# 信息学奥赛中
# 强连通分量与缩点算法的研究

广东实验中学 陈润祥

## 一、研究背景

强连通分量在图论中应用非常广泛。目前，求有向图强连通分量的方法有 3 种——Kosaraju 算法、Tarjan 算法和 Gabow 算法。Kosaraju 算法通过两次搜索，就可以把强连通分量求出来；Tarjan 算法和 Gabow 算法则用栈来记录，只需要一次搜索就能够求出强连通分量。这 3 种算法，对于中学生来说，并不好理解。

中学生的信息学奥赛，考察内容很广泛，既要求强连通分量，对于数据量比较大的图，还需要学生懂得缩点。然而，目前介绍强连通分量的书籍很少，即使有，也只是介绍了算法的流程、实现，很少介绍算法的由来、原理。算法本来就比较难，以至于很多学生很难看懂这个算法，只能强行记住结论，用的时候默写代码。这样并不利于学生的学习。

为了让学生能够更好地学习强连通分量算法，鼓励学生多思考、多钻研，本文将以链表、并查集、深度搜索为基础，研究一个高效的、容易理解的、容易编写代码的、能够缩点的强连通分量算法。

## 二、基本概念

### 1、强连通

在有向图中，如果存在两个顶点 U 和 V 能互相到达，即有一条 U 到 V 的路径，和一条 V 到 U 的路径，则称这两个顶点强连通。显然，图 1 中 U 和 V 能互相到达，U 和 V 强连通。对于图 2，U 和 V 也是强连通，因为存在路径 V→U 和 U→W→V，实现了 U 和 V 互相到达。

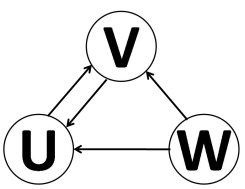


图 1

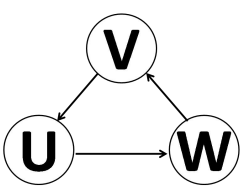


图 2

**2、强连通图**

有向图中，任意两个顶点都强连通，则这个有向图是一个强连通图。如图 2，顶点 U 和 V、U 和 W、V 和 W 分别强连通，因此图 2 是强连通图。

**3、强连通分量**

有向图中的极大强连通子图称为强连通分量。

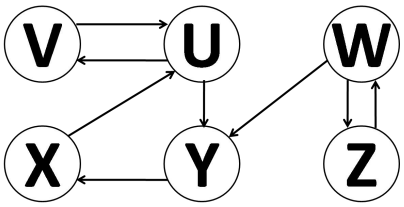


图 3

图的任意一部分都是这个图的子图。图 5、图 6、图 7、图 8 都是图 3 的子图，又因为是强连通图，所以是图 3 的强连通子图。其中，图 6 和图 8 是图 3 的强连通分量，图 5 和图 7 不是，因为图 5 和图 7 合并到一起也是强连通图，即图 5 和图 7 都不是极大强连通子图。

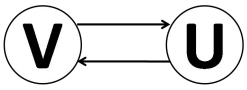


图 5

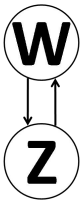


图 6

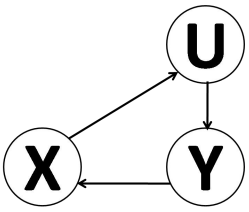


图 7

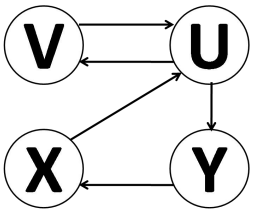


图 8

**4、缩点**

所谓缩点，就是把强连通分量看成一个点。有向图缩点之后，任意两点不会强连通。如果把图 6 看成一个点 P，把图 8 看成一个点 Q，则图 3 缩点后可以转换成图 9.

