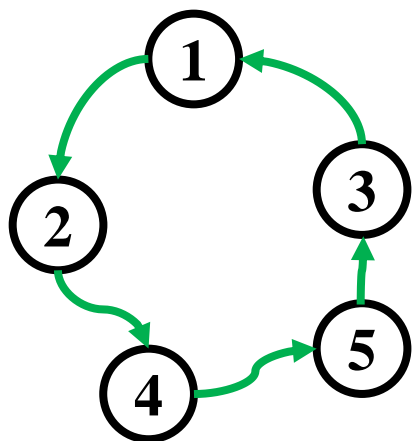


图算法篇：强连通分量

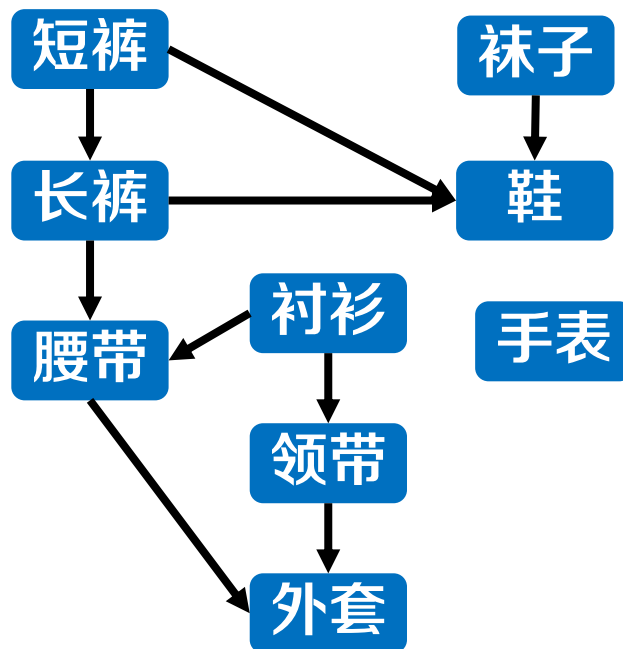
童咏昕

北京航空航天大学
计算机学院

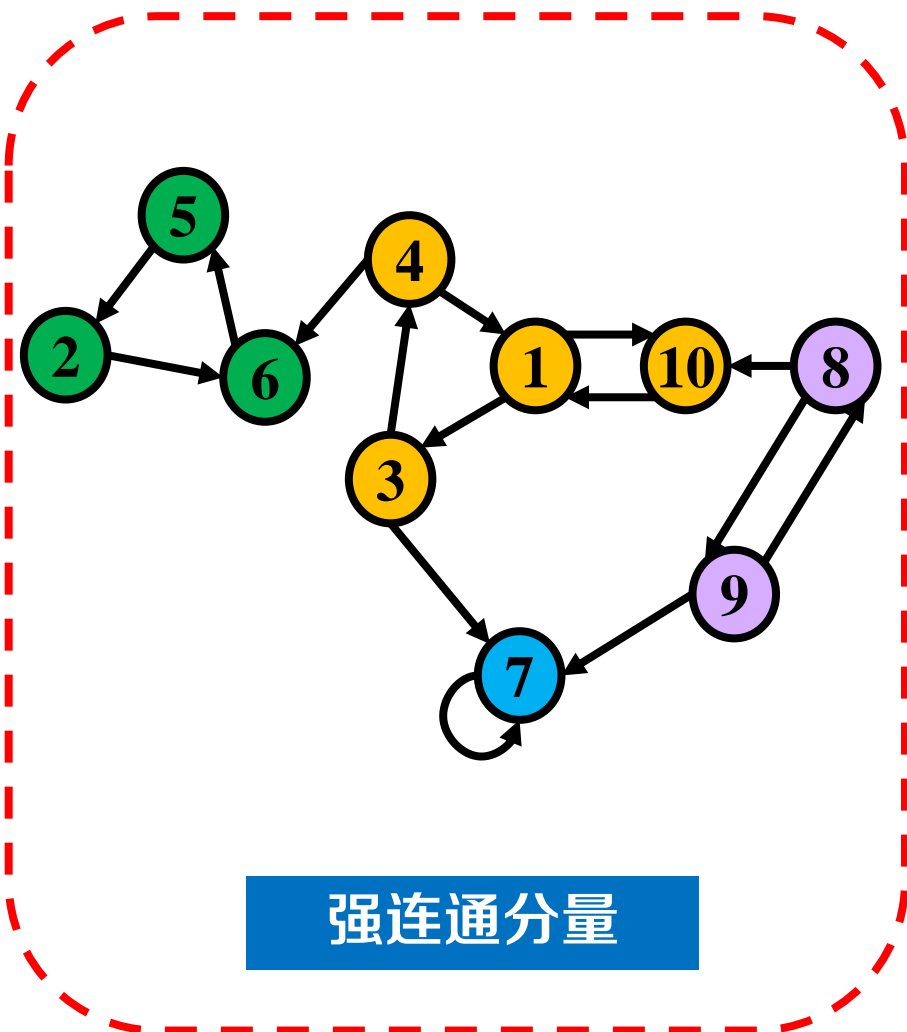
中国大学MOOC北航《算法设计与分析》



环路的存在性判断



拓扑排序



强连通分量

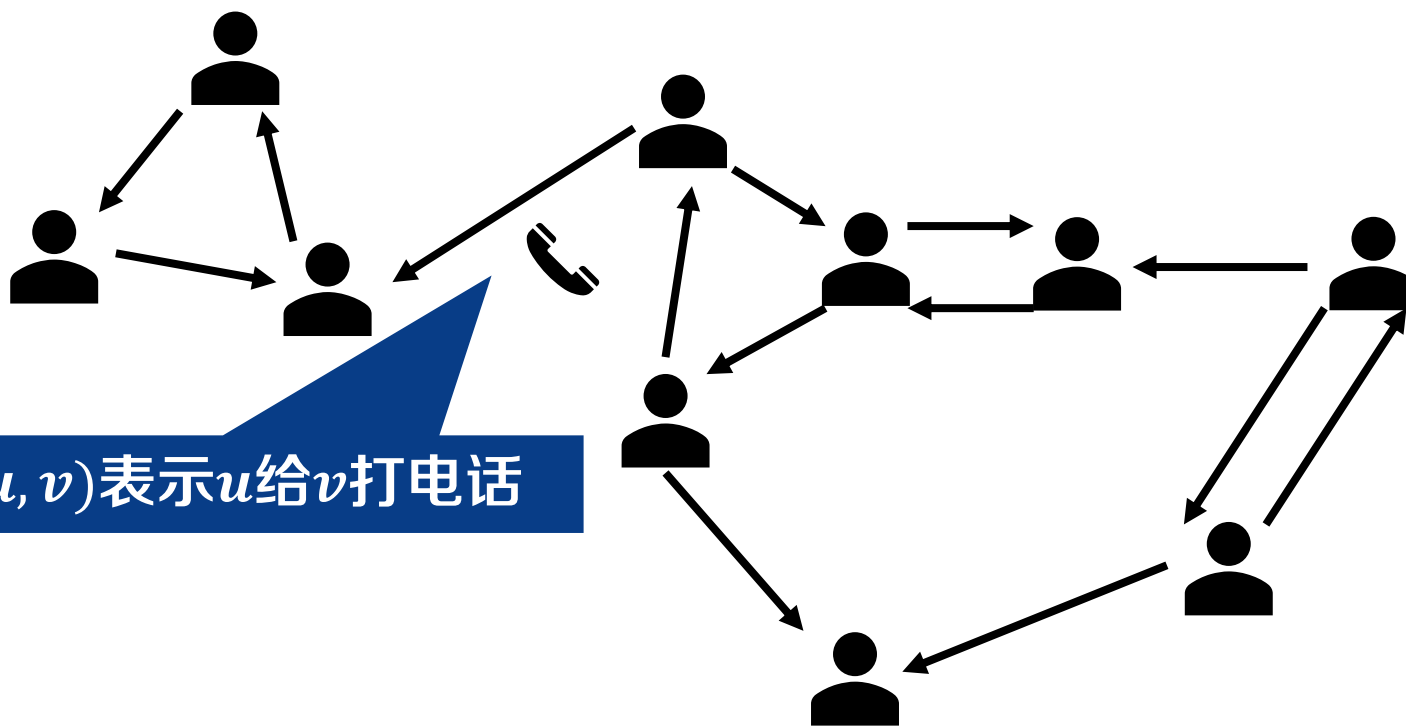
问题背景与定义

算法框架与实例

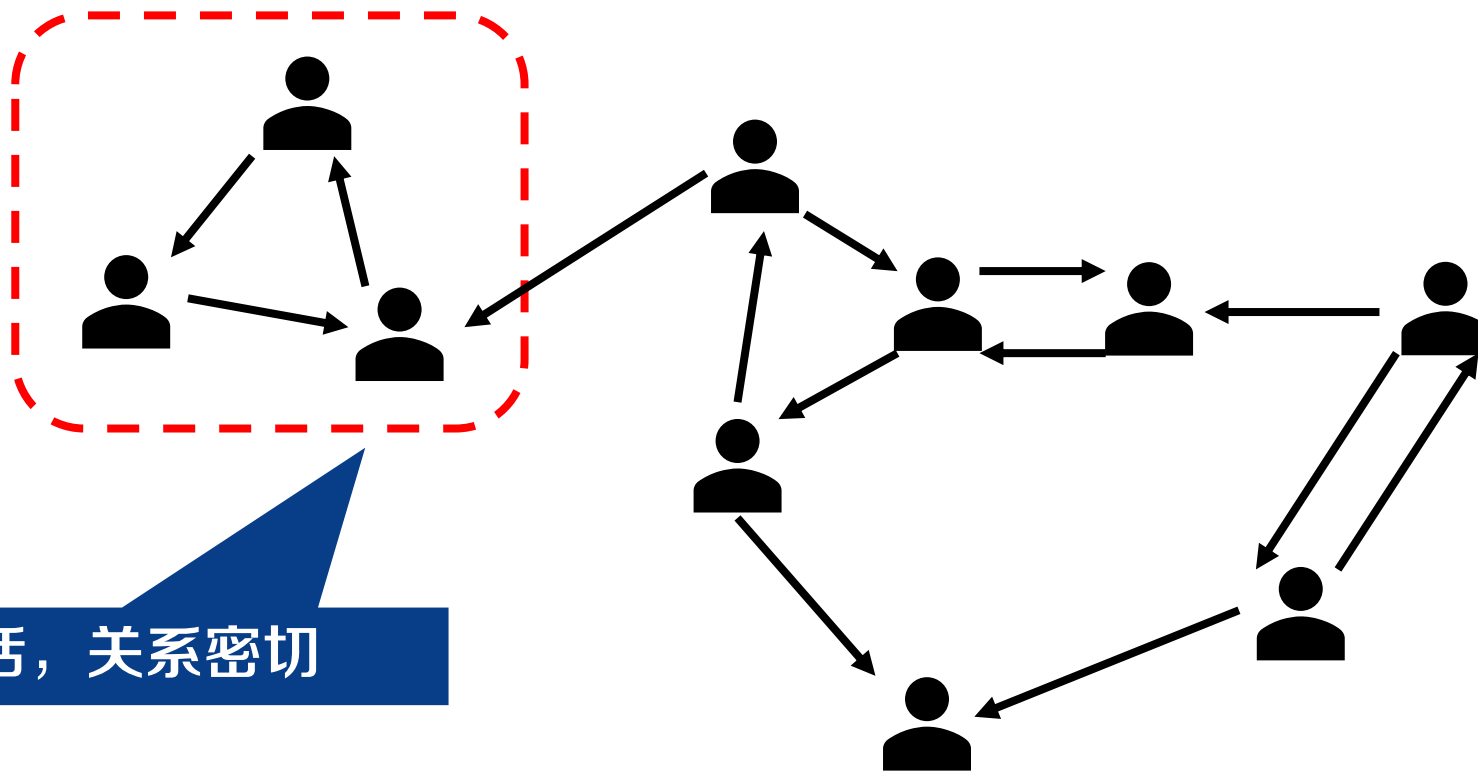
伪代码与复杂度

算法正确性证明

- 社交圈划分
 - 如何把人群按通话记录划分成不同的社交圈？

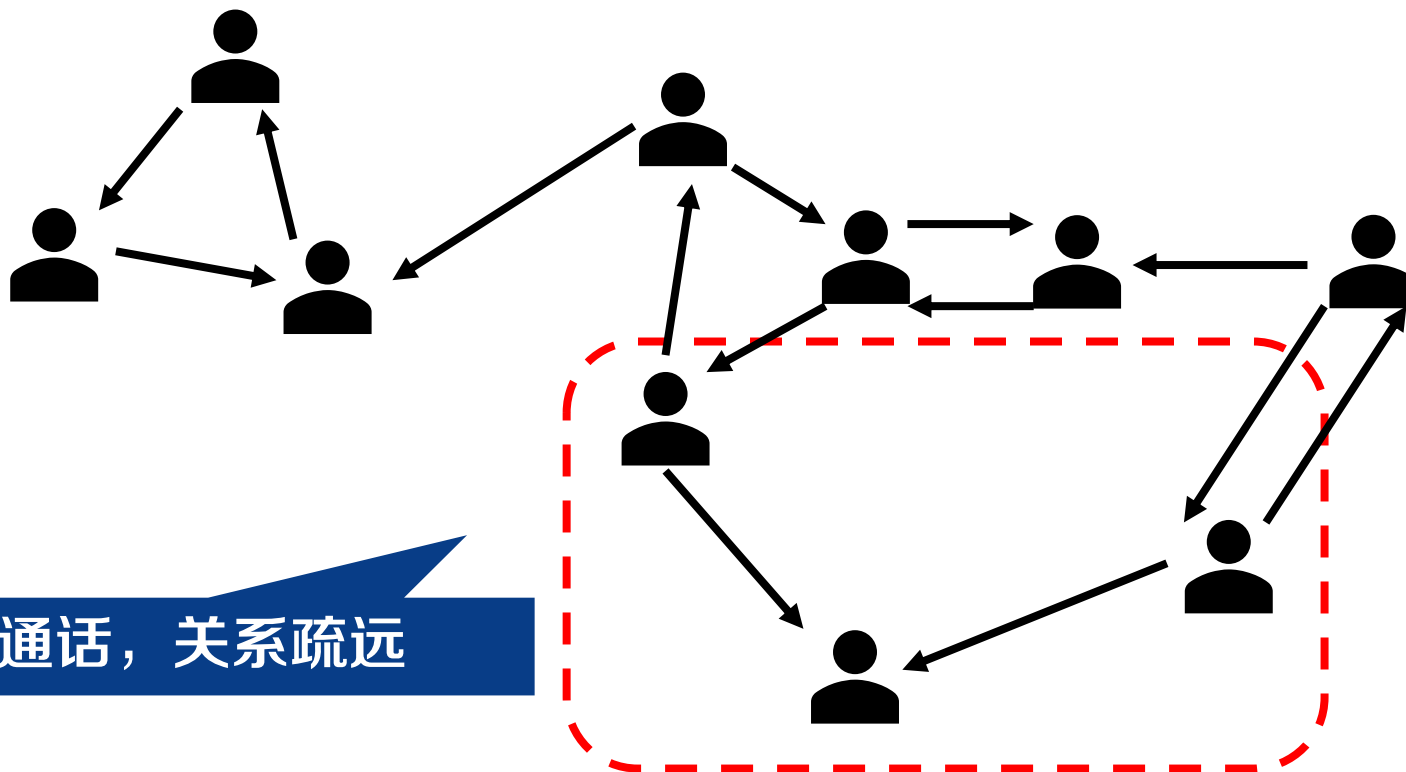


- 社交圈划分
 - 如何把人群按通话记录划分成不同的社交圈？



相互通话，关系密切

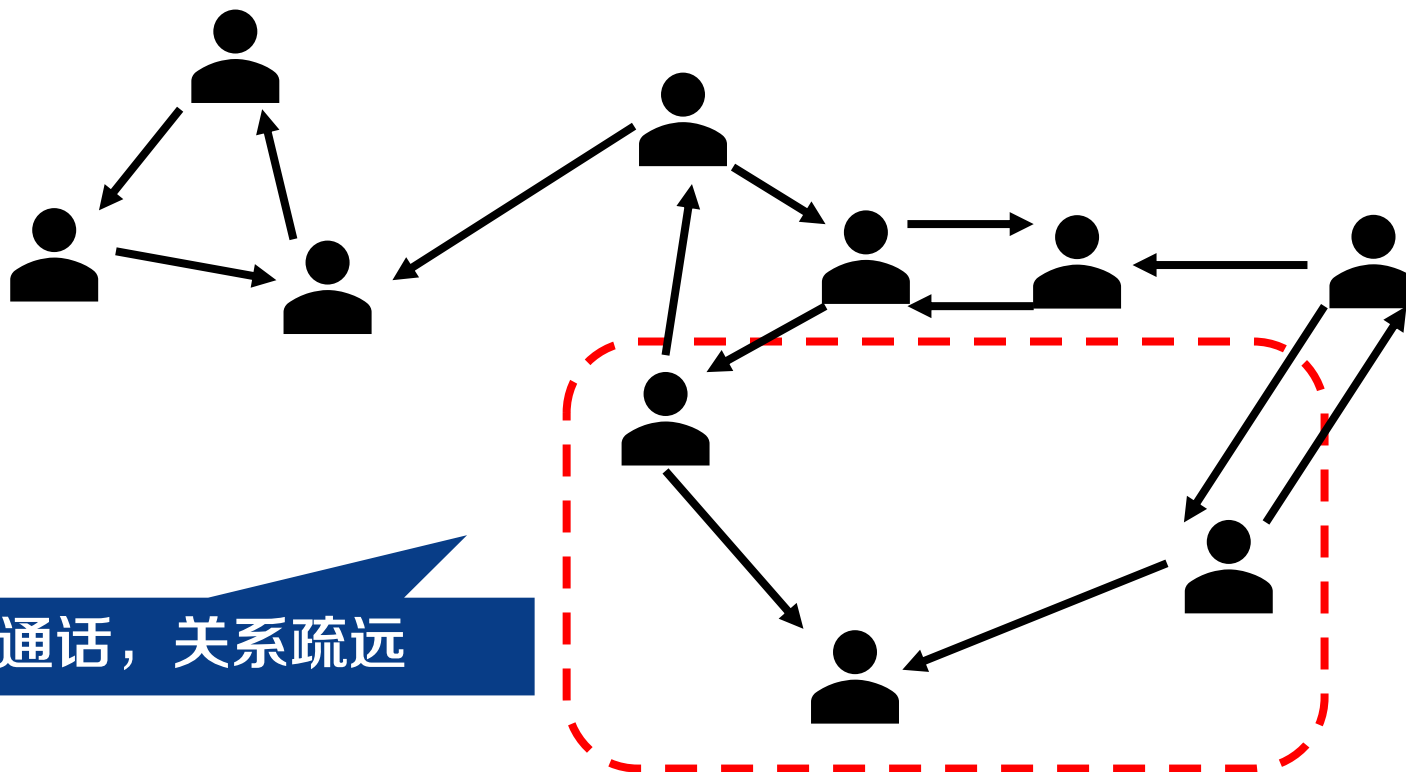
