加强有向图算法正确性的数学证明

马江岩　邢雨菡

2021 年 1 月 16 日

关于我们程序的闪光点、算法原理、思路和遇到的困难，在我们的汇报中已经做了详细的介绍，在此不再赘述. 因此，本篇实验报告，将主要给出我们程序中所用算法的正确性的数学证明.

我们的程序主要分三个步骤：利用参考文献 [1] 中的 Tarjan 算法找到有向图中的所有强连通分量；利用参考文献 [2] 中提出、并由参考文献 [3] 改正的 STCorrect 算法找到有向无环图中符合某些特定条件的一组点集 v 和 w；利用参考文献 [3] 中提出、并由我们整理改正的加边方法将有向无环图加强为强连通图.

在介绍我们程序算法的数学证明之前，我们先给出一些基本的定义.

考虑一个有向图 G，设它的顶点集为 $\mathscr{V}$、所有有向边构成的集合为 $\mathscr{E}$. 我们用 (v,w) 表示从顶点 v 指向顶点 w 的有向边. 倘若存在一系列有向边 (v,u_1)，(u_1,u_2)，$\cdots$，(u_n,w)，则称 v 连通到 w. 一个有向图称为**强连通的**，如果该有向图的任意两个顶点都是互相连通的.

一个有向图 G 的子图 T 称为**树**，如果该子图有且仅有一个顶点入度为 0（这一点称做**根**），其余顶点入度均为 1. 在一个树中，若存在有向边 (v,w)，则称 v 是 w 的**父节点**、w 是 v 的**子节点**. 若 v 连通到 w，则称 v 为 w 的**祖先**、w 为 v 的**后代**. 每个顶点都是它自己的祖先和后代. 设 v 是树 T 中的一个顶点，则 T_v 是 T 的一个子树，满足 T_v 中的所有顶点都是 v 的后代. 一个树 T 称为有向图 G 的**生成树**，如果 T 包含 G 的所有顶点.

设 $G=(\mathscr{V},\mathscr{E})$ 是一个有向图. 对于每一个顶点 $v\in\mathscr{V}$，我们构造一个包含所有顶点 w 的表，使得 $(v,w)\in\mathscr{E}$. 这个表称为顶点 v 的**邻接表**. 由 G 中所有顶点的邻接表构成的集合，称为有向图 G 的邻接表.

我们将从有向图的某个顶点开始进行的一次深度优先搜索（depth-first search），称为**一次搜索**.

1　Tarjan 算法

我们程序的第一步，是将输入的有向图转化为有向无环图. 这里先给出几个定义.

定义 1. 设 $G=(\mathscr{V},\mathscr{E})$ 是一个有向图. 有向图 G 的一个子图 $G_i=(\mathscr{V}_i,\mathscr{E}_i)$ 称为 G 的强连通子图（strongly connected subgraph），如果 $\mathscr{E}_i=\{(v,w)\in\mathscr{E}|v,w\in\mathscr{V}_i\}$，并且 G_i 是强连通的.

定义 2. 设 $G=(\mathscr{V},\mathscr{E})$ 是一个有向图. 有向图 G 的一个子图 $G_i=(\mathscr{V}_i,\mathscr{E}_i)$ 称为 G 的强连通分量（strongly connected component），如果 G_i 是 G 的一个强连通子图，并且 G_i 是不是任何 G 的异于 G_i 的强连通子图的子图.

定义 3. 一个有向图如果不含任何顶点数大于 1 的强连通分量，则称其为有向无环图.

为了将一个有向图 $G=(\mathscr{V},\mathscr{E})$ 压缩为有向无环图，我们的思路是先找到该有向图的所有强连通分量 G_i，$i=1,2,\cdots,n$，再将这些强连通分量压缩成一个个顶点. 压缩后的有向无环图中存在由一个顶点 G_i 指向另一个顶点 G_j 的边，当且仅当存在 $v\in G_i$，$w\in G_j$，$(v,w)\in\mathscr{E}$.

因此，我们的主要问题便是如何找到一个有向图的所有强连通分量.

