C++语言: Tarjan Cut Point, Cut Edge, SCC, BBC, EBBC

```
// 如此相像的几份代码就放在一个文件里好了!!!!

// 求割点
template <class T>
inline void tension(T &a, const T &b)
{
    if (b < a) a = b;
}
struct halfEdge
{
    int u;
    halfEdge *next;
};
halfEdge adj_pool[MaxM * 2], *adj_tail = adj_pool;
int n, m;
halfEdge *adj[MaxN + 1];
inline void addEdge(const int &v, const int &u)
{
    adj_tail->u = u, adj_tail->next = adj[v], adj[v] = adj_tail++;
}
inline halfEdge *opposite(const halfEdge *e)
{
    return adj_pool + ((e - adj_pool) ^ 1);
}
int dfn[MaxN + 1], low[MaxN + 1];
int curDfn = 0;
bool isCutPoint[MaxN + 1];
void dfs(const int &v, const halfEdge *preE)
{
    low[v] = dfn[v] = ++curDfn;
    int nSubtree = 0;
    for (halfEdge *i = adj[v]; i; i = i->next)
```

```
            if (i != preE)
            {
                if (!dfn[i->u])
                {
                    dfs(i->u, opposite(i));
                    nSubtree++;
                    if (dfn[v] != 1 && low[i->u] >= dfn[v])
                        isCutPoint[v] = true;
                    tension(low[v], low[i->u]);
                }
                else tension(low[v], dfn[i->u]);
            }
    if (dfn[v] == 1 && nSubtree >= 2) isCutPoint[v] = true;
}
int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        int v, u;
        cin >> v >> u;
        addEdge(v, u);
        addEdge(u, v);
    }
    dfs(1, NULL);
    cout << "cut point:";
    for (int v = 1; v <= n; v++) if (isCutPoint[v]) printf("%d ", v);
    cout << endl;
    return 0;
}

// 求桥
template <class T>
```

```
inline void tension(T &a, const T &b)
{
    if (b < a) a = b;
}
struct halfEdge
{
    int u;
    halfEdge *next;
};
halfEdge adj_pool[MaxM * 2], *adj_tail = adj_pool;
int n, m;
halfEdge *adj[MaxN + 1];
inline void addEdge(const int &v, const int &u)
{
    adj_tail->u = u, adj_tail->next = adj[v], adj[v] = adj_tail++;
}
inline halfEdge *opposite(const halfEdge *e)
{
    return adj_pool + ((e - adj_pool) ^ 1);
}
int dfn[MaxN + 1], low[MaxN + 1];
int curDfn = 0;
void dfs(const int &v, const halfEdge *preE)
{
    low[v] = dfn[v] = ++curDfn;
    for (halfEdge *i = adj[v]; i; i = i->next)
        if (i != preE)
        {
            if (!dfn[i->u])
            {
                dfs(i->u, opposite(i));
                if (low[i->u] > dfn[v]) cout << "(" << v << ", " << i->u << ") ";
                tension(low[v], low[i->u]);
```

```
                }
                else tension(low[v], dfn[i->u]);
            }
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        int v, u;
        cin >> v >> u;
        addEdge(v, u);
        addEdge(u, v);
    }
    cout << "bridge:";
    dfs(1, NULL);
    return 0;
}

//求强联通分量
template <class T>
inline void tension(T &a, const T &b)
{
    if (b < a) a = b;
}
struct halfEdge
{
    int u;
    halfEdge *next;
};
halfEdge adj_pool[MaxM], *adj_tail = adj_pool;
int n, m;
```

```
halfEdge *adj[MaxN + 1];
inline void addEdge(const int &v, const int &u)
{
    adj_tail->u = u, adj_tail->next = adj[v], adj[v] = adj_tail++;
}
inline halfEdge *opposite(const halfEdge *e)
{
    return adj_pool + ((e - adj_pool) ^ 1);
}
int dfn[MaxN + 1], low[MaxN + 1];
int curDfn = 0;
int nScc = 0;
int sccNum[MaxN + 1];
int sta[MaxN + 1], sta_n = 0;
void dfs(const int &v)
{
    low[v] = dfn[v] = ++curDfn;
    sta[sta_n++] = v;
    for (halfEdge *i = adj[v]; i; i = i->next)
    {
        if (!dfn[i->u])
        {
            dfs(i->u);
            tension(low[v], low[i->u]);
        }
        else if (!sccNum[i->u]) tension(low[v], dfn[i->u]);
    }
    if (low[v] == dfn[v])
    {
        nScc++;
        int u;
        do
            u = sta[--sta_n], sccNum[u] = nScc;