- 社交圈划分
 - 如何把人群按通话记录划分成不同的社交圈？



单向通话，关系疏远

- 社交圈划分

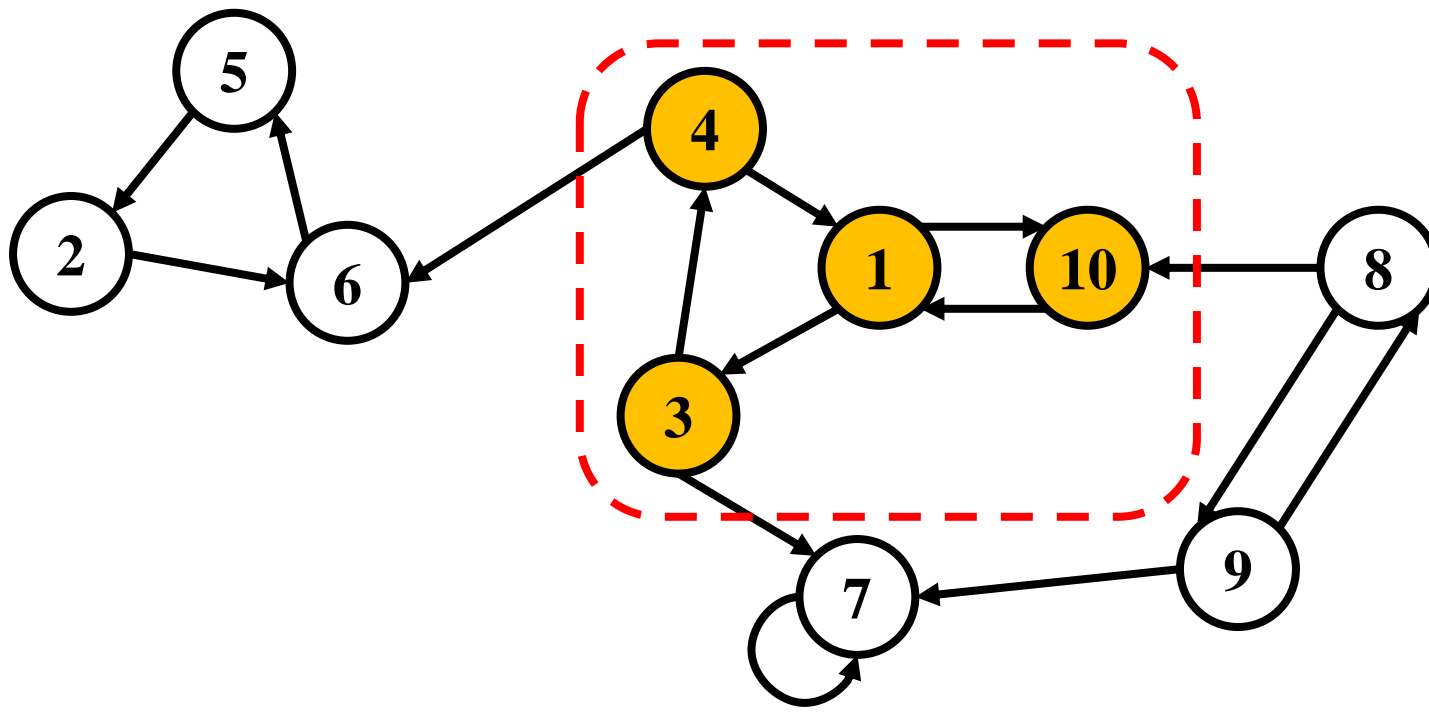
- 如何把人群按通话记录划分成不同的社交圈？



单向通话，关系疏远

问题：如何严格定义关系的亲密程度？

- 社交圈划分
 - 如何把人群按通话记录划分成不同的社交圈？
- 强连通分量
 - 一个强连通分量是顶点的子集

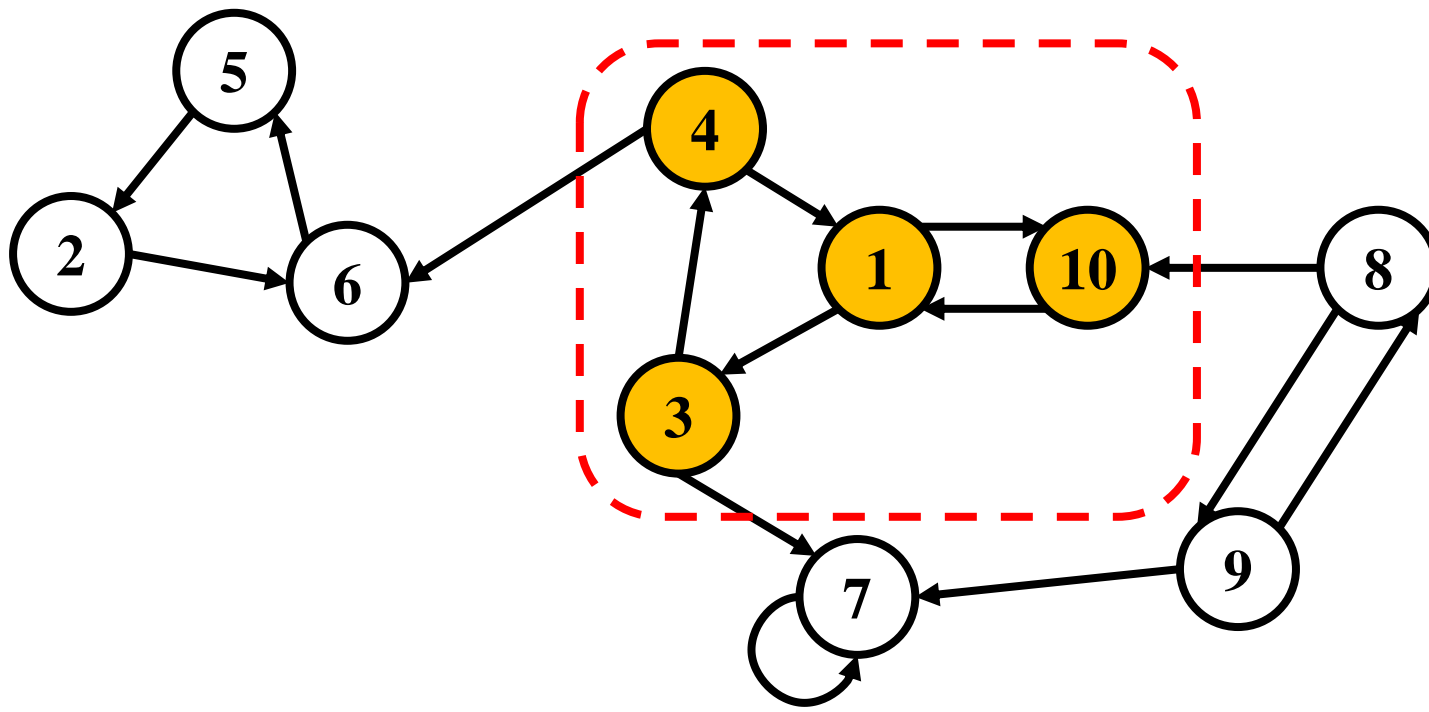


- 社交圈划分

- 如何把人群按通话记录划分成不同的社交圈？

- 强连通分量

- 一个强连通分量是顶点的子集
- 强连通分量中任意两点相互可达

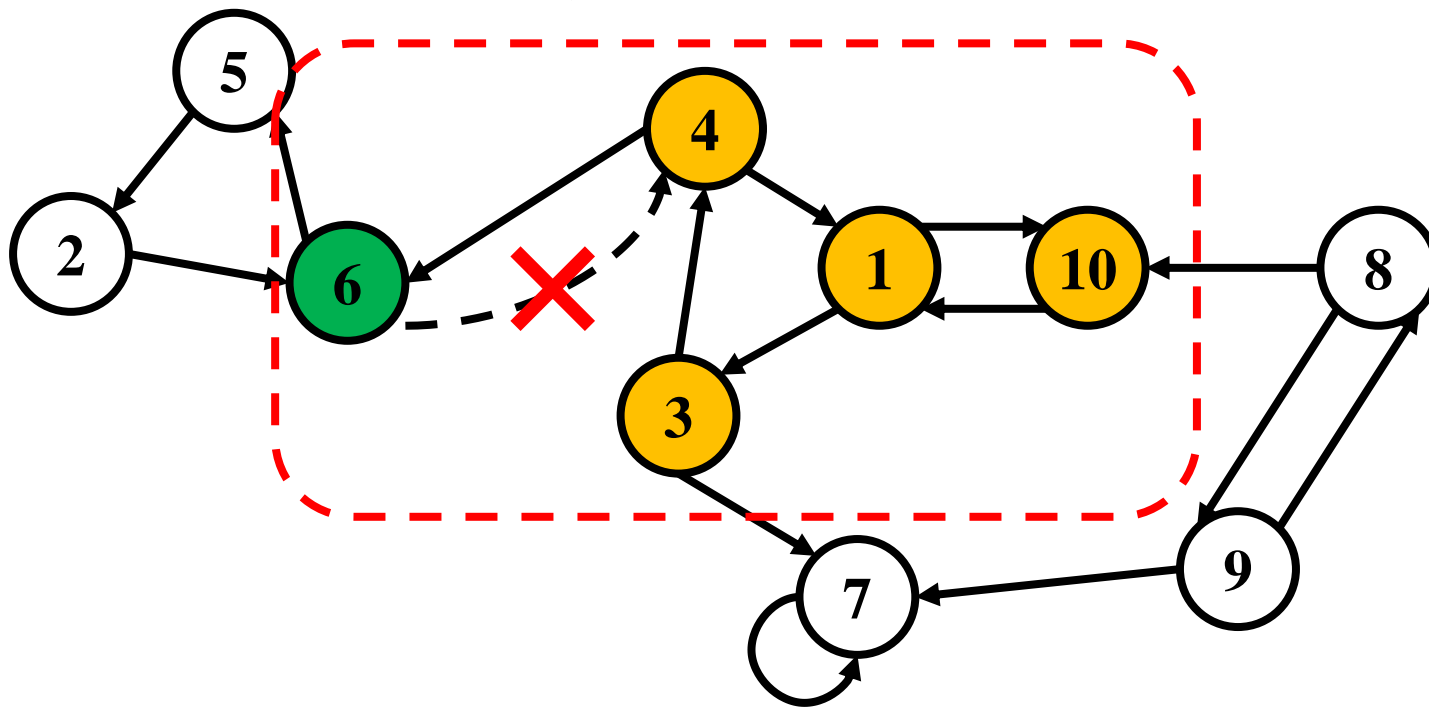


- 社交圈划分

- 如何把人群按通话记录划分成不同的社交圈？

- 强连通分量

- 一个强连通分量是顶点的子集
- 强连通分量中任意两点相互可达
- 满足**最大性**：加入新顶点，不保证相互可达