关于此问题，一个易于理解的算法是 Kosaraju 算法 [4]：首先对有向图进行深度优先搜索，按每个顶点被搜索到的次序为其标记上时间戳；然后将有向图中的每条有向边倒置（也就是说，将边 (v,w) 倒置为边 (w,v)），得到有向图的逆图；最后按时间戳由大到小的顺序对逆图再进行深度优先搜索，每次搜索中能够到达的点便构成一个强连通分量. 这种方法虽易于理解，也具有 $O(V+E)$ 的时间复杂度，但因为要进行两次深度优先搜索，效率较低. 那么，能否仅通过一次深度优先搜索，便找到有向图中的所有强连通分量呢？

我们程序所采用的 Tarjan 算法可以达到这一目的. 可以认为，Tarjan 算法是对 Kosaraju 算法的优化，它通过在搜索过程中不断计算每个顶点所能连接到的栈中最早搜索到的点的时间戳，同时完成了 Kosaraju 算法两次深度优先搜索完成的任务. 下面，我们先给出一些定义，然后介绍 Tarjan 算法的原理和数学证明.

在深度优先搜索中，每搜索到一个未被标记过的顶点，我们便将其置于栈中，并将其标记. 对于有向图中的顶点 v，我们定义 $\mathbf{number}(v)$ 为 v 在深度优先搜索中被搜索到的次序（即时间戳），从 1 开始编号；$\mathbf{lowlink}(v)$ 为 v 所能够连通到的在栈中的时间戳最小的顶点的次序.

显然，每一次搜索，沿其路径都会得到一个树. 对一个有向图的深度优先搜索可能包括若干次搜索，每一次搜索都得到一个树，这些互不相交的树称为有向图的一个生成森林，它包含有向图的所有顶点.

我们有如下定理.

定理 1. 令 v 和 w 是有向图 G 中位于同一个强连通分量中的顶点. 令 F 是有向图 G 通过深度优先搜索得到的生成森林. 则 v 和 w 在 F 中具有共同的祖先. 进一步，设 u 是 v 和 w 的次序最大的共同祖先，则 u 与 v 和 w 在同一个强连通分量中.

证明. 不妨设 $\mathbf{number}(v)\leqslant\mathbf{number}(w)$. 设 p 是 G 中从 v 连通到 w 的一条路径. 令 T_u 为以 u 为根的、包含 p 中所有顶点的 F 中某个树的最小子树. 我们断言：这样的树 T_u 必然存

在，因为 p 只能连通到 F 中次序较小的树，无法连通到 F 中次序较大的树（否则这个树在之前的搜索中就应当沿路径 p 被搜索到）. 又因为 $\mathbf{number}(v) \leqslant \mathbf{number}(w)$，故 p 不可能属于 2 个或更多个不同的树中，从而 T_u 存在.

下面我们证明：p 必然经过顶点 u. 设 T_{u_1} 和 T_{u_2} 是包含 p 中所有顶点的两个树，且满足存在从 u 到 u_i 的路径，$i = 1, 2$. 若找不到这样的两个树，那么 p 必然经过 u，否则存在 T_{u_1} 包含 p 的所有顶点、p 经过 u_1 且 u 连通到 u_1，T_{u_1} 是比 T_u 更小的树，这与我们先前假设的 T_u 的最小性矛盾. 若能找到这样的两个树，注意到，由于 T_{u_1} 与 T_{u_2} 互不相交，故 p 欲从 T_{u_1} 连通到 T_{u_2}，必然要经过 u. 综上，p 经过顶点 u，从而 u 与 v 和 w 在同一个强连通分量中. □

上述定理表明：若 C 是有向图 G 的一个强连通分量，F 是 G 通过深度优先搜索得到的生成森林，则 C 的所有顶点必然构成 F 中某个树的子树的顶点集. 我们将这个子树的根，称为强连通分量 C 的根.

注意到，一个顶点 v 与栈中更早的某个点 w 相连通有三种方式：$v = w$；v 直接与 w 连通；v 通过不在栈中的若干点间接与 w 连通. 故 $\mathbf{lowlink}(v)$ 应当等于下列三者的最小值：$\mathbf{number}(v)$，$\mathbf{number}(w)$（w 在栈中），$\mathbf{lowlink}(w)$（w 不在栈中），其中 w 是 v 的子节点. 下面的定理给出了寻找强连通分量的方式.

