Course Project Report

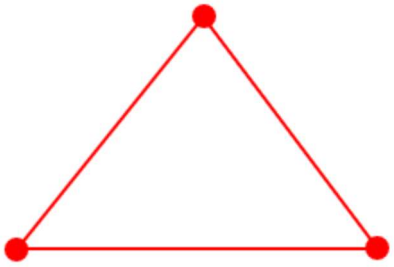
在给定简单无向图中找“三角形”

曹旭 大数据学院 16307110230

1. 题目描述与分析

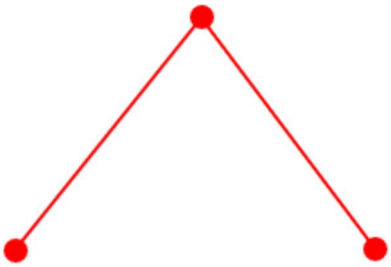
本次题目的任务是在给定简单无向图中找“三角形”，任意三个连通的顶点构成一个“三角形”。这样的“三角形”存在两种情况：

（1）任意两点之间都有边相连



GraphID : 1

（2）三个点由两条边相连，而某两个点之间不存在连边



GraphID : 2

这是一个很有趣的社交网络分析问题，在十万个结点中寻找结点之间的连接模式，应用于分析社交网络中的三元组关系的数目，并可以进一步设计算法对具有类似关系的个体进行推送广告等商业活动。早在上个世纪末，Alon 等人就提出了各种方法，用于在有向和无向图中查找和计算给定长度的简单循环，例如三角形循环、多边形循环等等，他们还发现这些算法复杂度的边界和图的边数相关[1]。传统的算法解决简单无向图中寻找第一种“三角形”的时间复杂度为$O(n^3)$，适当改进后，例如使用稀疏图表示和适当的排序，这一复杂度可以降低。相比第一种“三角形”，在简单无向图中寻找第二种“三角形”就要容易的多，可以利用排列组合公式逐个点计算出这样的三元组的数目并减去 3 倍的第一种“三角形”的数目计算得到，下文中我将列出计算的具体公式。

2. 算法框架描述——基于 forward 算法的改进

Schank 和 Wagner 在 Alon、Batagelj 等人研究的基础上提出了 forward 算法，

forward 算法基于 edge-iterator 思想，它的时间复杂度可以达到$O(m^{\frac{3}{2}})$和 Alon 等人提出

的 listing-ayz 算法相当。forward 算法的伪代码如下[2]：

Algorithm 1: *forward*

Input: ordered list (high degree first) of vertices $(1,\ldots,n)$; Adjacencies $Adj(v)$
Data: Node Data: $A(v)$;

```
for v ∈ V do
    A(v) ← ∅
for s ∈ (1,...,n) do
    for t ∈ Adj(s) do
        if s < t then
            foreach v ∈ A(s) ∩ A(t) do
                output triangle {v, s, t} ;
            A(t) ⟵ A(t) ∪ {s};
```

在 forward 算法中，我们的输入为图的所有顶点$\mathrm{V}(v_1, v_2, \ldots, v_n)$，和每个顶点的邻近顶点的信息，由于是无向无权图，每个顶点和邻近顶点有且只有一条边。

我并没有完全按照 forward 算法来编写程序，因为 forward 算法的目标是输出所有符合条件的三元组且不能重复，而我们的目标是计算第一种“三角形”的数目。我使用了图的邻接表表示来存储顶点和边，并且定义了一个额外的结构体用来存储每个顶点的编号和度。在输入完图的信息后，利用 qsort 排序函数按照度的大小从大到小进行排序（这里其实也可以用一个最大堆来实现，qsort 比较方便，因此我选择了 qsort），然后，按照排序后的顺序从度最大的顶点v_{max}开始，依次遍历。在每次遍历中，从顶点$v_x(\mathrm{x} = 1,2,\ldots,\max,\ldots,\mathrm{n})$的邻接表取出邻近顶点$s_y(s_y \epsilon Adj(v_x))$，首先应用链式存储的 Delete 函数删除$s_y$邻接表中的$v_x$，避免之后的重复情况，然后用 check 函数检测s_y邻接表$\mathrm{Adj}(s_y)$中的每个顶点t_z，check 函数用来判断v_x是否在t_z的邻接表中，一旦 check 函数返回 1，则第一种“三角形”的个数+1。

```c
int check(PtrToAdjVNode L, int n) {
    PtrToAdjVNode P;
    P = L->Next;
    while (P != NULL) {
        if (P->num == n) {
            return 1;
        }
        P = P->Next;
    }
    return 0;
}
```