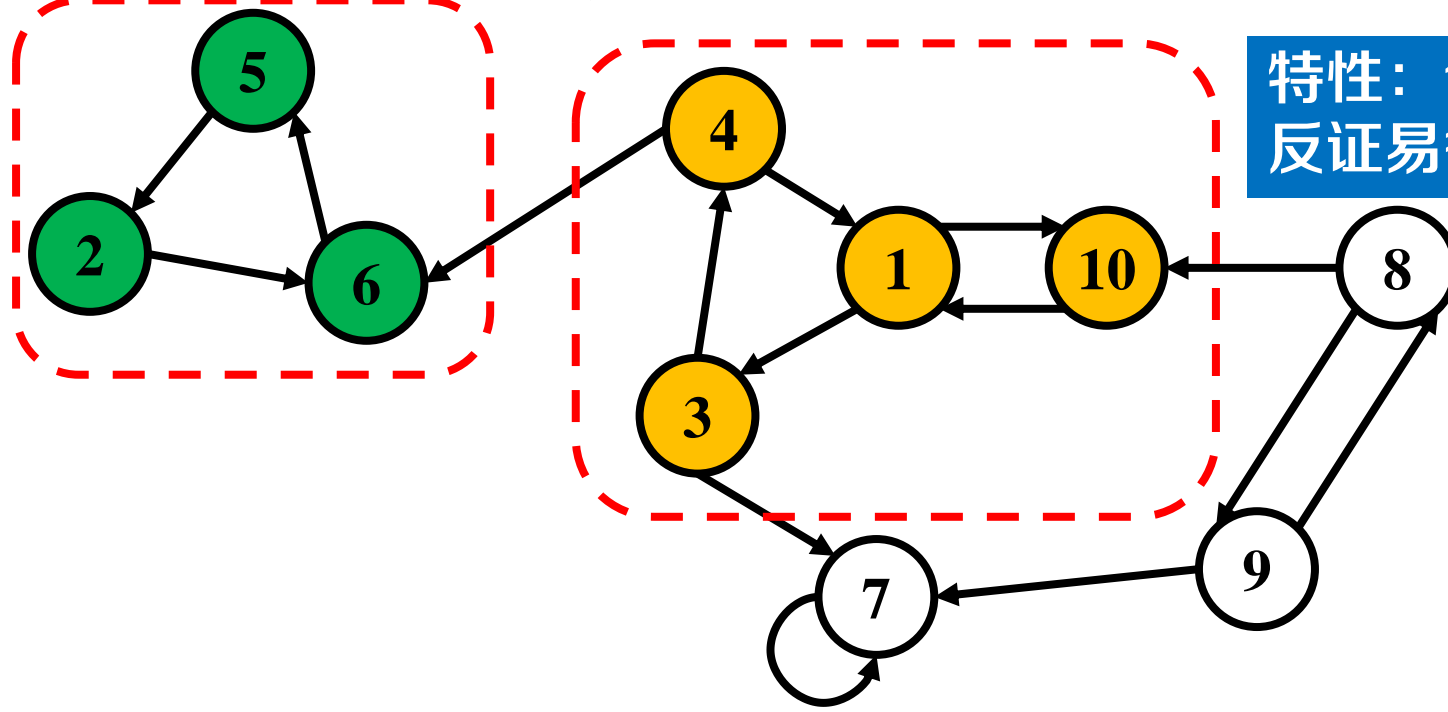


- 社交圈划分

- 如何把人群按通话记录划分成不同的社交圈？

- 强连通分量

- 一个强连通分量是顶点的子集
- 强连通分量中任意两点相互可达
- 满足**最大性**：加入新顶点，不保证相互可达



特性：任意两强连通分量不相交
反证易得：若相交，破坏最大性

强连通分量

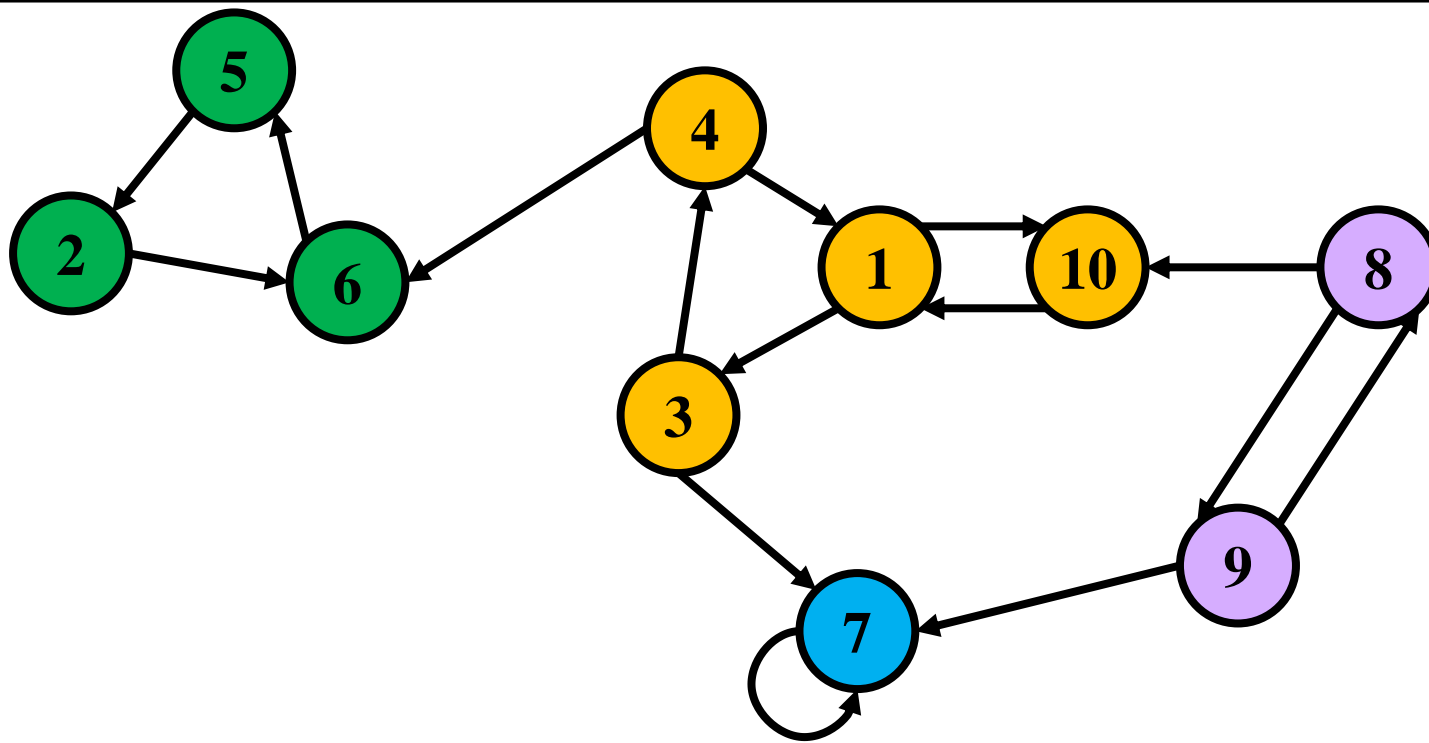
Strongly Connected Components

输入

- 有向图 $G = \langle V, E \rangle$

输出

- 图的所有强连通分量 C_1, C_2, \dots, C_n



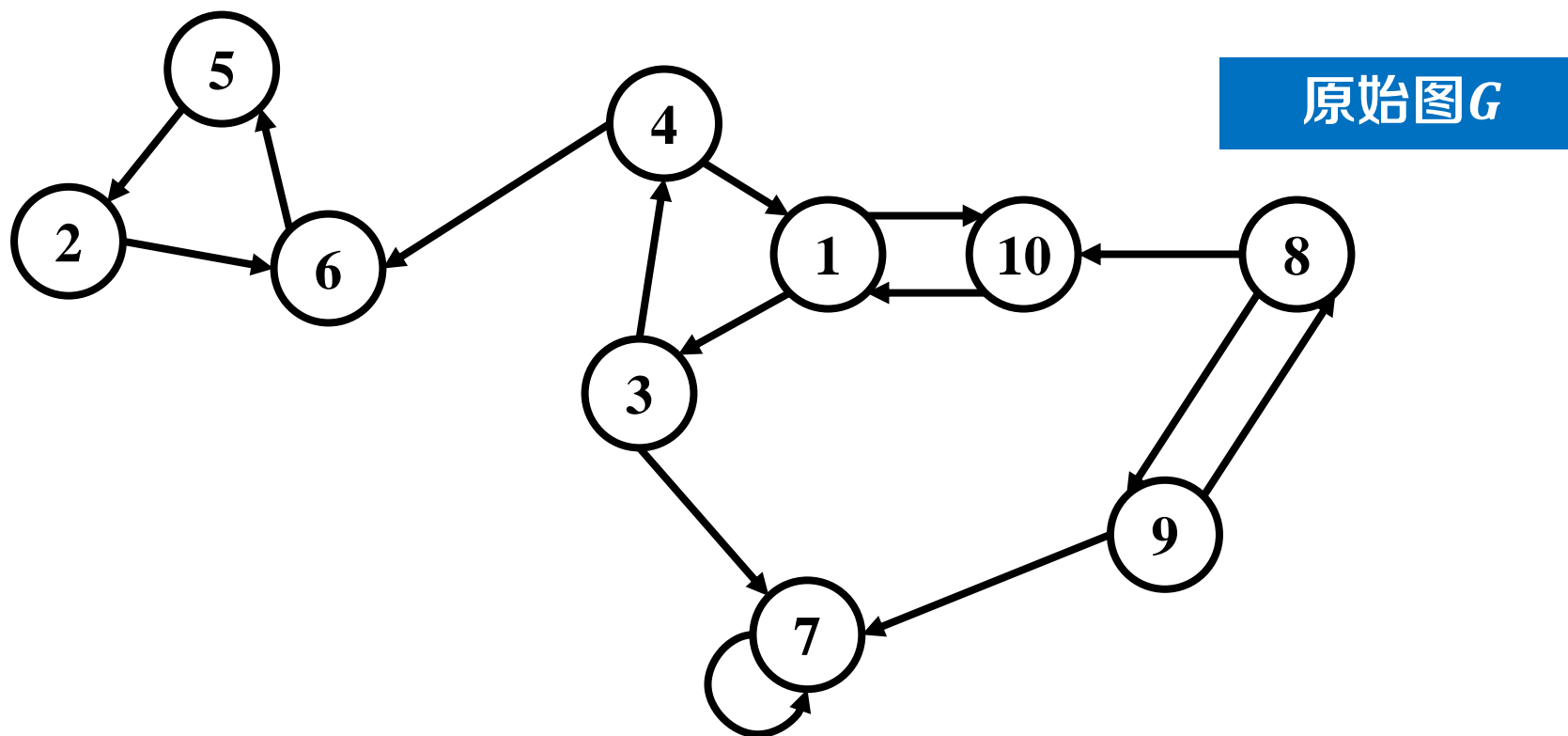
问题背景与定义

算法框架与实例

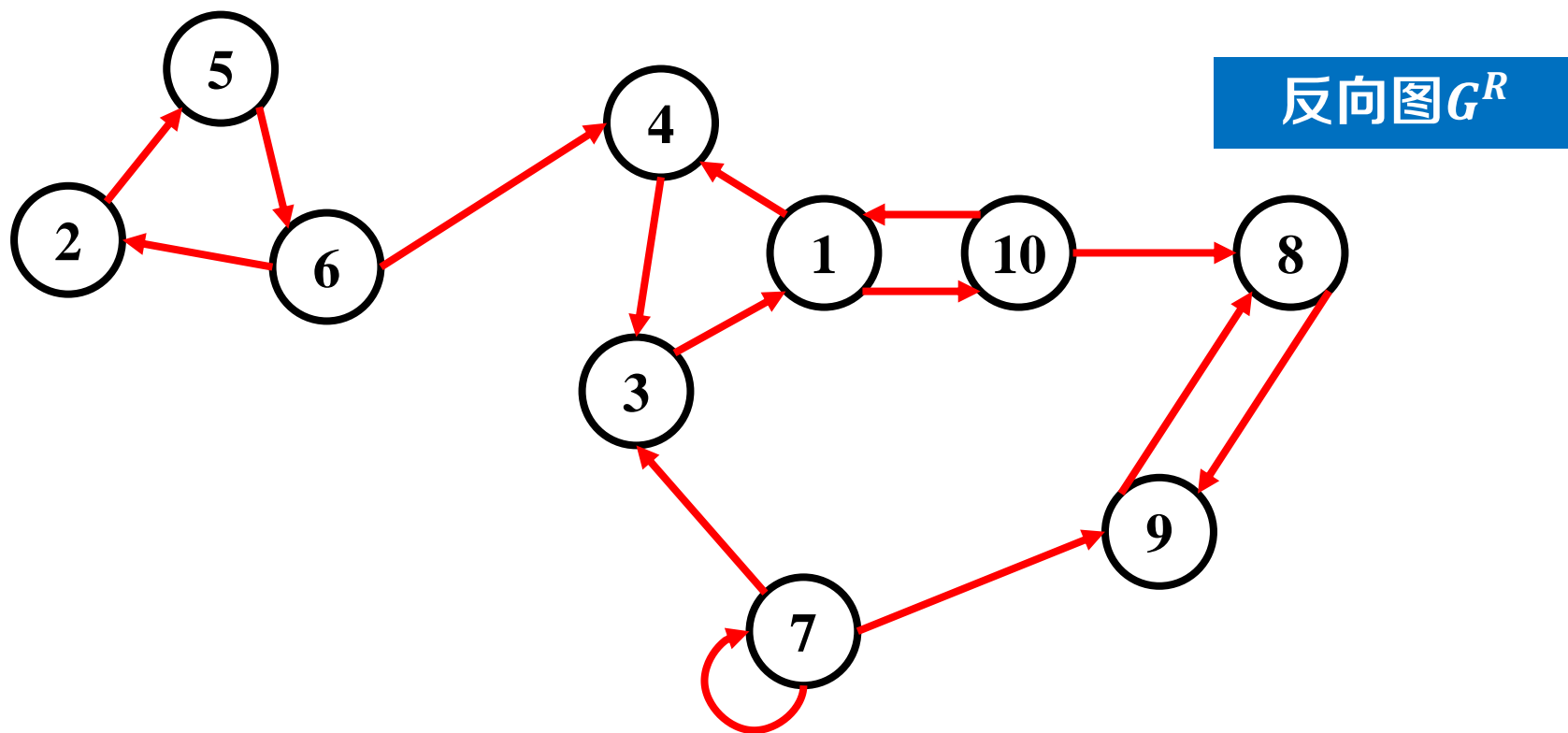
伪代码与复杂度

算法正确性证明

- 步骤1：把边反向，得到反向图 G^R