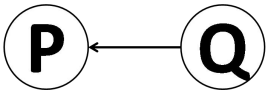


图 9

# 三、算法研究

**1、图的存储**

使用结构体 AtoB 存储有向边，a 记录弧尾，b 记录弧头，n 记录下一条弧尾是 a 的边的编号。h 数组记录头结点的编号。

```
struct AtoB{
      int a, b, n;
} d[maxm];
int h[maxn];
```

**2、强连通分量判断**

1）强连通分量必定有环

强连通分量中的任意两个点，都可以互相到达，也就是存在 U 到 V 的路径和 V 到 U 的路径，即有路径 U→V→U。因此，强连通分量中必定存在环。

2）有环不一定属于强连通分量

在有向图中，并非有环就一定可以互相到达，因为图中的边是有方向的。如图 10，V 可以到达 U 和 W，但 U 和 W 都不能达到 V，因此 V 不可能属于强连通分量中的点。同样地，U 和 W 也不属于强连通分量中的点。

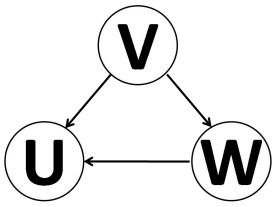


图 10

3）判断方法

使用深度优先搜索，如果下一个可以达到的点没有被访问过，则继续往下搜索；如果访问过，则说明出现环。接下来就需要判断这个环是否属于强连通分量。为了更方便讨论，我们对图中的点进行分类标记。

首先是分类。在搜索过程中，点可以分成 3 类，即没有访问过的点，访问完的点和访问中的点。其中，访问完是指以该结点为根的子树已经全部搜索完毕，访问中的点是指以该结点为根的子树正在被搜索。其次是标记。对于没有访问过的点，标记为 0；访问完的点，标记为-1；访问中的点，标记为 1。

不妨设当前结点是 a 点，下一个要搜索的结点是 b 点。如果 b 点的标记为 0，则表示没访问过，可以继续往下搜索；如果 b 点的标记是 1，则表示正在搜索 b 的子树，由于 b 可以到达 a， a 又可以到达 b，所以 a 点和 b 点强连通，a 和 b 必定属于某个强连通分量；如果 b 点的标记是-1，则 b 点的子树已搜索完毕，b 点不能再达到 a 点，因此 a 点与 b 点非强连通。

```
void dfs(int a){
    int b, i;
    v[a] = 1;//标记访问过
    for(i=h[a]; i!=0; i=d[i].n){//枚举点 a 可以到达的下一个点 b
        b = d[i].b;
        if(v[b] == 0){//没访问过
            dfs(b);
        }
        else if(v[b] == 1){//b 点访问中，与 a 点强连通
            ;//强连通处理
        }
    }
    v[a] = -1;//标记访问完
}
```

**3、强连通分量存储**

由强连通分量的判断方法可知，如果当前结点 a 要访问的下一个结点 b 还在搜索中，那么 a 和 b 属于同一个强连通分量。这时，我们可以使用并查集把他们合并到一起。搜索完毕后，祖先相同的结点，就组成一个强连通分量。祖先结点，也可以看做这个强连通分量缩成的点。

现在要考虑的是，结点 a 的父亲结点是否也属于这个强连通分量。假设 x 是 a 的父亲结点，那么 a 点搜索完后，回到 x 点继续搜索时，a 肯定是搜索完了，标记是-1。如果根据 a 的标记来判断他们是否强连通，那么 a 和 b 肯定不是强连通。但如图 10，若从 U 开始搜索，V 搜索完后标记是-1，但很明显，U 可以到 V

且 V 可以到 U，U 和 V 强连通。因此，我们不应该看 a 的标记，而应该看 a 的祖先的标记。只要 a 的祖先的标记是 1，即 a 的祖先还在搜索中，x 能到达 a 的祖先，a 的祖先也能达到 x，即 a 的祖先和 x 强连通。