```

```
            while (u != v);
        }
    }
    int main()
    {
        cin >> n >> m;
        for (int i = 0; i < m; i++)
        {
            int v, u;
            cin >> v >> u;
            addEdge(v, u);
        }
        for (int v = 1; v <= n; v++) if (!sccNum[v]) dfs(v);
        for (int v = 1; v <= n; v++) cout << v << ":" << sccNum[v] << endl;
        return 0;
    }

    // 求点双连通分量
    template <class T>
    inline void tension(T &a, const T &b)
    {
        if (b < a) a = b;
    }
    struct halfEdge
    {
        int u;
        halfEdge *next;
    };
    halfEdge adj_pool[MaxM * 2], *adj_tail = adj_pool;
    int n, m;
    halfEdge *adj[MaxN + 1];
    inline void addEdge(const int &v, const int &u)
    {
```

```
    adj_tail->u = u, adj_tail->next = adj[v], adj[v] = adj_tail++;
}
inline halfEdge *opposite(const halfEdge *e)
{
    return adj_pool + ((e - adj_pool) ^ 1);
}
inline int edgeIdx(halfEdge *e)
{
    return (e - adj_pool) >> 1;
}
int dfn[MaxN + 1], low[MaxN + 1];
int curDfn = 0;
int sta_n = 0;
int sta[MaxM];
bool isCutPoint[MaxN + 1];
int nBcc = 0;
int bccNum[MaxM];
void dfs(const int &v, const halfEdge *preE)
{
    low[v] = dfn[v] = ++curDfn;
    int nSubtree = 0;
    for (halfEdge *i = adj[v]; i; i = i->next)
        if (i != preE)
        {
            if (!dfn[i->u])
            {
                sta[sta_n++] = edgeIdx(i);
                dfs(i->u, opposite(i));
                nSubtree++;
                if (low[i->u] >= dfn[v])
                {
                    if (dfn[v] != 1) isCutPoint[v] = true;
                    nBcc++;
```

```
                int e;
                do
                    e = sta[--sta_n], bccNum[e] = nBcc;
                while (e != edgeIdx(i));
            }
            tension(low[v], low[i->u]);
        }
        else
        {
            tension(low[v], dfn[i->u]);
            if (dfn[i->u] < dfn[v]) sta[sta_n++] = edgeIdx(i);
        }
    }
    if (dfn[v] == 1 && nSubtree >= 2) isCutPoint[v] = true;
}
int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        int v, u;
        cin >> v >> u;
        addEdge(v, u);
        addEdge(u, v);
    }
    dfs(1, NULL);
    for (int i = 0; i < m; i++) cout << i << ":" << bccNum[i] << endl;
    return 0;
}

// 求边双连通分量
template <class T>
inline void tension(T &a, const T &b)
```

```
{
    if (b < a)
        a = b;
}
struct halfEdge
{
    int u;
    halfEdge *next;
};
halfEdge adj_pool[MaxM * 2], *adj_tail = adj_pool;
int n, m;
halfEdge *adj[MaxN + 1];
inline void addEdge(const int &v, const int &u)
{
    adj_tail->u = u, adj_tail->next = adj[v], adj[v] = adj_tail++;
}
inline halfEdge *opposite(const halfEdge *e)
{
    return adj_pool + ((e - adj_pool) ^ 1);
}
int dfn[MaxN + 1], low[MaxN + 1];
int curDfn = 0;
int sta[MaxN + 1], sta_n = 0;
int nEbcc = 0;
int ebccNum[MaxN + 1];
void dfs(const int &v, const halfEdge *preE)
{
    low[v] = dfn[v] = ++curDfn;
    sta[sta_n++] = v;
    for (halfEdge *i = adj[v]; i; i = i->next)
        if (i != preE)
        {
            if (!dfn[i->u])
```

```
            {
                dfs(i->u, opposite(i));
                tension(low[v], low[i->u]);
            }
            else tension(low[v], dfn[i->u]);
        }
    if (low[v] == dfn[v])
    {
        nEbcc++;
        int u;
        do
            u = sta[--sta_n], ebccNum[u] = nEbcc;
        while (u != v);
    }
}
int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        int v, u;
        cin >> v >> u;
        addEdge(v, u);
        addEdge(u, v);
    }
    dfs(1, NULL);
    for (int v = 1; v <= n; v++) cout << v << ":" << ebccNum[v] << endl;
    return 0;
}
```