定理 2. 设有向图 G 通过深度优先搜索得到生成森林 F. 则顶点 v 是某个强连通分量的根，当且仅当 $\mathbf{lowlink}(v) = \mathbf{number}(v)$.

证明. 一方面，若 v 是强连通分量 C 的根，必有 $\mathbf{lowlink}(v) = \mathbf{number}(v)$，因为若 $\mathbf{lowlink}(v) < \mathbf{number}(v)$，则存在 v 的祖先 u 也在 C 中，从而 v 不是 C 的根，矛盾.

另一方面，设强连通分量 C 的根为 u，且 v 是 C 中异于 u 的顶点. 则存在从 v 连通到 u 的路径 p. 设 w 是 p 所经过的第一个不属于 T_v 的顶点，由于一条路径不可能从次序低的树连通到次序高的树，故 $\mathbf{lowlink}(v) \leqslant \mathbf{number}(w) < \mathbf{number}(v)$.

综上，定理成立. □

上述定理告诉我们，只要在搜索过程中，不断计算顶点的 **lowlink**，则当计算出顶点 v 满足 $\mathbf{lowlink}(v) = \mathbf{number}(v)$ 时，栈中所有晚于 v 的顶点（含 v）构成强连通分量. 下面的 Tarjan 算法运用这一原理，可以找出有向图的所有强连通分量.

Algorithm 1 Tarjan's Algorithm

```
integer i
procedure TARJAN(v)
    lowlink(v) ← number(v) ← i ← i + 1
    put v on stack of points
    for w in the adjacency list of v do
```

```
        if w is not yet numbered then                          ▷ (v, w) is a tree arc
            TARJAN(w)
            lowlink(v) ← min(lowlink(v), lowlink(w))
        else if number(w) < number(v) then                     ▷ (v, w) is a frond or cross-link
            if w is on stack of points then
                lowlink(v) ← min(lowlink(v), number(w))
            end if
        end if
    end for
    if lowlink(v) = number(v) then                             ▷ v is the root of a component
        start new strongly connected component
        while w is on top of point stack satisfies number(w) ⩾ number(v) do
            delete w from point stack and put w in current component
        end while
    end if
end procedure
i ← 0
empty stack of points
for w is a vertex do
    if w is not yet numbered then
        TARJAN(w)
    end if
end for
```

不难看出，算法 1 计算出的顶点的 **lowlink** 是正确的. 下面我们证明算法 1 正确地找到了有向图的所有强连通分量.

定理 3. 对于任意给定的有向图 G，算法 1 正确地找出了 G 的所有强连通分量.

证明. 设 F 是 G 通过深度优先搜索得到的生成森林. 对于任意顶点 v，设 T_v 是以 v 为根的 F 中某个树的子树. 根据算法的结构，在计算出 **lowlink**(v) 之前，T_v 中的所有顶点已经被置于栈中.

当计算出 **lowlink**(v) 时，若 v 不是某个强连通分量的根，则不会有顶点出栈. 若 v 是某个强连通分量 C 的根，则由定理 1 ，此时栈中的所有顶点均为 C 中的点；且如前所述，C 中所有的顶点均在栈中. 此时算法 1 将栈中所有顶点出栈并存储在一个强连通分量中，故算法 1 找到的强连通分量是正确的. □

不难看出，算法 1 的时间复杂度为 $O(V+E)$.

2 STCorrect 算法

运用 Tarjan 算法，我们可以将有向图压缩成有向无环图. 从而我们的添边操作，都可以在压缩后的有向无环图中进行. 这里，我们需要证明：在原有向图中的添边操作和在压缩后的有向无环图中的添边操作是等价的.

令 $G=(\mathscr{V},\mathscr{E})$ 为一个有向图，$G'=(\mathscr{V}',\mathscr{E}')$ 为其压缩而成的有向无环图. 对 $v\in\mathscr{V}$，令 $\alpha(v)$ 为其在有向无环图中所对应的压缩后的点. 对 $x\in\mathscr{V}'$，令 $\beta(x)$ 为其在原有向图中所对应的强连通分量中的任意点. 显然，对任意 $x\in\mathscr{V}'$，$\alpha(\beta(x))=x$. 我们有如下定理.

