环计数问题

环计数问题是一类比较有意思的问题。

无向图三元环计数

先从最简单的三元环考虑起。

最暴力的做法是以每一个点为起点大力枚举，但这样每一个环会被算 6 次，时间复杂度也很高。下面介绍的是一种“优化过的暴力算法”。

首先给点定义大小关系，这里的大小关系定义为二元组 $(\deg_i, i)$ 的大小关系，其中 $\deg_i$ 表示编号为 i 的点的度。即如果 $i < j$ 那么 $(\deg_i, i) < (\deg_j, j)$。

按照这种大小关系可以构造一张 DAG，边 (i, j) 表示给定的无向图中 i, j 相连且按上述定义 $i < j$。然后在这张图上找环。具体的，分为三步：

1. 枚举点 i。
2. 枚举 i 在 DAG 上连接的所有点 j，将其标记上 i。
3. 标记完后枚举 j 连接的所有点 k，如果其已经被标记上 i，那么 i, j, k 构成一个环。

这种做法的原理在于，将无向图上的三元环 A, B, C 转换成为一个 DAG 上的结构 $A \to B, A \to C, B \to C$，且通过定义某种“序”强制规定 A 比 B, C 都要小。这样只能通过枚举 A 统计这个环，且由于定了 B, C 的大小关系使得这个环只能被统计一次。

实际上，这种定序的技巧在离线 LCA 中也有出现。

下面来粗略分析一下这个算法的时间复杂度。整个算法的瓶颈在于第 3 步，耗费的时间成本为 out_j，即 (i, j) 这条边中 j 的出度。我们希望估算出 $\sum_{i=1}^{|E|} \text{out}_j$。下面分情况考虑：

1. 如果 $\text{out}_j \le \sqrt{|E|}$，那么这部分时间复杂度为 $O(|E|^{1.5})$。
2. 如果 $\text{out}_j > \sqrt{|E|}$，那么这样的边数目为 $O(\sqrt{|E|})$ 级别，否则把这些 out_j 加起来会导致边的总数大于 $|E|$。把 out_j 放缩到 $|E|$，表明这部分时间复杂度为 $O(|E|^{1.5})$。

两部分拼起来还是 $O(|E|^{1.5})$，那么就大致分析完了。

以本题为例，给出算法实现：

```cpp
#include <bits/stdc++.h>
#define INF 2000000000
using namespace std;
typedef long long ll;
int read(){
    int f = 1, x = 0;
    char c = getchar();
    while(c < '0' || c > '9'){if(c == '-') f = -f; c = getchar();}
    while(c >= '0' && c <= '9')x = x * 10 + c - '0', c = getchar();
    return f * x;
}
int n, m, du[100005] = {0};
int to[200005], nxt[200005], at[100005] = {0}, cnt = 0;
int to2[200005], nxt2[200005], at2[100005] = {0}, cnt2 = 0;
int mk[100005] = {0};
inline int cmp(int i, int j){
    return (du[i] == du[j] ? i < j: du[i] < du[j]);
```

```
}
void init(){
    n = read(), m = read();
    for (int i = 1; i <= m; ++i){
        int u = read(), v = read();
        to[++cnt] = v, nxt[cnt] = at[u], at[u] = cnt;   // 只保存单向
        ++du[u], ++du[v];
    }
    for (int i = 1; i <= n; ++i)
        for (int j = at[i]; j; j = nxt[j]){
            if (cmp(i, to[j]))
                to2[++cnt2] = to[j], nxt2[cnt2] = at2[i], at2[i] = cnt2;
            else
                to2[++cnt2] = i, nxt2[cnt2] = at2[to[j]], at2[to[j]] = cnt2;
        }
}
void solve(){
    int ans = 0;
    for (int i = 1; i <= n; ++i){
        for (int j = at2[i]; j; j = nxt2[j])
            mk[to2[j]] = i;
        for (int j = at2[i]; j; j = nxt2[j]){
            int v = to2[j];
            for (int t = at2[v]; t; t = nxt2[t]){
                if (mk[to2[t]] == i)
                    ++ans;
            }
        }
    }
    printf("%d\n", ans);
}
int main(){
    init();
    solve();
    return 0;
}
```