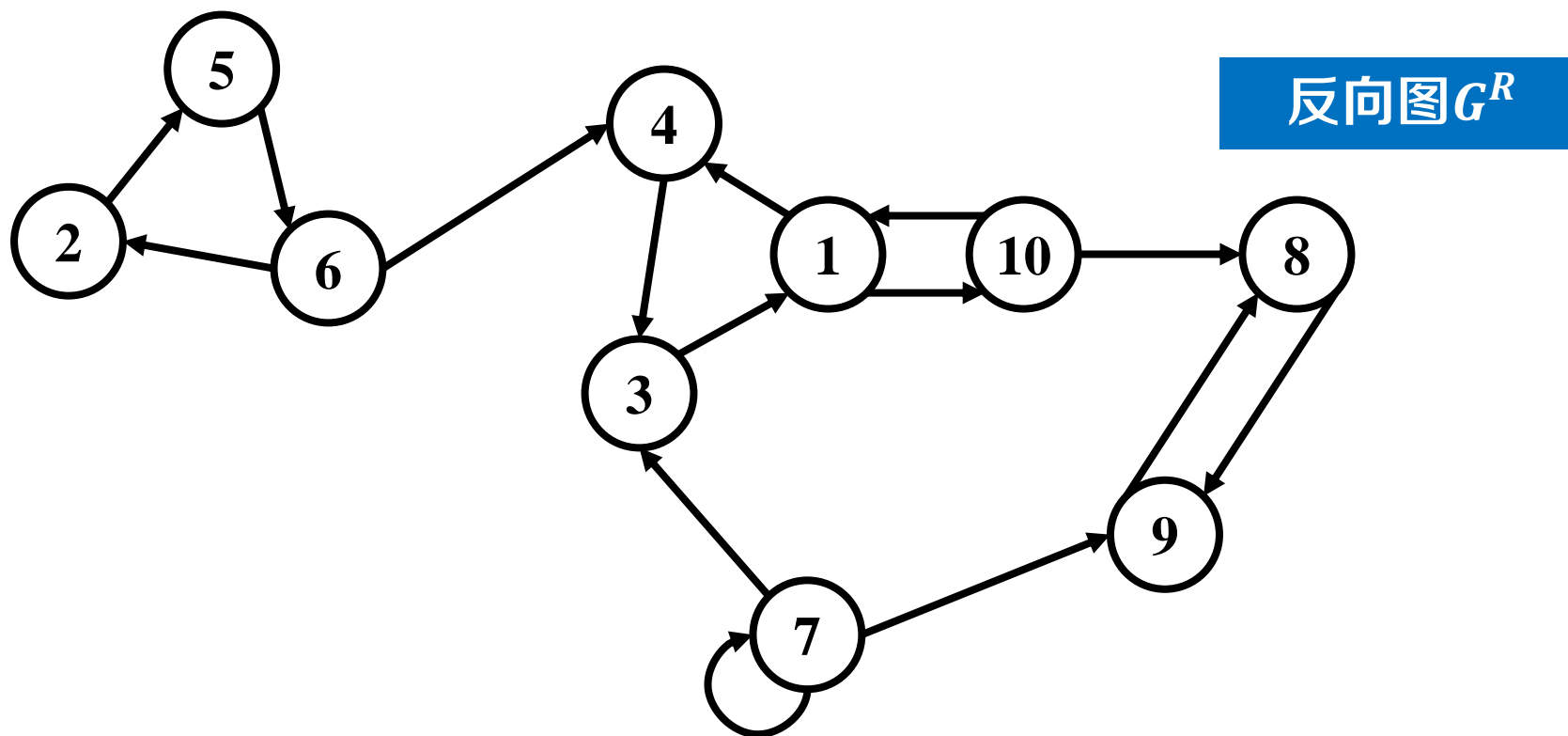
- 步骤1: 把边反向, 得到反向图 G^R



算法框架与实例



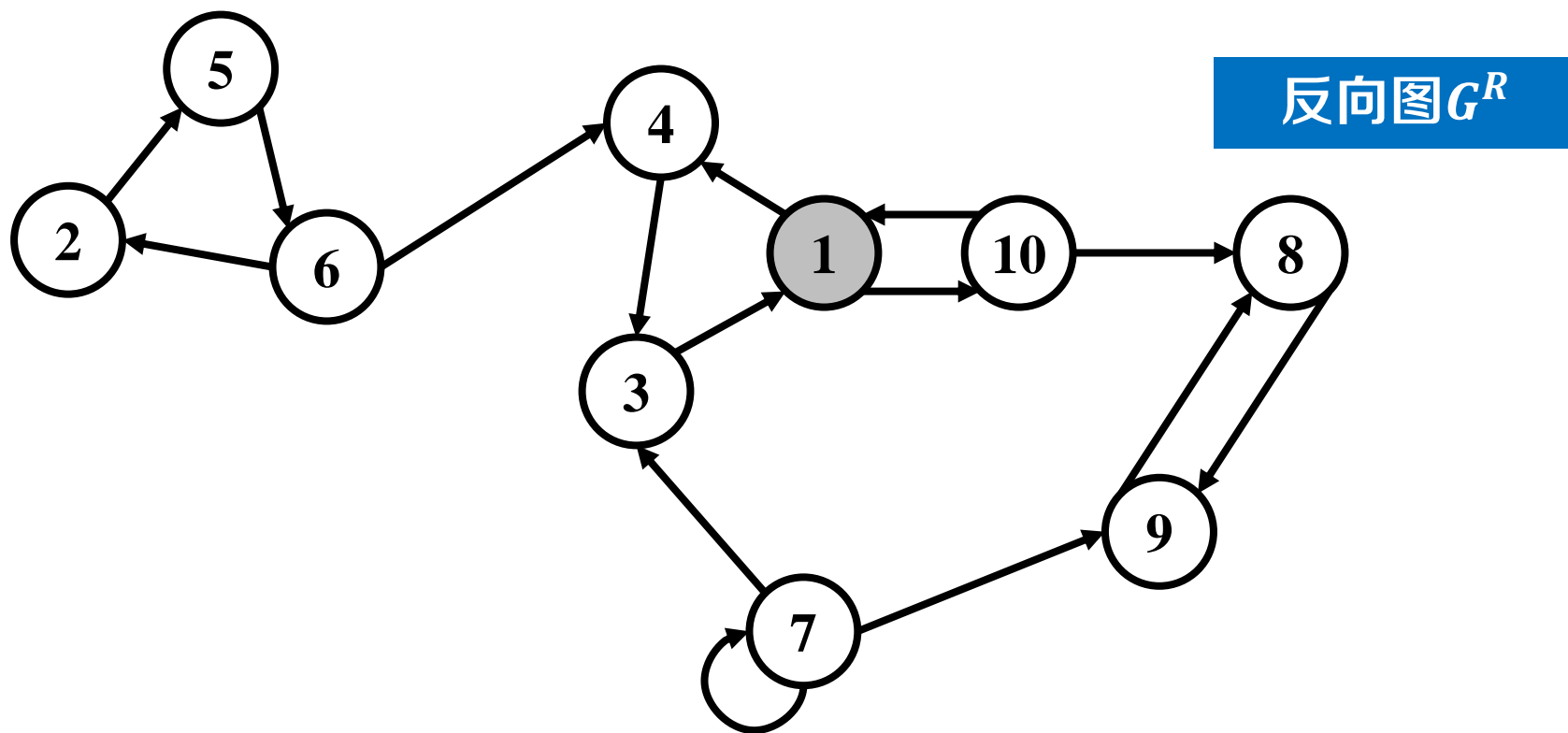
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L



算法框架与实例



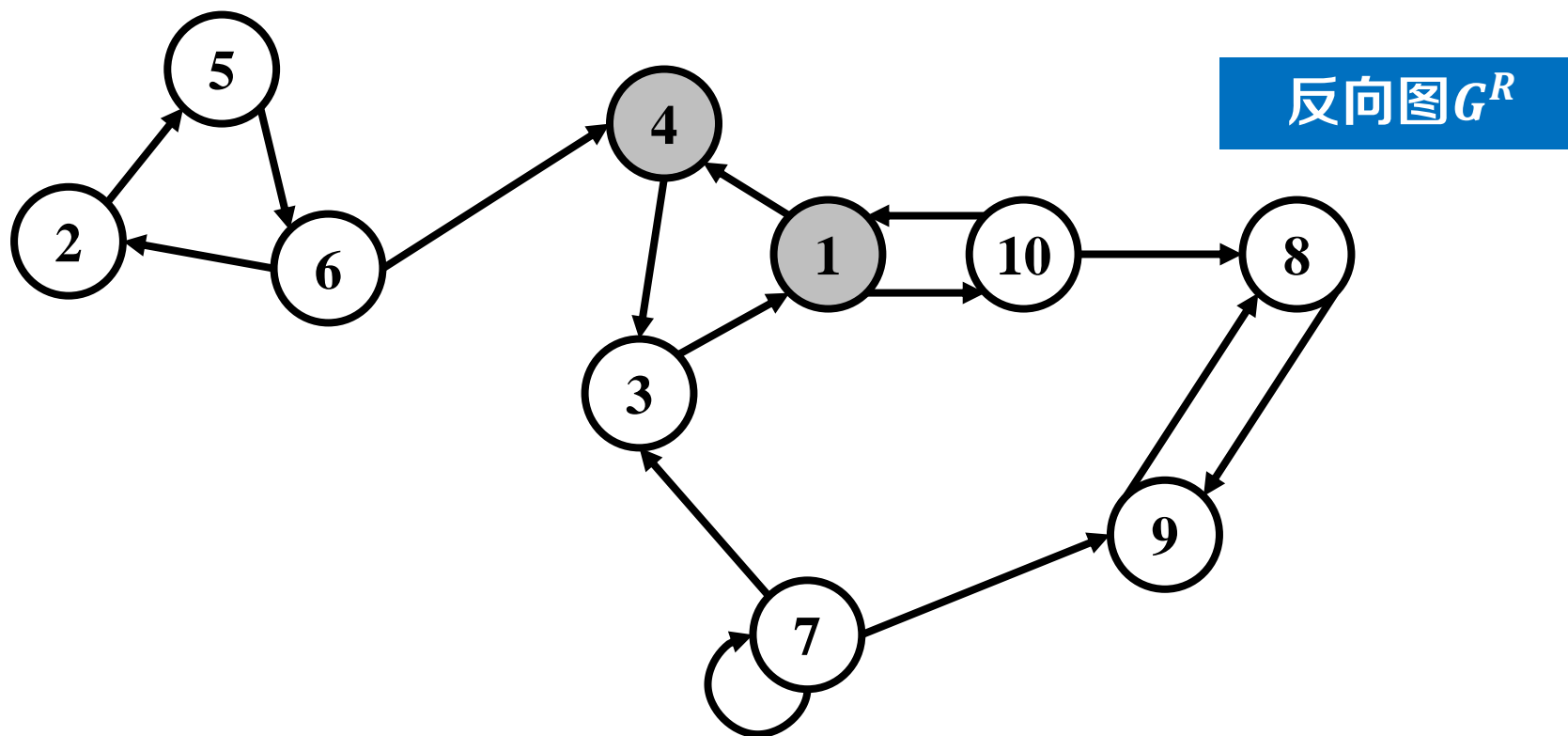
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L



算法框架与实例



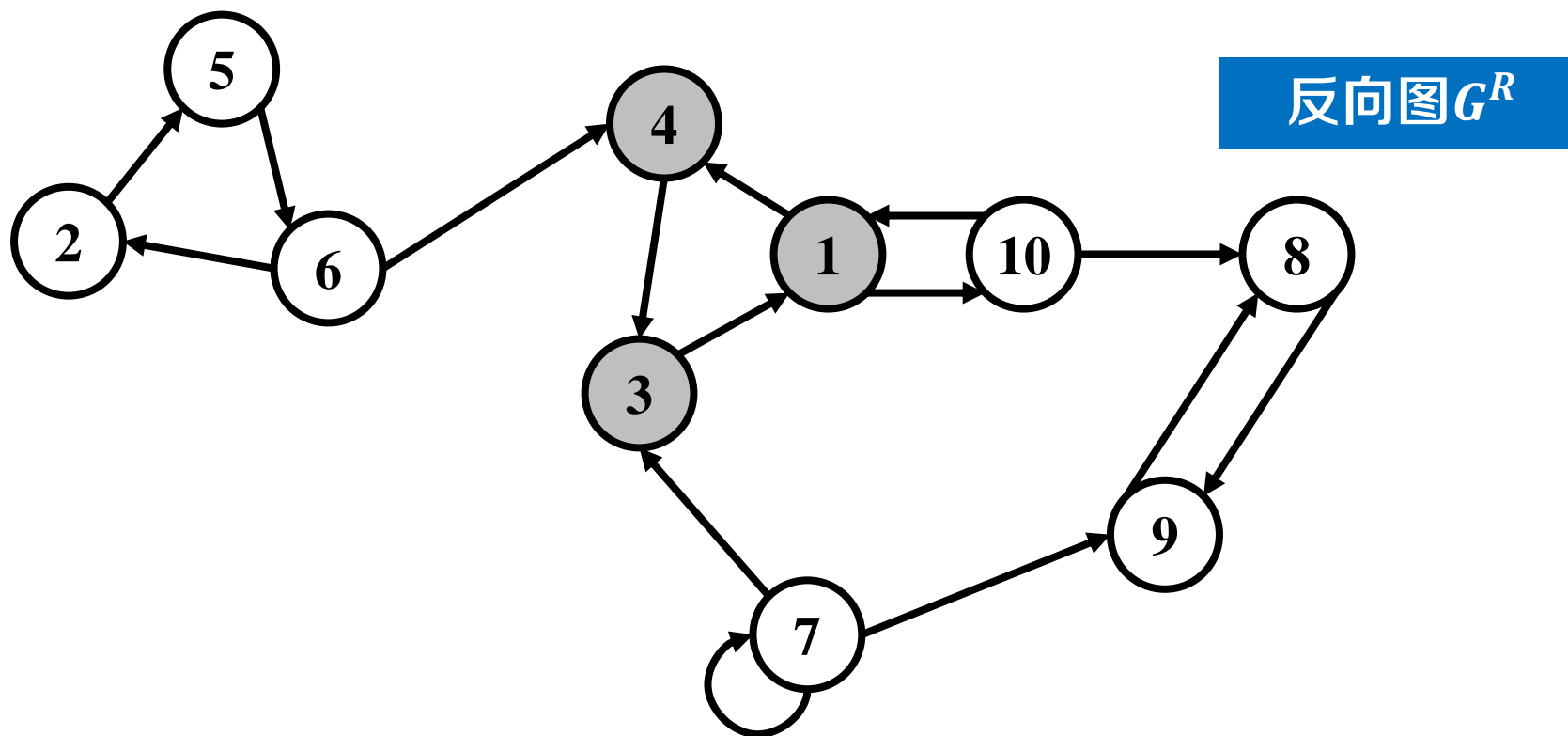
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L



算法框架与实例



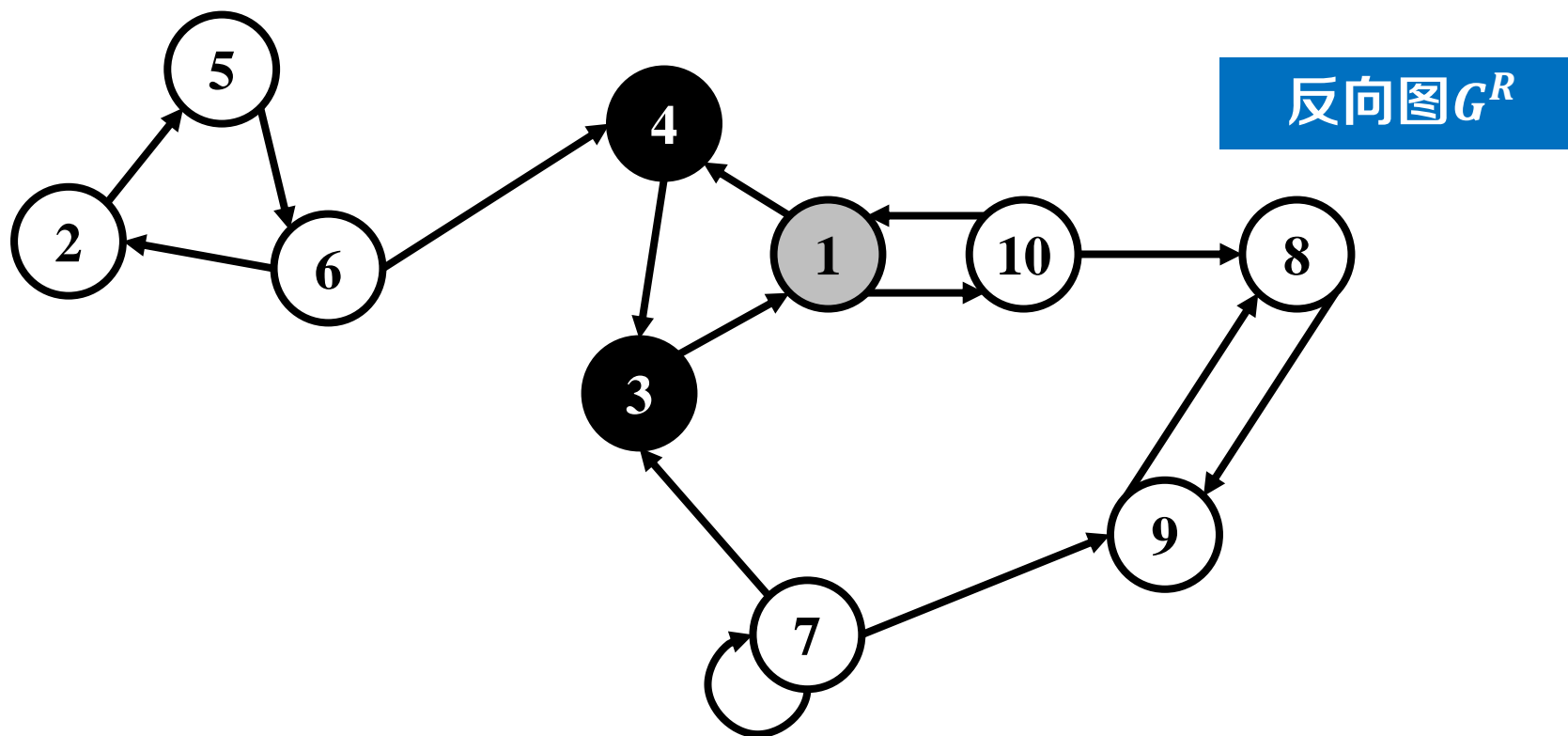
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L