Check 函数

通过这种方法，我们可以求解出第一种“三角形”的个数。对于第二种“三角形”，可以使用公式直接计算：

之前已经提到，我们在一个额外的结构体存储了每个顶点的编号和度，假设对于顶点$v_x(\mathrm{x} = 1,2,\ldots,\mathrm{n})$，度为$l_x(x = 1,2,..,n)$，则全部 A-B-C 这种顺序连接三元组（先不考虑 A、C 是否相连）的个数 k 为：

$$k = \sum_{x=1}^{n} \binom{l_x}{2} = \sum_{x=1}^{n} \frac{l_x!}{(l_x-2)! * 2!} = \sum_{x=1}^{n} \frac{l_x * (l_x - 1)}{2}$$

在实际代码中，还需要考虑$l_x < 2$的情况，如下：

```
for (i = 1; i <= n; i++) {
        if (count[i].value < 2) {
            triangle2 += 0;
        }
        else if (count[i].value >= 2) {
            triangle2 += count[i].value * (count[i].value-1) /2;
        }
    }
```

由此，我们计算出了全部 A-B-C 这种顺序连接三元组的个数k，但这并不是第二类“三角形”的数目，因为它其中还包括了第一类“三角形”，需要减去这部分数值。

triangle2 = triangle2-3*triangle1

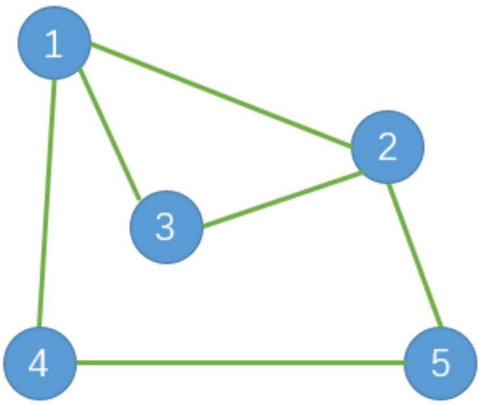
这样，我们就能得到了第一类和第二类“三角形”的数目。

3. 实例分析

以题目给的案例进行具体分析：

5
1 2
1 3
1 4
2 3
2 5
4 5
2 1
5 4
end

在输入案例中还包括了平行边，我在这里使用 check 函数进行判断，假如输入为平行边则说明已经录入了这一数据，通过 if 语句自动跳过。



利用上文的算法，计算出第一类三角形的数目为triangle1 = 1

第二类三角形的数目由下式计算得到：

$$\text{triangle2} = \sum_{x=1}^{n} \frac{l_x * (l_x - 1)}{2} - 3 * \text{triangle1}$$

$$\text{triangle2} \quad = \quad 3+3+1+1+1-3*1=6$$

结果如下：

```
gID:1 freq:1
gID:2 freq:6
```

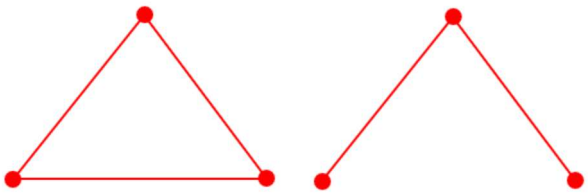
4. 算法复杂度分析

假如用 n 来表示顶点数，m 来表示边数，由于使用了图的邻接表表示，对于每个顶点，都会考虑其邻近顶点，全部顶点总共有 2m 个邻接顶点（边的两端重复考虑）。在 forward 算法中，建邻接表的时间复杂度为$O(n)$，排序的时间复杂度为$O(n\log(n))$。现在通过简易计算寻找这个算法时间复杂度的上界。

平均每个顶点的邻接点个数为$\frac{2m}{n}$

当图为稠密图时，$m=\Omega(n^2)$，$T(n)=O\left(\sum_{x=1}^{n}\left(\left(\frac{2m}{n}-x\right)*\frac{2m}{n}\right)\right)=O\left(\frac{m^2}{n}\right)<O(m^{1.5})$

5. 总结



传统的算法解决简单无向图中寻找第一种“三角形”的时间复杂度为$O(n^3)$，适当改进后，例如使用稀疏图表示和适当的排序，这一复杂度可以降低至$O(m^{1.5})$。相比第一种“三角形”，在简单无向图中寻找第二种“三角形”就要容易的多，可以利用排列组合公式逐个点计算出这样的三元组的数目并减去 3 倍的第一种“三角形”的数目计算得到，具体的实现过程可以参见我写的代码。

参考文献

[1] Alon N , Yuster R , Zwick U . Finding and counting given length cycles[J]. Algorithmica, 1997, 17(3):209-223.

[2] Schank T , Wagner D . Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study[J]. 2005.