定理 4. 设 A 为若干有向边的集合，使得 G 添加 A 后强连通. 则

$$\alpha(A)=\big\{\big(\alpha(v),\alpha(w)\big)|(v,w)\in A,\alpha(v)\neq\alpha(w))\big\}$$

使得 G' 强连通. 反之，设 B 为若干有向边的集合，使得 G' 添加 B 后强连通. 则

$$\beta(B)=\big\{\big(\beta(x),\beta(y)\big)|(x,y)\in B\big\}$$

使得 G 强连通.

证明. 由有向图的压缩规则我们知道，有向无环图中存在边 (G_i,G_j)，当且仅当存在有向图中的边 (v,w)，其中 $v\in G_i$，$w\in G_j$.

设 A 使得 G 强连通，则 $G\cup A$ 中任意两个属于不同强连通分量的顶点互相连通. 故在 $G'\cup\alpha(A)$ 中，任意两个顶点互相连通.

设 B 使得 G' 强连通，则 $G'\cup B$ 中任意两个顶点互相连通. 故在 $G\cup\beta(B)$ 中，任意两个属于不同强连通分量的顶点互相连通. 而 G 中属于同一个强连通分量的两个顶点显然互相连通，故在 $G\cup\beta(B)$ 中，任意两个顶点互相连通. □

在压缩后的有向无环图中，我们称入度为 0、出度不为 0 的顶点为源，称出度为 0、入度不为 0 的顶点为汇，称入度和出度均为 0 的顶点为 孤立节点. 设源的个数为 s、汇的个数为 t、孤立节点的个数为 q. 我们希望找到所有源的一个排列 $v(1),v(2),\cdots,v(s)$ 和所有汇的一个排列 $w(1),w(2),\cdots,w(t)$，以及一个数 p，满足：

1. 存在从 $v(i)$ 到 $w(i)$ 的路径，$1\leqslant i\leqslant p$;
2. 对每个 $v(i)$，$p+1\leqslant i\leqslant s$，存在其到某个 $w(j)$ 的路径，$1\leqslant j\leqslant p$;
3. 对每个 $w(j)$，$p+1\leqslant j\leqslant t$，存在某个 $v(i)$ 到其的路径，$1\leqslant i\leqslant p$.

并记所有的孤立节点为 $x(1), x(2), \cdots, x(q)$.

下面的 STCorrect 算法可以找到这样的排列.

Algorithm 2 STCorrect

```
procedure SEARCH(x)
    if x is unmarked then
        if x is a sink then
            w ← x
            sinknotfound ← false
        end if
        mark x
        for each y such that (x, y) is an edge do
            if sinknotfound then
                SEARCH(y)
            end if
        end for
    end if
end procedure
initialise all vertices to be unmarked
i ← 0
while some source is unmarked do
    choose some unmarked source v
    w ← 0
    sinknotfound ← true
    SEARCH(v)
    if w ≠ 0 then
        i ← i + 1
        v(i) ← v
        w(i) ← w
    end if
end while
p ← i
```

我们来证明算法 2 的正确性.

定理 5. 对于任意给定的有向无环图 G'，算法 2 正确地找到一组满足如前所述条件的源和汇，

以及一个数 p.

证明. 注意到，每一次 Search 都从一个未被标记的源开始，并且每一次 Search 中都不会经过另一个源. 因此，有向无环图中的每一个源恰作为一次 Search 的起点.

若从某个源出发，搜索到了某个汇，则这组源和汇被记录到数组 v 和 w 中，且它们显然满足条件 1.

对每个不在 $v(1), v(2), \cdots, v(p)$ 中的源，由于有向无环图中无环，故该源必连通到某个汇. 若这个汇不在 $w(1), w(2), \cdots, w(p)$ 中，那么，在以该源为起点的 Search 中，就应当可以找到这个汇，从而它们应当被记录到数组 v 和 w 中，矛盾. 故条件 2 成立.