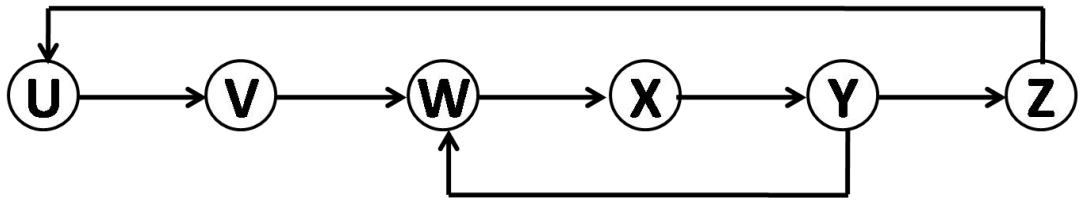


图 10

同时，强连通分量的祖先结点，应选择搜索深度最小的结点，因为深度小的结点，更晚退出搜索。搜索是没有固定顺序的，假设图 10 从 U 点开始搜索，UVWXYZ 的搜索深度分别是 1，2，3，4，5，6。搜索到 Y 点的时候，假设下一个点先搜索 Z，后搜索 W，那么，会先得到 U 和 Z 强连通，需要合并 U 和 Z。显然，为了让 Y 能够加入强连通分量，应选择 U 点作为强连通分量的祖先。同样地，从 Z 回溯到 Y 时，Z 的祖先是 U，即 Y 与 U 强连通，合并时也选择 U 点为祖先。前面两个例子，都是选择弧头的祖先作为强连通分量的祖先的，然而并不全是这样。从 Y 搜索到 W 时，Y 和 W 强连通，如果选择弧头节点作为祖先，即选择 W 结点，那么回溯到 V 时，WXYZU 的祖先都是 W，而 W 的标记是-1，V 结点就不能合并到强连通分量了。因此，在选择强连通分量的祖先时，应选择深度小的，这样才能保证强连通分量的完整。

为了方便，我们可以用标记的数组顺便记录结点的深度，深度为 0 表示没访问过，深度为正数表示搜索中，深度为负数表示搜索完。对之前的代码稍作修改，可以得到以下代码，其中 f 数组记录的是结点的祖先，find 函数的功能是查找并更新节点的祖先。

```
void dfs(int a){//根结点搜索前标记深度为 1
    int i, b;
    for(i=h[a]; i; i=d[i].n){
        b = d[i].y;
        if(!v[b]){
            v[b] = v[a] + 1; //下一个结点的深度比前一个大 1
            dfs(b);
            if(v[find(b)] > 0){ //搜索中的结点与当前结点强连通
                if(v[f[b]] < v[find(a)]) f[f[a]] = f[b];
                else f[f[b]] = f[a]; //取深度小的结点为祖先结点
            }
        }
        else if(v[find(b)] > 0){//搜索中的结点与当前结点强连通
            if(v[f[b]] < v[find(a)]) f[f[a]] = f[b];
            else f[f[b]] = f[a]; //取深度小的结点为祖先结点
        }
    }
    v[a] = -v[a];
}
```

# 四、算法分析

**1、时间复杂度**

对于 n 个节点、m 条边的有向图，在搜索过程中，没访问过的点才会被搜索，因此，每个结点只会访问一次，每一条边也只会访问一次。又由于并查集的复杂度是常数级别的，所以整个算法的复杂度是 O(m+n)。

**2、空间复杂度**

存储每个结点的信息，复杂度是 O(n)，存储每条边的信息，复杂度是 O(m)，整体复杂度是 O(m+n)。

# 五、算法应用

## 1、强连通分量个数

```
for(ans=0, i=1; i<=n; i++){
    if(i == find(i)) ans++;
}
printf("%d\n", ans);
```

## 2、输出强连通分量

```
memset(h, 0, sizeof(h));
for(i=1; i<=n; i++){
    f[i] = find(i);
    d[i].a = f[i], d[i].b = i;
    d[i].n = h[f[i]], h[f[i]] = i;
}
for(i=1; i<=n; i++){
    if(h[i]){
        for(j=h[i]; j; j=d[j].n) printf("%d ", d[j].b);
        printf("\n");
    }
}
```

## 3、缩点

```
memset(h, 0, sizeof(h));
for(i=1; i<=m; i++){
    a = find(d[i].a), b = find(d[i].b);
    if(a != b){
        d[i].a = a, d[i].b = b;
        d[i].n = h[a], h[a] = i;
    }
}
```