算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

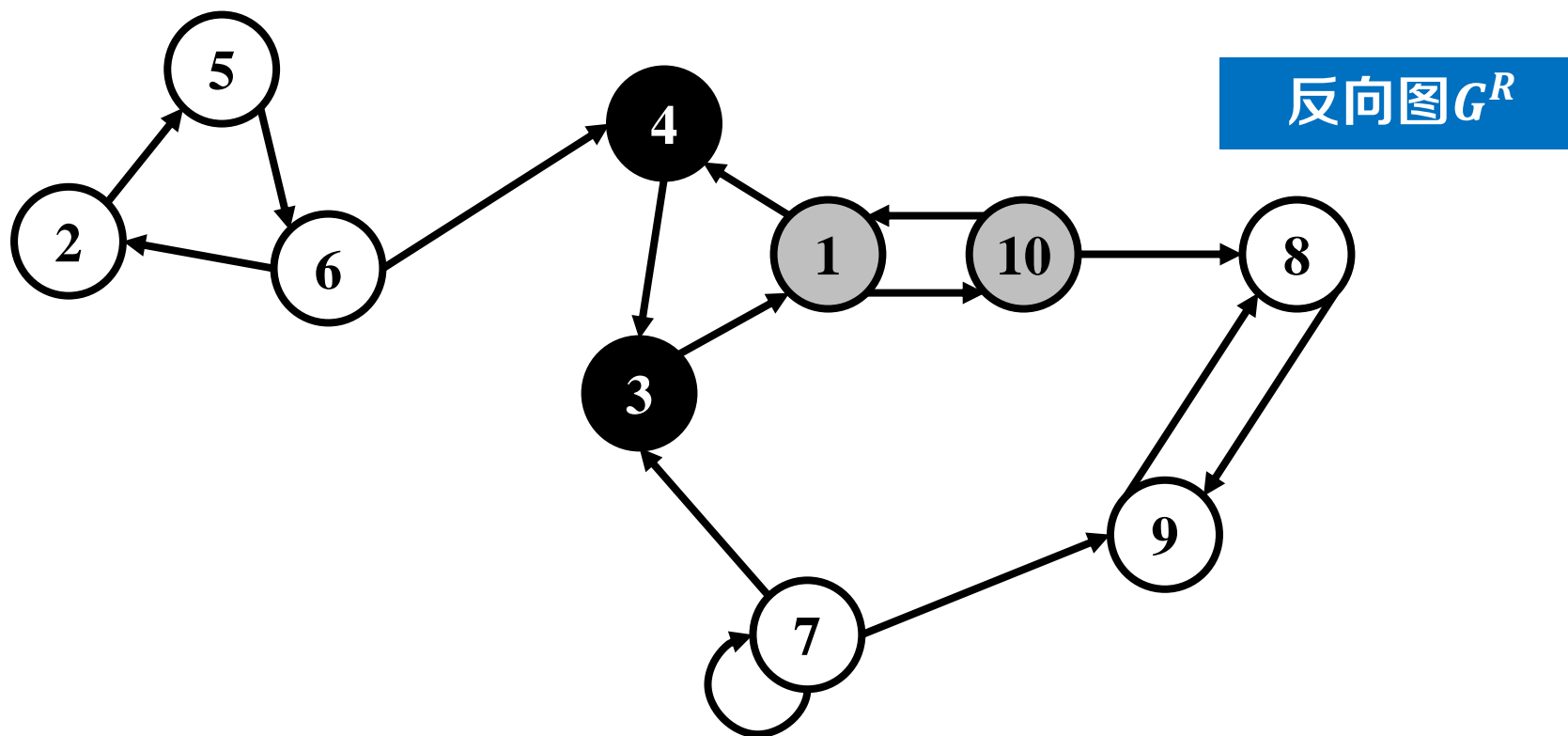


L	3	4								
-----	---	---	--	--	--	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

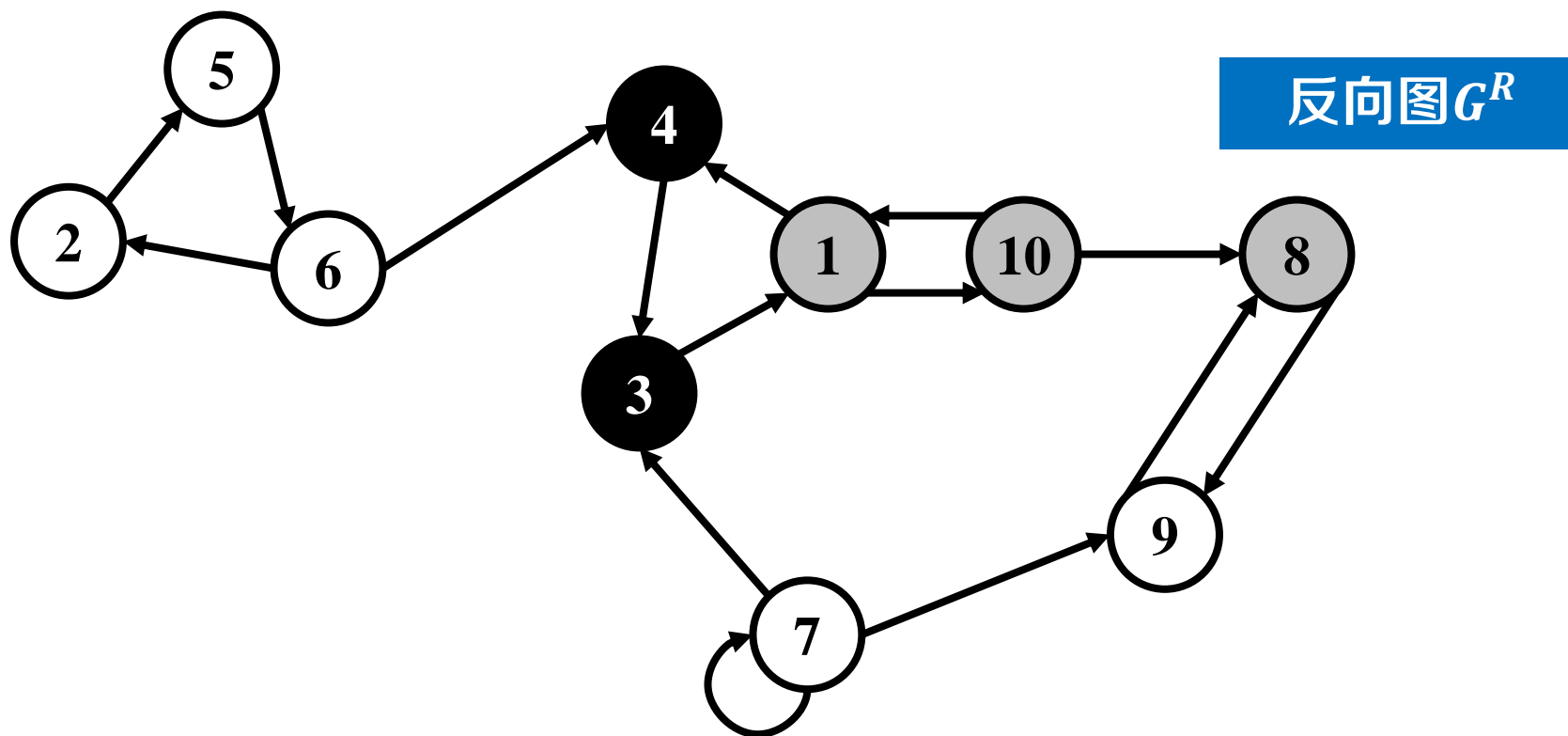


L	3	4								
-----	---	---	--	--	--	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

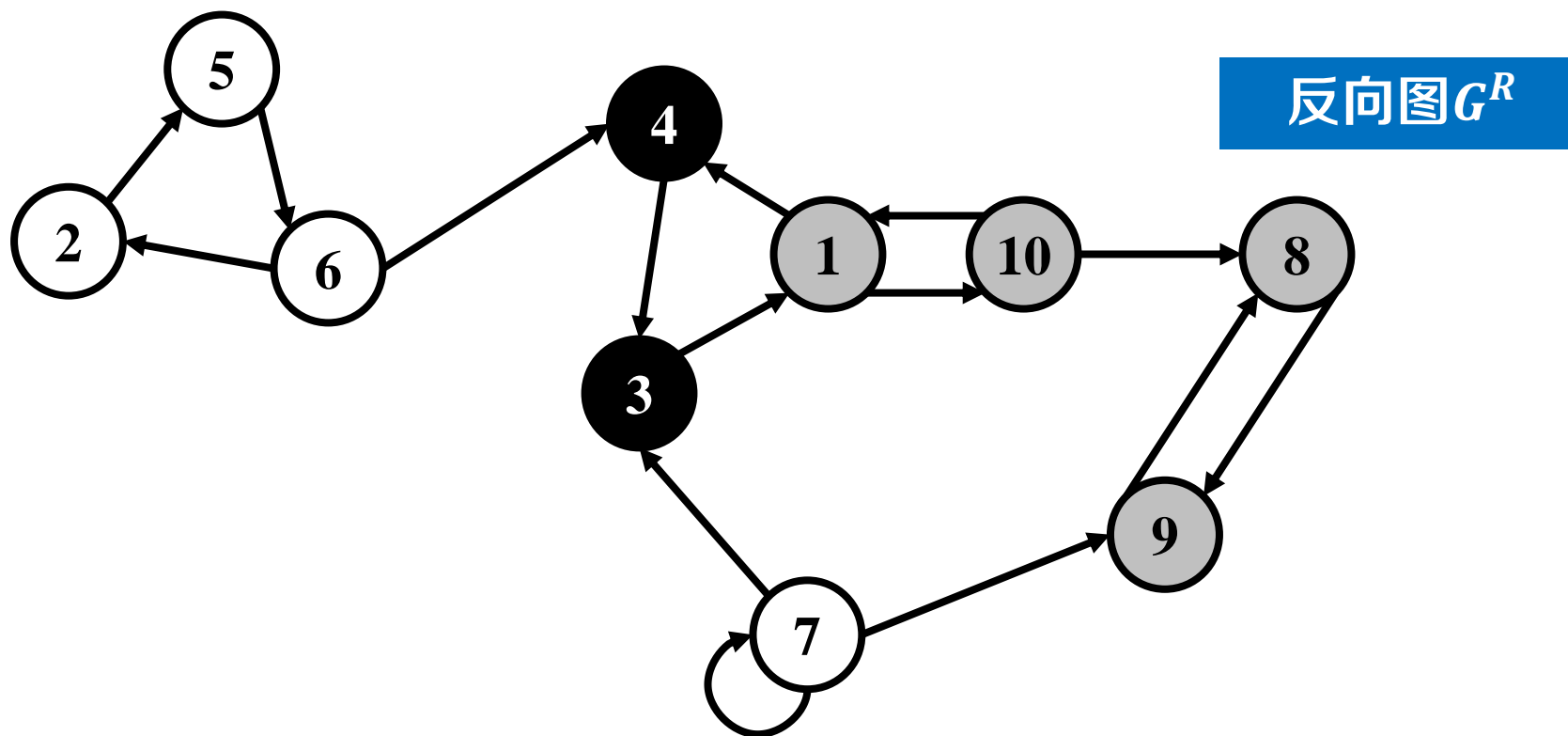


L	3	4								
-----	---	---	--	--	--	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

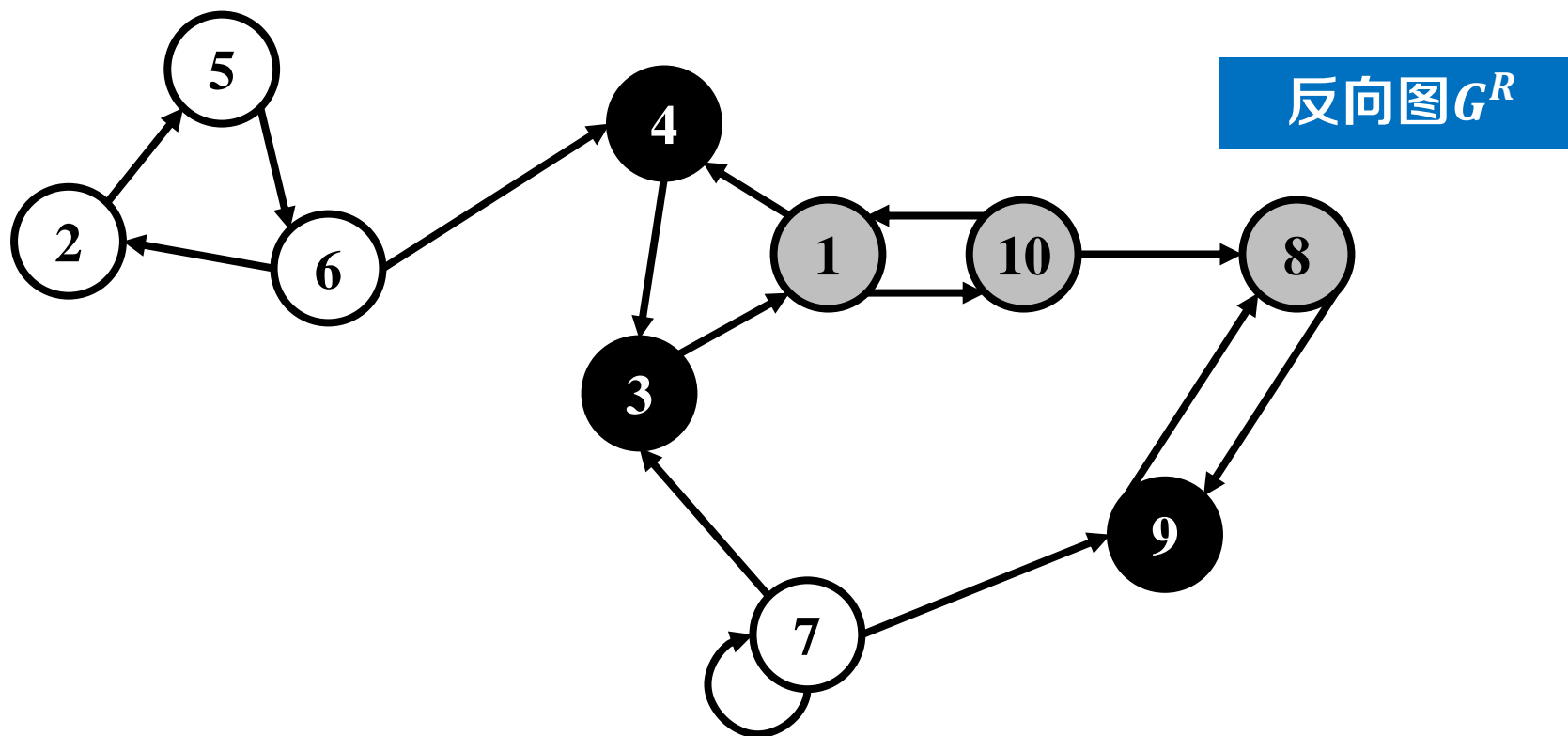


L	3	4								
-----	---	---	--	--	--	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