同理，条件 3 成立. 故算法 2 正确地找到了满足如前所述条件的源和汇，以及一个数 p. □

不难看出，算法 2 的时间复杂度也为 $O(V + E)$.

可以这样理解 STCorrect 算法：该算法的目的是找到尽可能大的一个环，使其包括有向无环图中所有的孤立节点、尽可能多的源和尽可能多的汇. 那么接下来的添边操作就只需完成两个任务：一是构建这个环，二是将不在数组 v 和 w 中的源和汇与这个环相连. 这些添边操作只在源、汇和孤立节点间进行，对于既不是源和汇、也不是孤立节点的顶点，由于其必被某个源所连通并且必连通到某个汇，因此必与这个环互相连通. 这样的点，我们添边时便不用再去管它们了.

3 DFS 的非递归实现

在介绍我们的添边方法之前，首先我们说明如何将基于深度优先搜索的递归算法优化为非递归版本. 注意到，在算法 1 和算法 2 的伪代码中，均用到了递归调用；但在我们程序的实际代码中，却只有一个 main 函数，没有用到递归函数. 这是因为我们将所有递归函数都用非递归的方式实现了出来. 这样做有两个好处：一是减少函数调用的开销，提升程序效率；二是不用担心“爆栈”的问题.

将深度优先搜索非递归实现的基本思路是用一个数组 stack 去模拟调用栈，并通过一定的算法去模拟函数递归调用的过程. 一种可行的算法如下.

Algorithm 3 Non-recursive DFS

```
initialise stack
mark starting point x
push x in the stack
while the stack is not empty do
    let v be the vertex on top of stack
    if there is a unmarked vertex w in the adjacency list of v then
```

```
        mark w
        push w in the stack
    else
        pop v from the stack
    end if
end while
```

该算法完成了深度优先搜索的过程，并且没有用到递归调用. 下面我们以 Tarjan 算法为例，给出非递归版本的 Tarjan 算法的伪代码.

Algorithm 4 Non-recursive Tarjan's Algorithm

```
integer i
procedure TARJAN(k)
    push k in vstack
    while vstack is not empty do
        let v be the vertex on top of vstack
        pop v from vstack
        if v is not yet numbered then
            number(v) ← lowlink(v) ← i ← i + 1
            push v in stack
        end if
        jexists ← false
        for w in the adjacency list of v do
            if w is not yet numbered then
                push v in vstack
                push w in vstack
                jexists ← true
            else if number(w) < number(v) then
                if w is on stack of points then
                    lowlink(v) ← min (lowlink(v), lowlink(w))
                end if
            end if
        end for
        if jexists = false then
            if number(v) = lowlink(v) then
                start new strongly connected component
```

```
            while w on top of stack satisfies number(w) ⩾ number(v) do
                pop w from stack and put w in current component
            end while
        end if
        if vstack is not empty then
            w ← v
            let v be the vertex on top of vstack
            lowlink(v) ← min (lowlink(v), lowlink(w))
        end if
    end if
  end while
end procedure
i ← 0
empty stack of points
for w is a vertex do
  if w is not yet numbered then
    TARJAN(w)
  end if
end for
```

类似于算法 4 ，我们可以将 STCorrect 算法也改写为非递归版.

4 添边使有向图强连通

在第 2 节中，我们已经找到了符合三个条件的一组源和汇，以及一个数 p. 在此基础上，我们希望添加尽可能少的有向边，使有向无环图强连通. 下面的定理给出了添加有向边数量的一个下界.

定理 6. 设 G' 为一个给定的有向无环图，包含 s 个源、t 个汇、q 个孤立节点，且 $s+t+q>1$. 则至少需要添加 $\max(s,t)+q$ 条有向边，才能使 G' 强连通.