需要注意的是，这里连边可以按照 $i < j$ 连也可以按照 $i > j$ 连。有时候如果一种连法超时了不妨尝试另外一种连法，有可能会有奇效。

无向图四元环计数

四元环计数可以套用三元环的思路，即也采用定序的方法。

假设我们希望找的四元环是 A, B, C, D，并且仍然要求这个环只能在对最小的 A 统计时被计算一次。那么在 DAG 上有 $A \to B, A \to C$。然后会发现 D 和 B, C 的大小情况不明确，可以介于两者之间，也可以比两者都大或者小。这意味着应当对所有和 B, C 连接的**无向边**都予以考虑。

由此，我们可以使用和之前类似的方法，只不过这次相当于把路径拆成了 A, B, D 和 A, C, D 进行统计。需要一个辅助数组 $cnt[D]$ 记录以 D 为结尾的这样长度为 2 的路径的数目。

1. 枚举点 i。
2. 枚举 i 在 DAG 上连接的所有点 j，枚举 j 在原图上连接的所有点 k，如果 i 小于 k，那么答案加上 $cnt[k]$，然后令 $cnt[k]$ 自增 1。
3. 对 i 统计完后把 cnt 清空。

可以证明，这一算法的时间复杂度为 $O(|E|^{1.5})$。（具体的证明等待补充）

以本题为例，示例代码如下：（注意这里边是按照 (i,j) 为 $i > j$ 连的，原因见上方）

```cpp
#include <bits/stdc++.h>
#define INF 2000000000
using namespace std;
typedef long long ll;
int read(){
    int f = 1, x = 0;
    char c = getchar();
    while(c < '0' || c > '9'){if(c == '-') f = -f; c = getchar();}
    while(c >= '0' && c <= '9')x = x * 10 + c - '0', c = getchar();
    return f * x;
}
int n, m;
int du[100005] = {0}, rk[100005];
int to[200005], nxt[200005], at[100005] = {0}, cnt = 0;
int to2[400005], nxt2[400005], at2[100005] = {0}, cnt2 = 0;
int pcnt[100005] = {0};
pair<int, int> pp[100005];
void init(){
    n = read(), m = read();
    for (int i = 1; i <= m; ++i){
        int u = read(), v = read();
        ++du[u], ++du[v];
        to2[++cnt2] = v, nxt2[cnt2] = at2[u], at2[u] = cnt2;
        to2[++cnt2] = u, nxt2[cnt2] = at2[v], at2[v] = cnt2;
    }
    for (int i = 1; i <= n; ++i)
        pp[i].first = -du[i], pp[i].second = -i;        // 降序
    sort(pp + 1, pp + n + 1);
    for (int i = 1; i <= n; ++i)
        rk[-pp[i].second] = i;
    for (int i = 1; i <= n; ++i)
        for (int j = at2[i]; j; j = nxt2[j])
            if (rk[i] < rk[to2[j]])
                to[++cnt] = to2[j], nxt[cnt] = at[i], at[i] = cnt;
}
void solve(){
    ll ans = 0;
    for (int i = 1; i <= n; ++i){
        int id = -pp[i].second;
        int v, vv;
        for (int j = at[id]; j; j = nxt[j]){
            v = to[j];
            for (int k = at2[v]; k; k = nxt2[k])
                if (i < rk[to2[k]])
                    ans += pcnt[to2[k]]++;
        }
        for (int j = at[id]; j; j = nxt[j]){
            v = to[j];
            for (int k = at2[v]; k; k = nxt2[k])
                pcnt[to2[k]] = 0;
        }
    }
    printf("%lld\n", 8ll * ans);
```

```
}
int main(){
    init();
    solve();
    return 0;
}
```

有向图环计数

（等待补充）