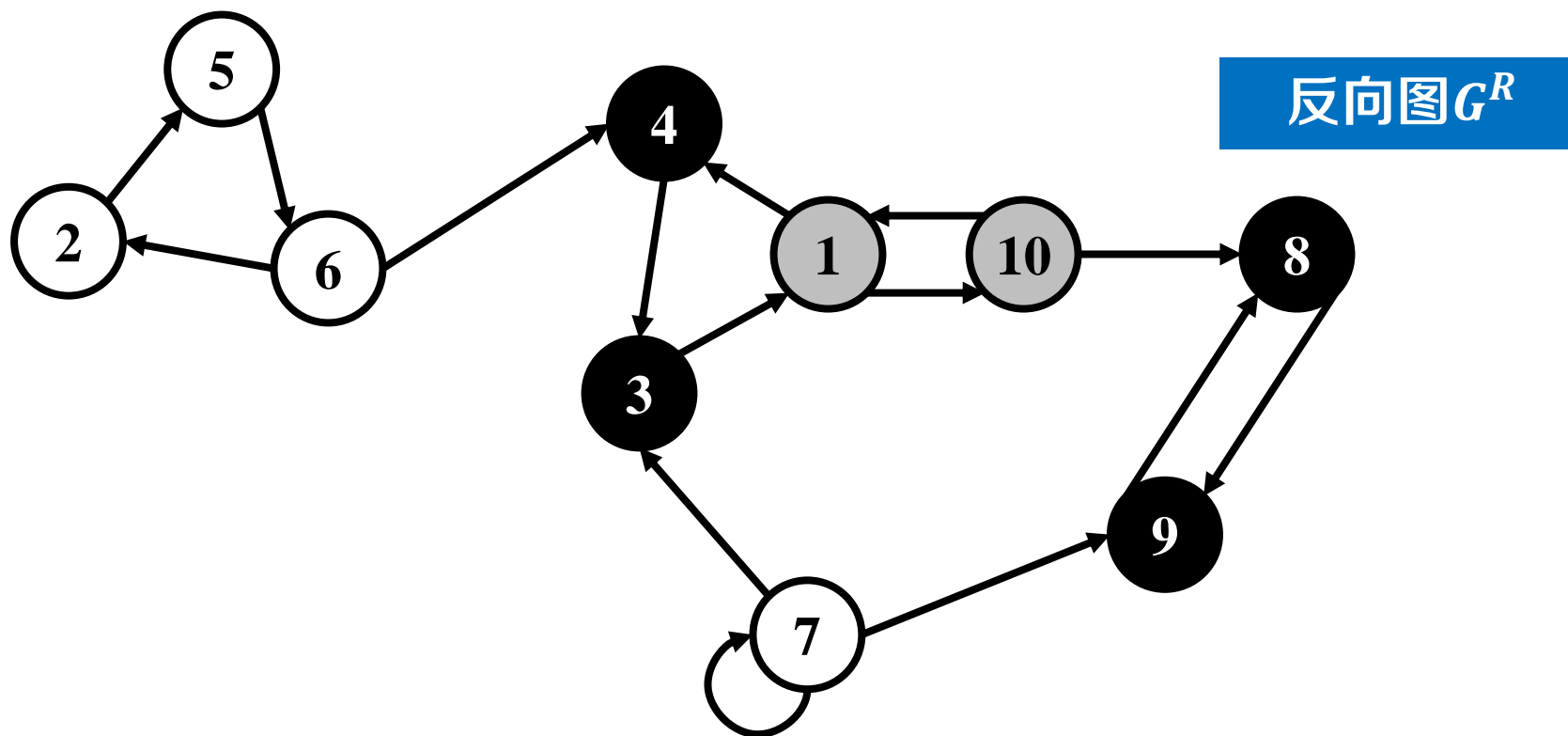


L	3	4	9							
-----	---	---	---	--	--	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

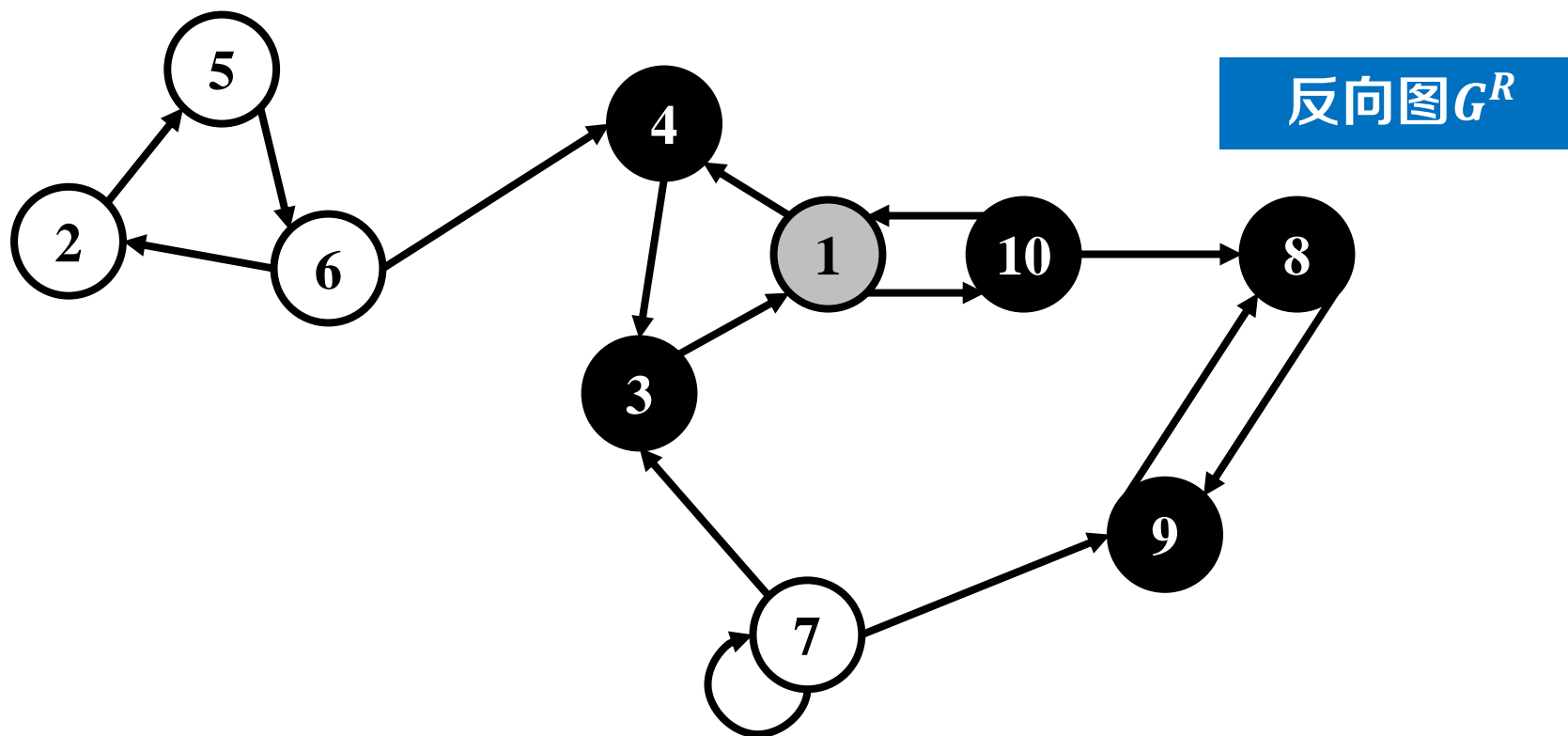


L	3	4	9	8						
-----	---	---	---	---	--	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

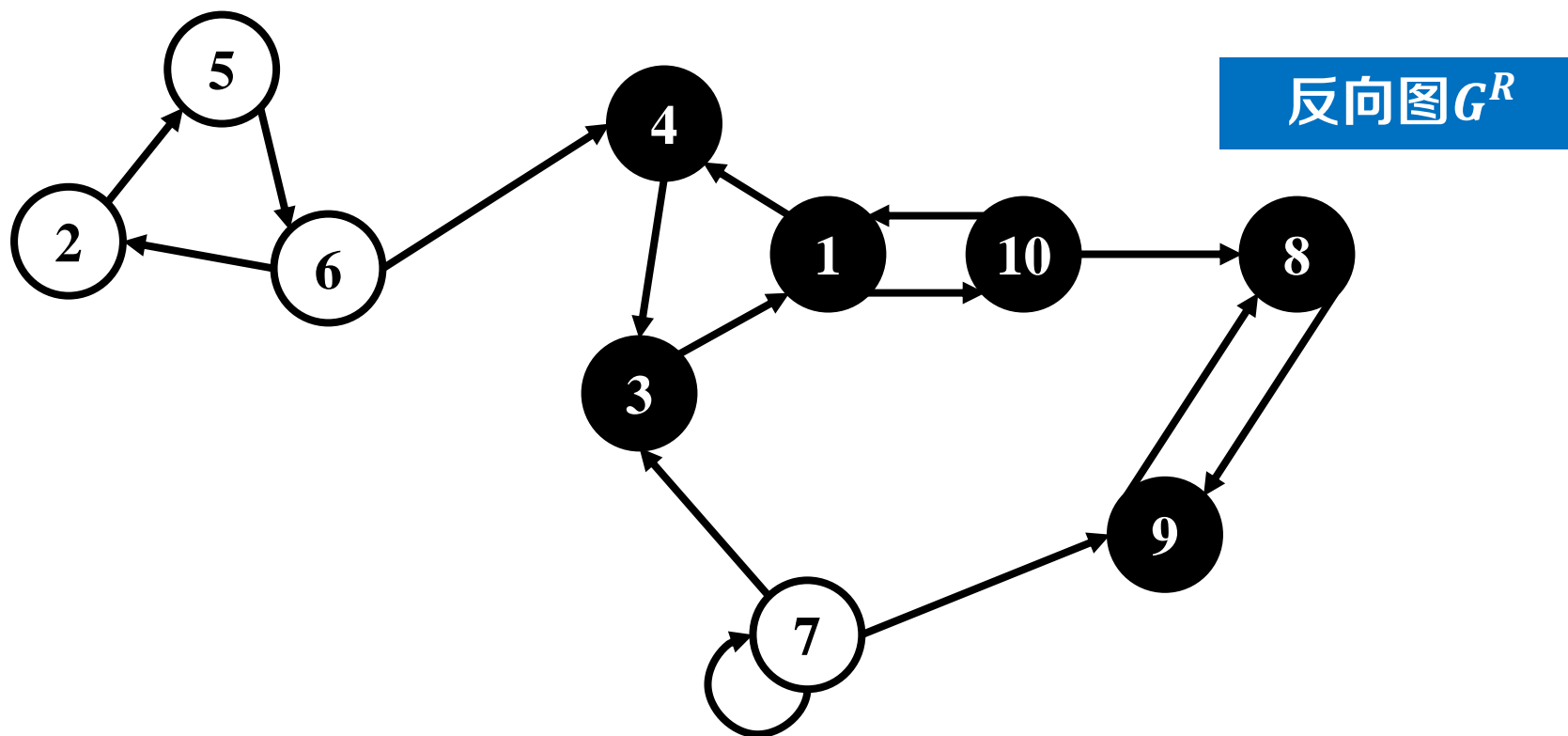


L	3	4	9	8	10					
-----	---	---	---	---	----	--	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

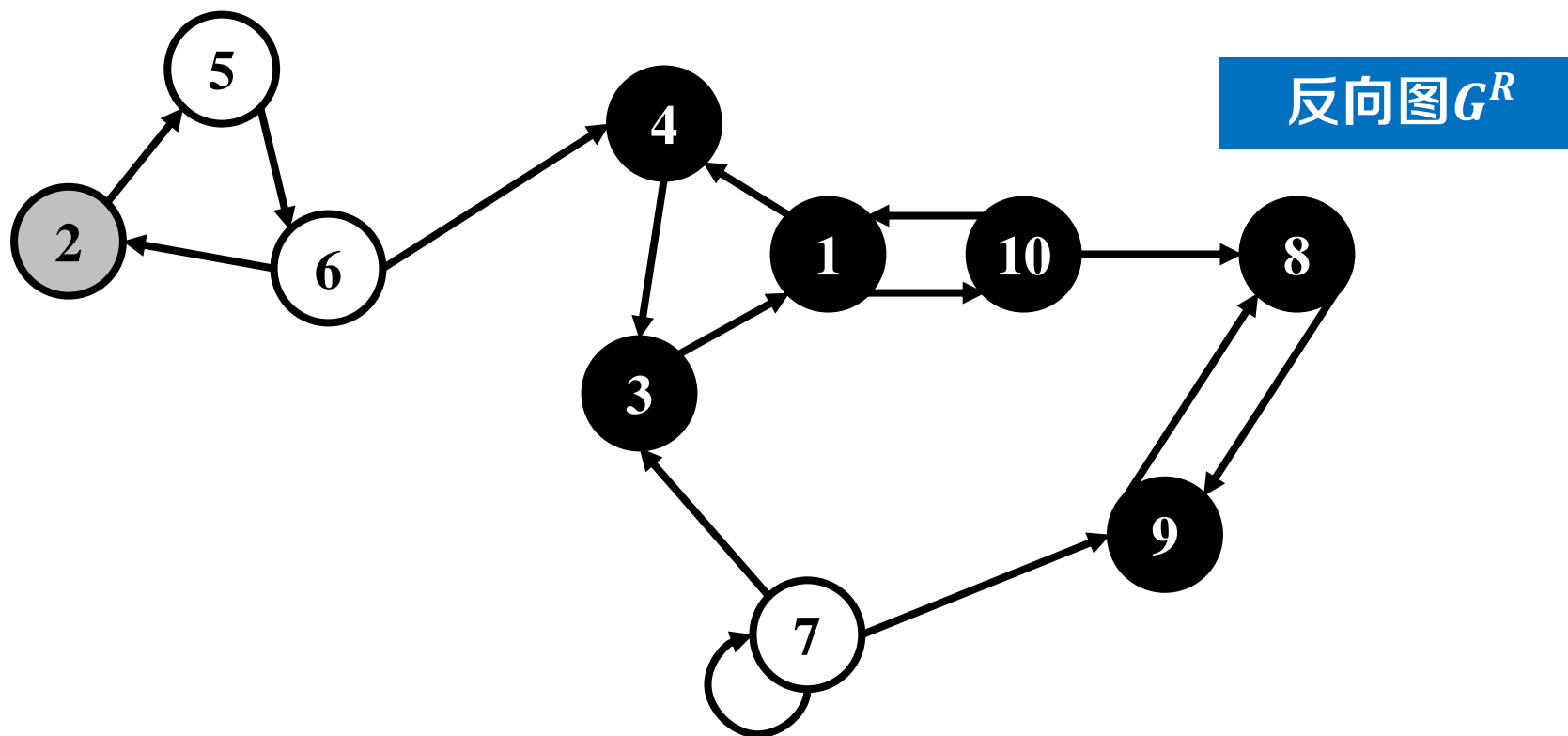


L	3	4	9	8	10	1				
-----	---	---	---	---	----	---	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