证明. 在添加若干条有向边使得 G' 强连通后，对原来的有向无环图中的任意源和孤立节点，都必然存在一条有向边指向它，故至少需要添加 $s+q$ 条有向边. 类似地，对原来的有向无环图中的任意汇和孤立节点，都必然存在一条以其为起点的有向边，故至少需要添加 $t+q$ 条有向边. 综上，至少需要添加 $\max(s,t)+q$ 条有向边，才能使 G' 强连通. □

那么，这一下界是否是充分的呢？答案是肯定的. 参考文献 [3] 中给出了如下的添边方式. 设添加的有向边构成的集合为 A^{ASC}. 不失一般性，不妨假设 $s \leqslant t$. 这一假设是合理的，因为倘若 $s > t$，则在该有向无环图 G' 的逆图中，有 $s < t$. 我们利用 $s \leqslant t$ 时的加边方法，将 G' 的逆图加强为强连通图，再将所有添加的有向边倒置，即得到了使得原有向无环图强连通的有向边的集合 A^{ASC}.

参考文献 [3] 中的添边方法如下.

$$
\begin{aligned}
A^{\mathrm{ASC}} = {} & \big\{\big(w(i), v(i+1)\big) | 1 \leqslant i < p\big\} \cup \big\{\big(w(i), v(i)\big) | p+1 \leqslant i \leqslant s\big\} \\
& \cup \begin{cases}
\big(w(p), v(1)\big), & \text{若 } q = 0 \text{ 且 } s = t, \\
\big(w(p), w(s+1)\big) \cup \big\{\big(w(i), w(i+1)\big) | s+1 \leqslant i < t\big\} & \\
\cup \big(w(t), v(1)\big), & \text{若 } q = 0 \text{ 且 } s < t, \\
\big(w(p), w(s+1)\big) \cup \big\{\big(w(i), w(i+1)\big) | s+1 \leqslant i < t\big\} & \\
\cup \big(w(t), x(1)\big) \cup \big\{\big(x(i), x(i+1)\big) | 1 \leqslant i < q\big\} & \\
\cup \big(x(q), v(1)\big), & \text{否则.}
\end{cases}
\end{aligned}
$$

我们指出，上面的加边方法存在 3 处错误：

1. 当 $p = 0$ 时，此时必有 $s = t = 0$. 按照上面的加边方法，程序会加上 $\big(w(p), v(1)\big)$ 这条边. 可是 $w(0)$ 是没有定义的.

2. 当 $s = t$ 且 $q > 0$ 时，按照上面的加边方法，程序会加上 $\big(w(p), w(s+1)\big)$ 这条边. 可是 $w(s+1)$ 是没有定义的.

3. 当 $s = t$ 且 $q > 0$ 时，按照上面的加边方法，程序会加上 $\big(w(t), x(1)\big)$ 这条边. 这是不正确的，有可能导致加强后的有向图不是强连通的. 正确的加边方法应当是加上 $\big(w(p), x(1)\big)$ 这条边.

可以认为，前 2 处错误是无关痛痒的小错误，读者很容易自行发现并改正. 但第 3 处错误则是较严重的错误，它导致按照上述加边方式加强得到的有向图可能不是强连通的. 事实上，我们可以构造如下的反例.

如图 1 的左图，该图是一个有向无环图，含有 2 个源 a、d，2 个汇 b、c，以及一个孤立节点 e. 我们假设程序在运行 STCorrect 算法时，先以点 a 为起点进行 Search，并且在 Search 的过程中先经过有向边 (a, b). 沿路径 (a, b) 算法会找到一个汇 b，进而这次搜索会记录 $v(1) = a$，$w(1) = b$. 之后程序以 d 为起点进行 Search，无法找到汇. 最终 STCorrect 算法运行结束，程序记录 $v(1) = a$，$v(2) = d$，$w(1) = b$，$w(2) = c$，$x(1) = e$，$p = 1$.

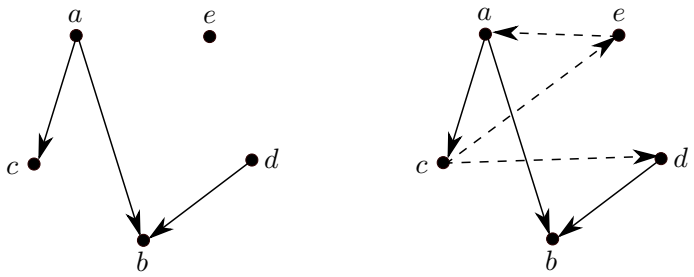

图 1: [3] 中加边方法的反例

随后我们按照 [3] 中给出的加边方法进行添边，则添加的边有：$\big(w(2),v(2)\big)$，$\big(w(2),x(1)\big)$，$\big(x(1),v(1)\big)$. 加边后的有向图如图 1 右图所示. 可以看出，它不是强连通的. 如前所述，此处正确的加边方法应是加 $\big(w(p),x(1)\big)$ 这条边.

