    x = stk[tp];
                    T[cnt].push_back(x);
                    T[x].push_back(cnt);
                }
                T[cnt].push_back(u);
                T[u].push_back(cnt);
            }
        }
        else low[u] = std::min(low[u], dfn[v]);
    }
}

int idf[MN * 2], faz[MN * 2], siz[MN * 2], dep[MN * 2], son[MN * 2], top[MN
* 2];

void DFS0(int u, int fz) {
    faz[u] = fz, dep[u] = dep[fz] + 1, siz[u] = 1;
    for (auto v : T[u]) if (v != fz) {
        DFS0(v, u);
        siz[u] += siz[v];
        if (siz[son[u]] < siz[v]) son[u] = v;
    }
}

void DFS1(int u, int fz, int tp) {
    dfn[u] = ++dfc, idf[dfc] = u, top[u] = tp;
    if (son[u]) DFS1(son[u], u, tp);
    for (auto v : T[u])
        if (v != fz && v != son[u])
```

```
            DFS1(v, u, v);
}

#define li (i << 1)
#define ri (i << 1 | 1)
#define mid ((l + r) >> 1)
#define ls li, l, mid
#define rs ri, mid + 1, r

int dat[MS];

void Build(int i, int l, int r) {
    if (l == r) { dat[i] = w[idf[l]]; return ; }
    Build(ls), Build(rs);
    dat[i] = std::min(dat[li], dat[ri]);
}

void Mdf(int i, int l, int r, int p, int x) {
    if (l == r) { dat[i] = x; return ; }
    if (p <= mid) Mdf(ls, p, x);
    else Mdf(rs, p, x);
    dat[i] = std::min(dat[li], dat[ri]);
}

int Qur(int i, int l, int r, int a, int b) {
    if (r < a || b < l) return Inf;
    if (a <= l && r <= b) return dat[i];
    return std::min(Qur(ls, a, b), Qur(rs, a, b));
}

int main() {
    scanf("%d%d%d", &N, &M, &Q);
    for (int i = 1; i <= N; ++i)
        scanf("%d", &w[i]);
    cnt = N;
    for (int i = 1; i <= M; ++i) {
        int u, v;
        scanf("%d%d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    Tarjan(1), DFS0(1, 0), dfc = 0, DFS1(1, 0, 1);
    for (int i = 1; i <= N; ++i) if (faz[i])
        S[faz[i]].insert(w[i]);
    for (int i = N + 1; i <= cnt; ++i)
        w[i] = *S[i].begin();
    Build(1, 1, cnt);
    for (int q = 1; q <= Q; ++q) {
        char opt[3]; int x, y;
        scanf("%s%d%d", opt, &x, &y);
        if (*opt == 'C') {
```

```
            Mdf(1, 1, cnt, dfn[x], y);
            if (faz[x]) {
                int u = faz[x];
                S[u].erase(S[u].lower_bound(w[x]));
                S[u].insert(y);
                if (w[u] != *S[u].begin()) {
                    w[u] = *S[u].begin();
                    Mdf(1, 1, cnt, dfn[u], w[u]);
                }
            }
            w[x] = y;
        }
        else {
            int Ans = Inf;
            while (top[x] != top[y]) {
                if (dep[top[x]] < dep[top[y]])
                    std::swap(x, y);
                Ans = std::min(Ans, Qur(1, 1, cnt, dfn[top[x]], dfn[x]));
                x = faz[top[x]];
            }
            if (dfn[x] > dfn[y]) std::swap(x, y);
            Ans = std::min(Ans, Qur(1, 1, cnt, dfn[x], dfn[y]));
            if (x > N) Ans = std::min(Ans, w[faz[x]]);
            printf("%d\n", Ans);
        }
    }
    return 0;
}
```