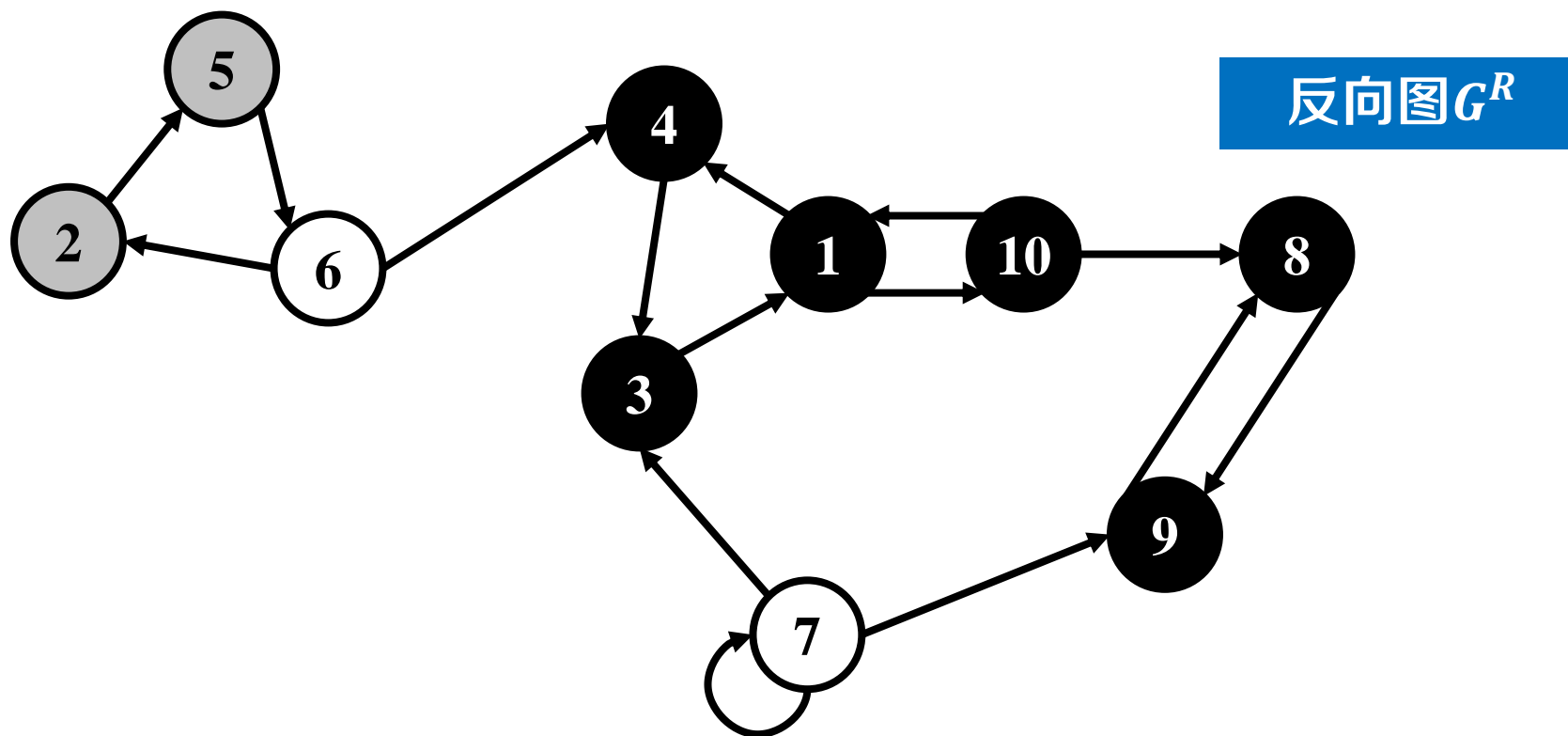


L	3	4	9	8	10	1				
-----	---	---	---	---	----	---	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

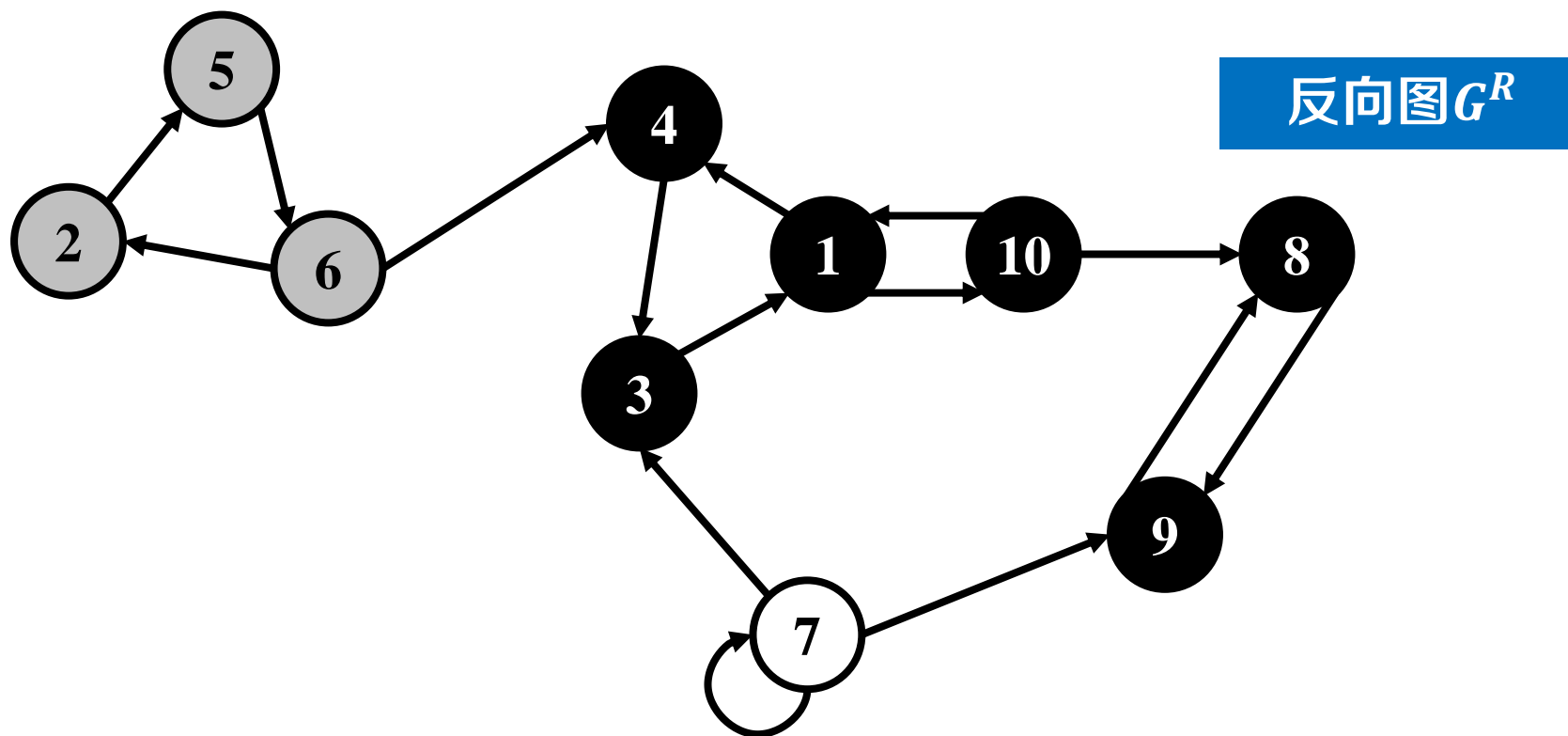


L	3	4	9	8	10	1				
-----	---	---	---	---	----	---	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

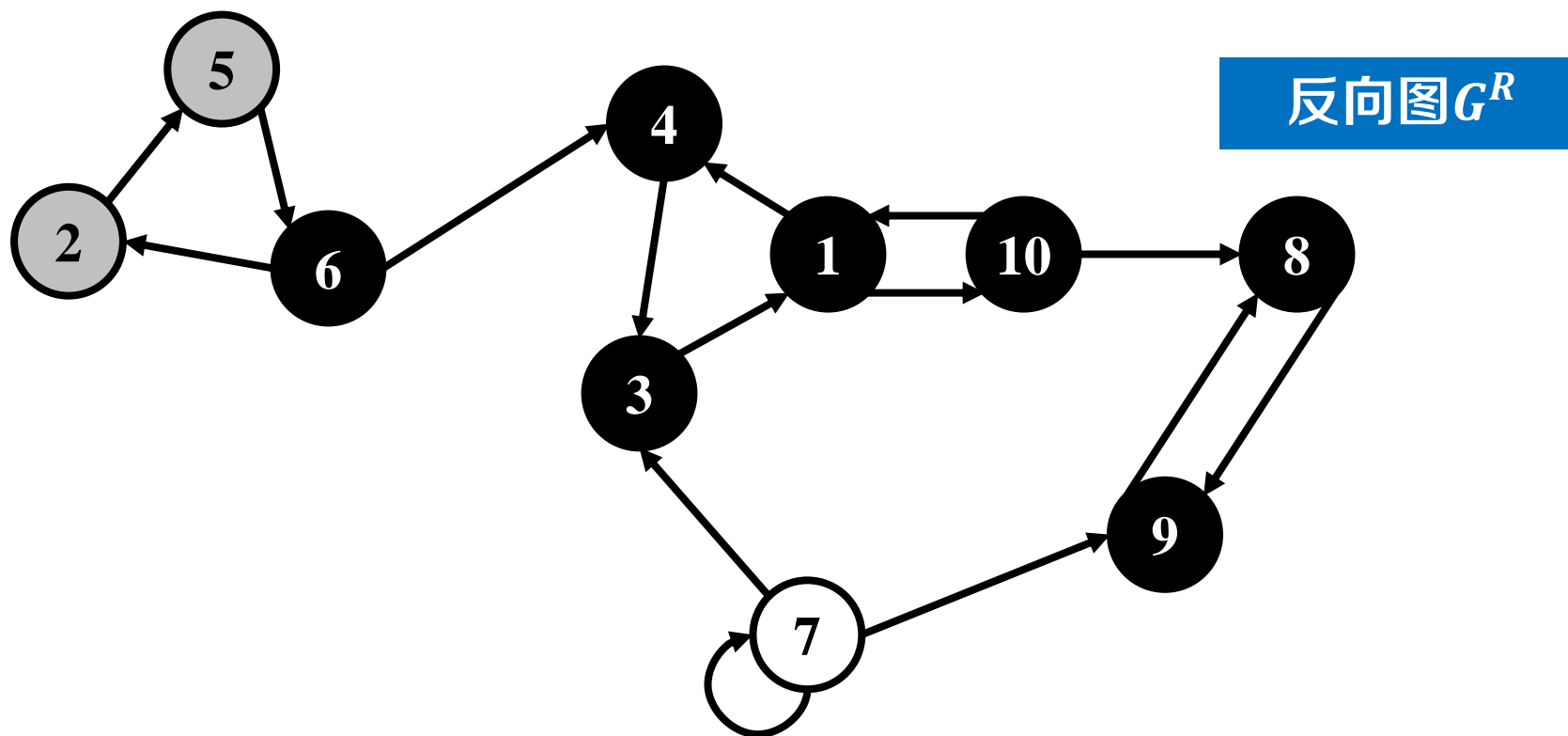


L	3	4	9	8	10	1				
-----	---	---	---	---	----	---	--	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

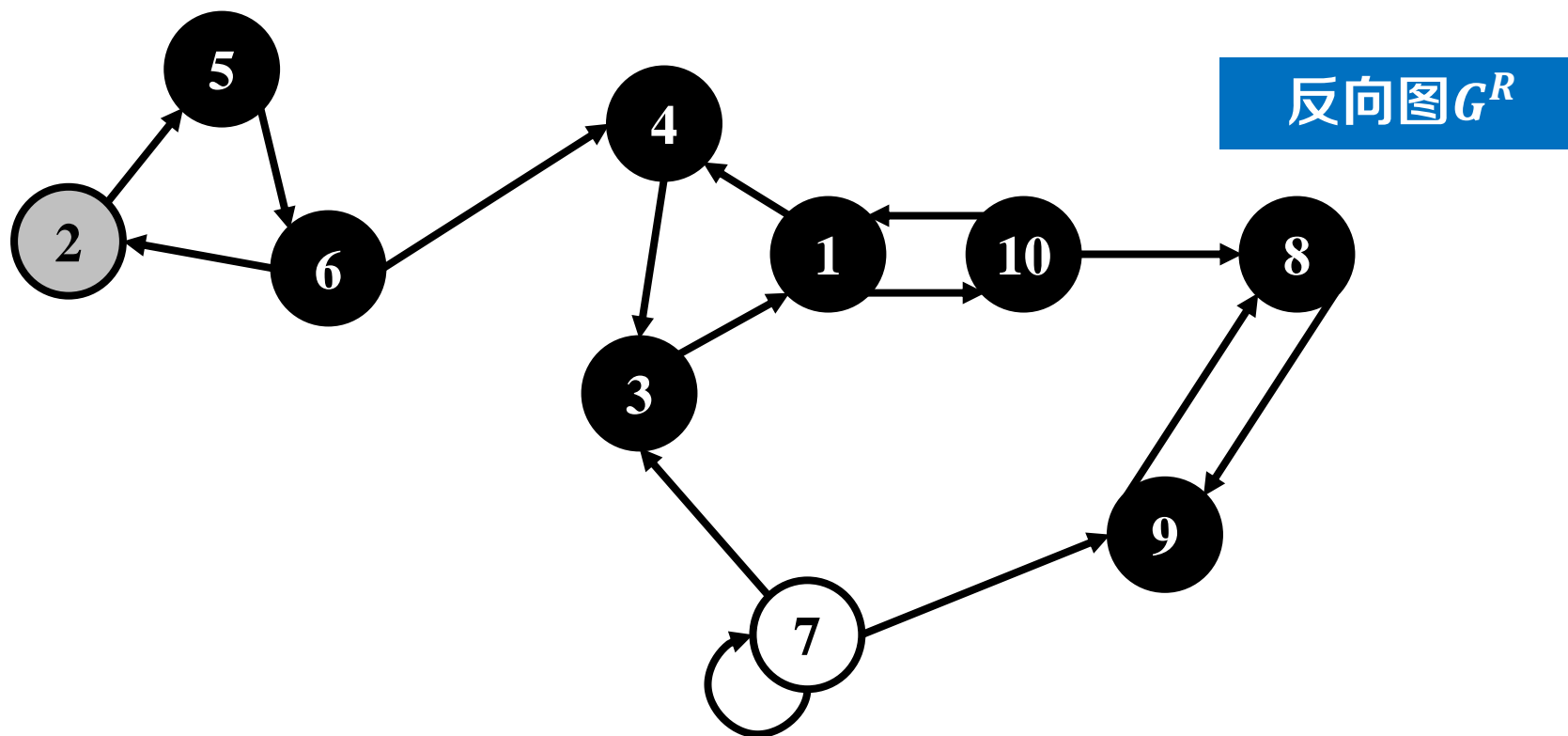


L	3	4	9	8	10	1	6			
-----	---	---	---	---	----	---	---	--	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