为此，我们重新讨论了所有可能的情况，整理出了正确的加边方法，如算法 5 所示.

Algorithm 5 Adding Edges

if $s=t$ **then**
  if $p=0,q>1$ **then**
    Add $\big(x(q),x(1)\big)$
    for $1\leqslant i<q$ **do**
      Add $\big(x(i),x(i+1)\big)$
    end for
  else if $p>0$ **then**
    for $1\leqslant i<p$ **do**
      Add $\big(w(i),v(i+1)\big)$
    end for
    for $p+1\leqslant i\leqslant s$ **do**
      Add $\big(w(i),v(i)\big)$
    end for
    if $q=0$ **then**
      Add $\big(w(p),v(1)\big)$
    else if $q>0$ **then**
      Add $\big(w(p),x(1)\big)$
      for $1\leqslant i<q$ **do**
        Add $\big(x(i),x(i+1)\big)$

```
            end for
            Add (x(q), v(1))
        end if
    end if
else if s < t then
    for 1 ⩽ i < p do
        Add (w(i), v(i + 1))
    end for
    for p + 1 ⩽ i ⩽ s do
        Add (w(i), v(i))
    end for
    Add (w(p), w(s + 1))
    for s + 1 ⩽ i < t do
        Add (w(i), w(i + 1))
    end for
    if q = 0 then
        Add (w(t), v(1))
    else if q > 0 then
        Add (w(t), x(1))
        for 1 ⩽ i < q do
            Add (x(i), x(i + 1))
        end for
        Add (x(q), v(1))
    end if
else if s > t then
    for 1 ⩽ i < p do
        Add (w(i + 1), v(i))
    end for
    for p + 1 ⩽ i ⩽ t do
        Add (w(i), v(i))
    end for
    Add (v(t + 1), v(p))
    for t + 1 ⩽ i < s do
        Add (v(i + 1), v(i))
```

```
    end for
    if q = 0 then
        Add (w(1), v(s))
    else if q > 0 then
        Add (x(1), v(s))
        for 1 ⩽ i < q do
            Add (x(i + 1), x(i))
        end for
        Add (w(1), x(q))
    end if
end if
```

下面我们证明我们的加边方法的正确性.

定理 7. 算法 5 将任意给定的有向无环图 G' 加强为强连通图.

证明. 考虑 3 种情况.

1. $s = t$.

   若 $p = 0$ 且 $q > 1$，此时 G' 中只有孤立节点 $x(1), x(2), \cdots, x(q)$. 算法 5 将这些孤立节点连接成一个环，使得它们强连通.

   若 $p > 0$ 且 $q = 0$，此时 G' 中没有孤立节点. 算法 5 将所有 $v(i)$ 和 $w(i)$ 连接成一个环，$1 \leqslant i \leqslant p$，因此它们是强连通的. 显然那些既不是源也不是汇的点与这个环互相连通. 再考虑剩下的那些源和汇. 一方面，由条件 2、3，这些源可以连通到这个环，这个环可以连通到这些汇；另一方面，由于算法 5 加了边 $\big(w(i), v(i)\big)$，$p + 1 \leqslant i \leqslant s$，故这些汇可以通过这些源连通到这个环，这个环可以通过这些汇连通到这些源. 综上，加强后的有向图中任意两点都是互相连通的.

   若 $p > 0$ 且 $q > 0$，算法 5 将所有 $v(i)$、$w(i)$ 和 $x(j)$ 连接成一个环，$1 \leqslant i \leqslant p$，$1 \leqslant j \leqslant q$，因此它们是强连通的. 显然那些既不是源也不是汇的非孤立点与这个环互相连通. 再考虑剩下的那些源和汇. 一方面，由条件 2、3，这些源可以连通到这个环，这个环可以连通到这些汇；另一方面，由于算法 5 加了边 $\big(w(i), v(i)\big)$，$p + 1 \leqslant i \leqslant s$，故这些汇可以通过这些源连通到这个环，这个环可以通过这些汇连通到这些源. 综上，加强后的有向图中任意两点都是互相连通的.

2. $s < t$

若 $q=0$，算法 5 将所有 $v(i)$ 和 $w(j)$ 连接成一个环，$1 \leqslant i \leqslant p$，$j \in [1,p] \cup [s+1,t]$，因此它们是强连通的. 显然那些既不是源也不是汇的点与这个环互相连通. 再考虑剩下的那些源和汇. 一方面，由条件 2、3，这些源可以连通到这个环，这个环可以连通到这些汇；另一方面，由于算法 5 加了边 $\big(w(i),v(i)\big)$，$p+1 \leqslant i \leqslant s$，故这些汇可以通过这些源连通到这个环，这个环可以通过这些汇连通到这些源. 综上，加强后的有向图中任意两点都是互相连通的.

若 $q>0$，算法 5 将所有 $v(i)$、$w(j)$ 和 $x(k)$ 连接成一个环，$1 \leqslant i \leqslant p$，$j \in [1,p] \cup [s+1,t]$，$1 \leqslant k \leqslant q$，因此它们是强连通的. 显然那些既不是源也不是汇的非孤立点与这个环互相连通. 再考虑剩下的那些源和汇. 一方面，由条件 2、3，这些源可以连通到这个环，这个环可以连通到这些汇；另一方面，由于算法 5 加了边 $\big(w(i),v(i)\big)$，$p+1 \leqslant i \leqslant s$，故这些汇可以通过这些源连通到这个环，这个环可以通过这些汇连通到这些源. 综上，加强后的有向图中任意两点都是互相连通的.

3. $s>t$

若 $q=0$，算法 5 将所有 $v(i)$ 和 $w(j)$ 连接成一个环，$1 \leqslant j \leqslant p$，$i \in [1,p] \cup [t+1,s]$，因此它们是强连通的. 显然那些既不是源也不是汇的点与这个环互相连通. 再考虑剩下的那些源和汇. 一方面，由条件 2、3，这些源可以连通到这个环，这个环可以连通到这些汇；另一方面，由于算法 5 加了边 $\big(w(i),v(i)\big)$，$p+1 \leqslant i \leqslant t$，故这些汇可以通过这些源连通到这个环，这个环可以通过这些汇连通到这些源. 综上，加强后的有向图中任意两点都是互相连通的.

若 $q>0$，算法 5 将所有 $v(i)$、$w(j)$ 和 $x(k)$ 连接成一个环，$1 \leqslant j \leqslant p$，$i \in [1,p] \cup [t+1,s]$，$1 \leqslant k \leqslant q$，因此它们是强连通的. 显然那些既不是源也不是汇的非孤立点与这个环互相连通. 再考虑剩下的那些源和汇. 一方面，由条件 2、3，这些源可以连通到这个环，这个环可以连通到这些汇；另一方面，由于算法 5 加了边 $\big(w(i),v(i)\big)$，$p+1 \leqslant i \leqslant t$，故这些汇可以通过这些源连通到这个环，这个环可以通过这些汇连通到这些源. 综上，加强后的有向图中任意两点都是互相连通的.

综上所述，算法 5 将 G' 加强为强连通图. □

至此，我们的程序以 $O(V+E)$ 的时间复杂度，可将任意的有向图加强为强连通图.

参考文献

[1] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[2] Kapali Eswaran and Robert Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[3] S. Raghavan. A note on Eswaran and Tarjan' s algorithm for the strong connectivity augmentation problem. In *The Next Wave in Computing, Optimization, and Decision Technologies*, pages 19–26. Springer, 2005.

[4] Micha Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981.

[5] David J. Pearce. A space-efficient algorithm for finding strongly connected components. *Information Processing Letters*, 116(1):47–52, 2016.

[6] András Frank and Tibor Jordán. Minimal edge-covering of pairs of sets. *Journal of Combinatorial Theory*, 65(1):73–110, 1995.