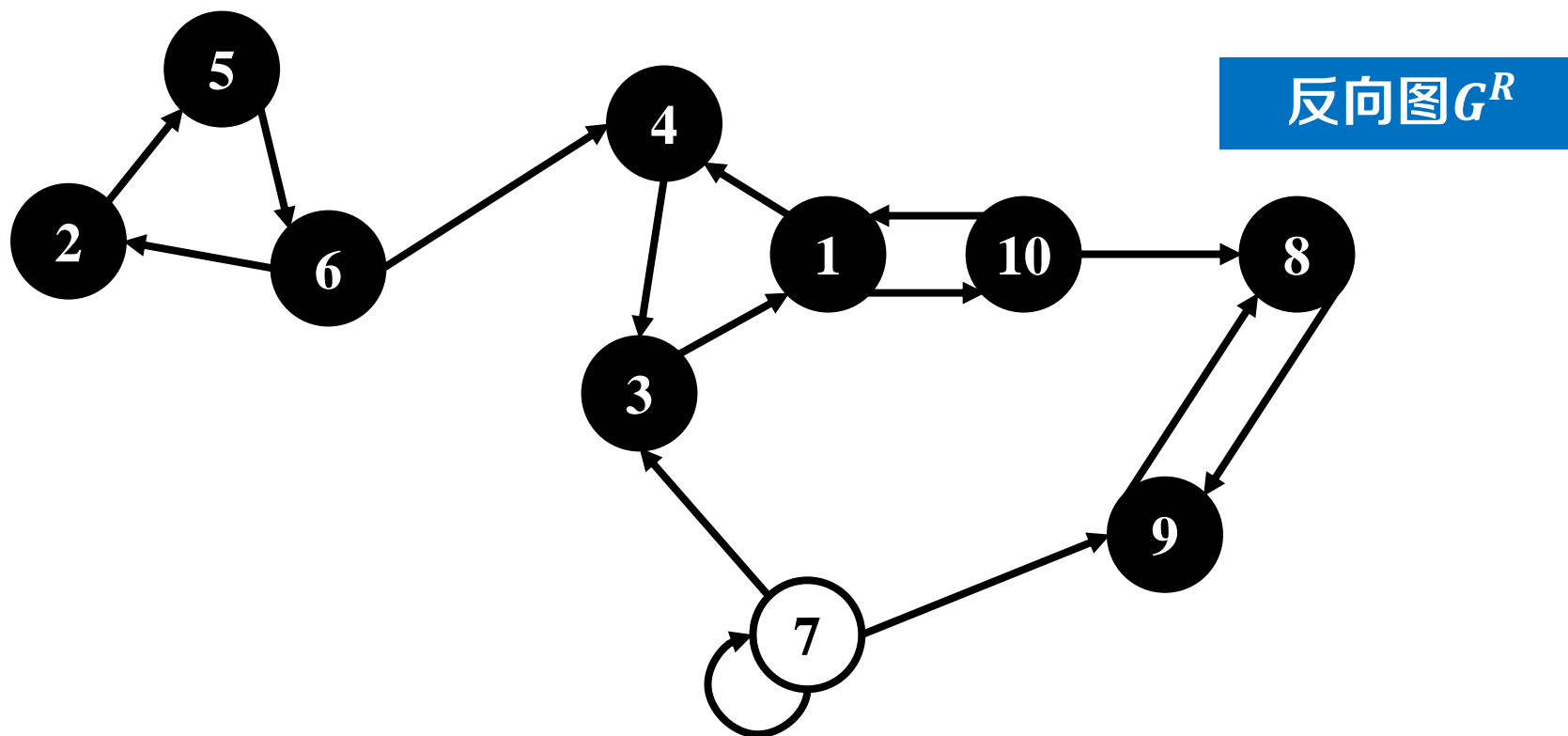


L	3	4	9	8	10	1	6	5		
-----	---	---	---	---	----	---	---	---	--	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

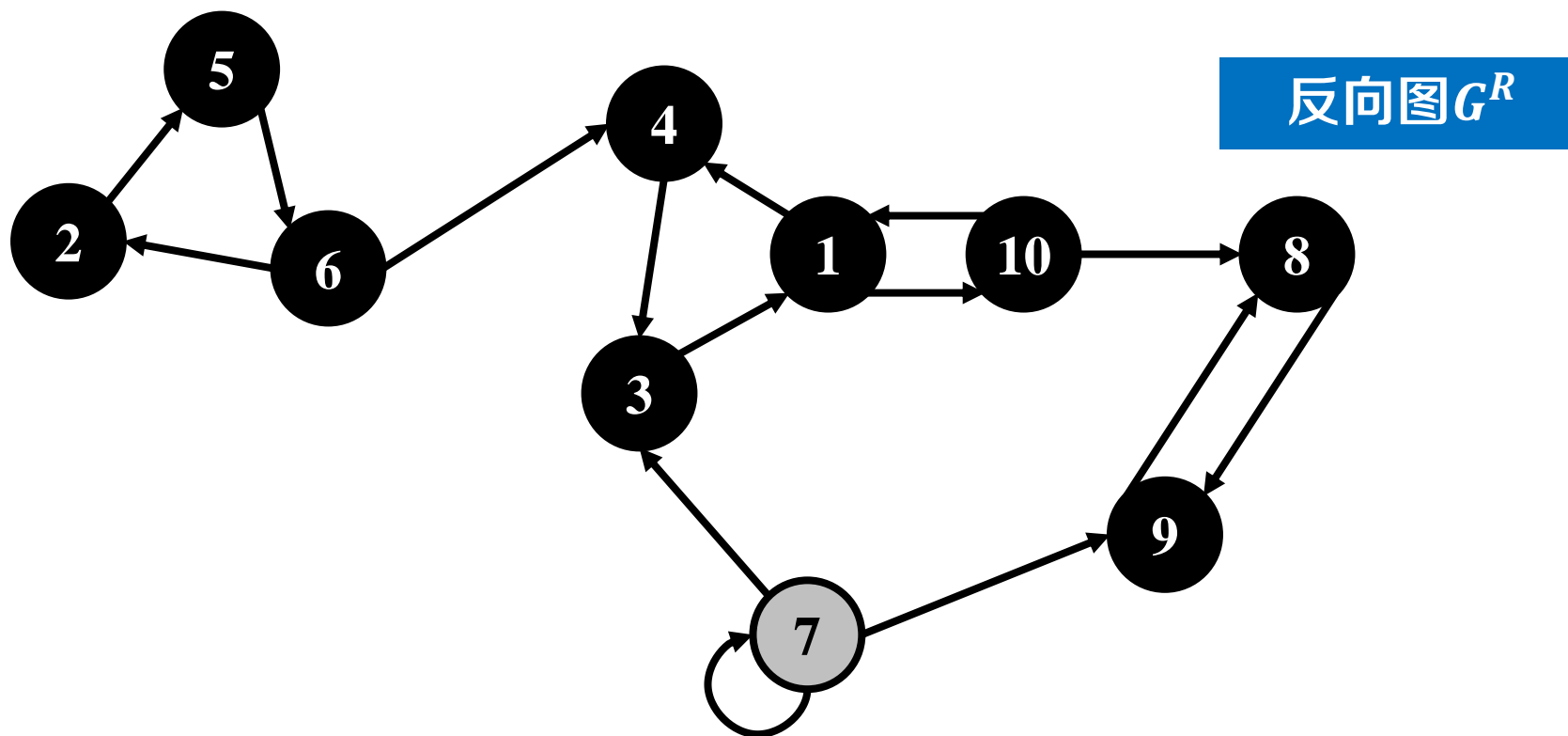


L	3	4	9	8	10	1	6	5	2	
-----	---	---	---	---	----	---	---	---	---	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

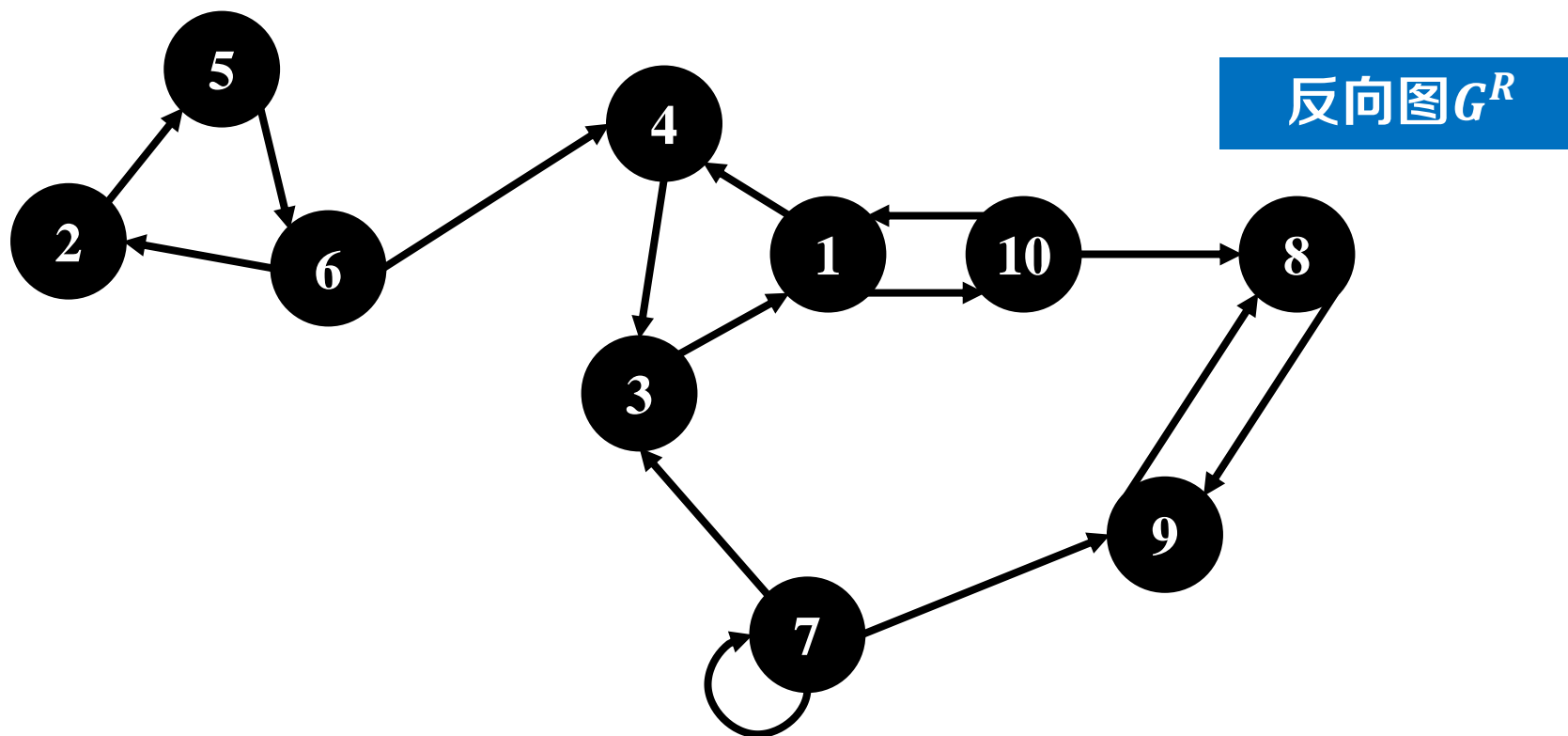


L	3	4	9	8	10	1	6	5	2	
-----	---	---	---	---	----	---	---	---	---	--

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L

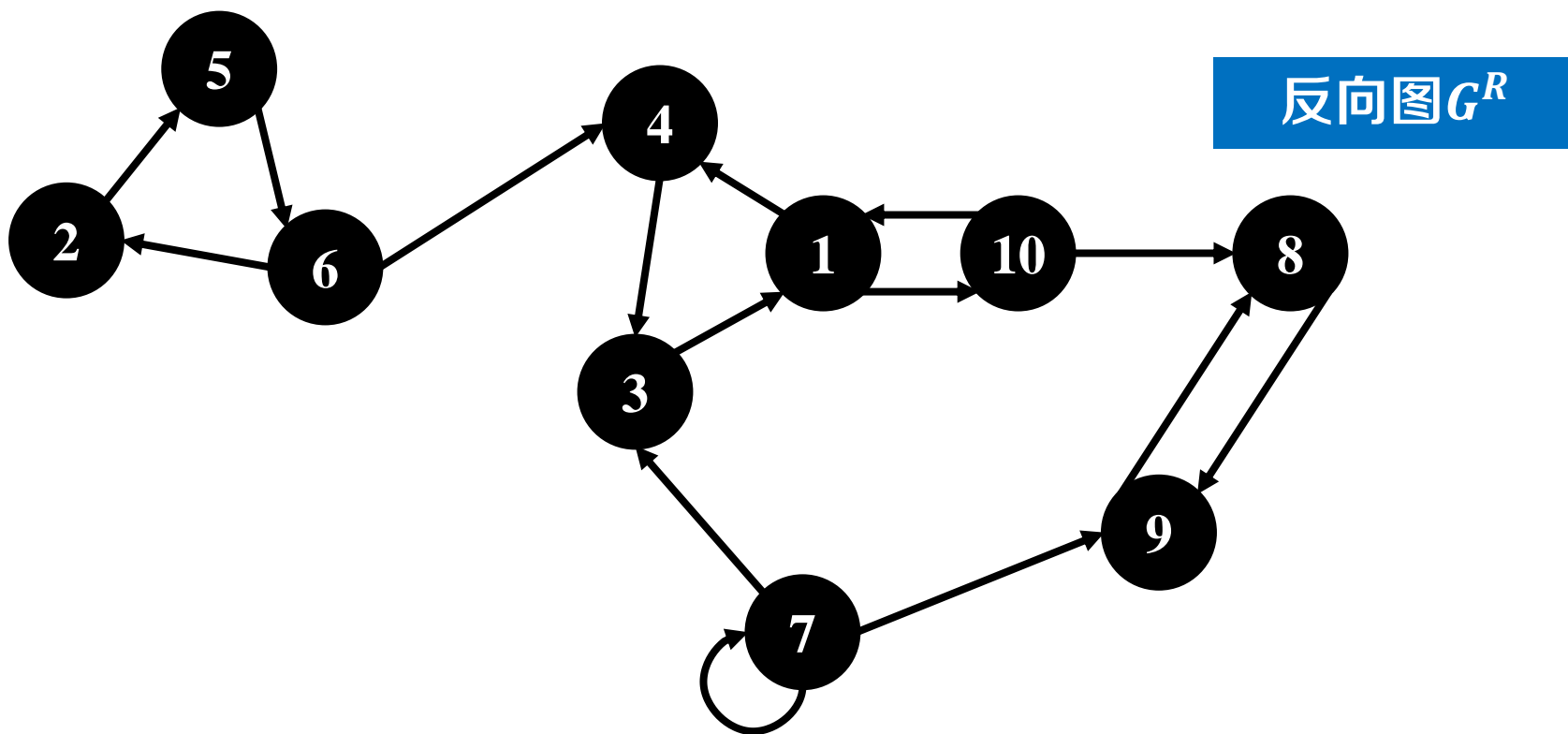


L	3	4	9	8	10	1	6	5	2	7
-----	---	---	---	---	----	---	---	---	---	---

算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量

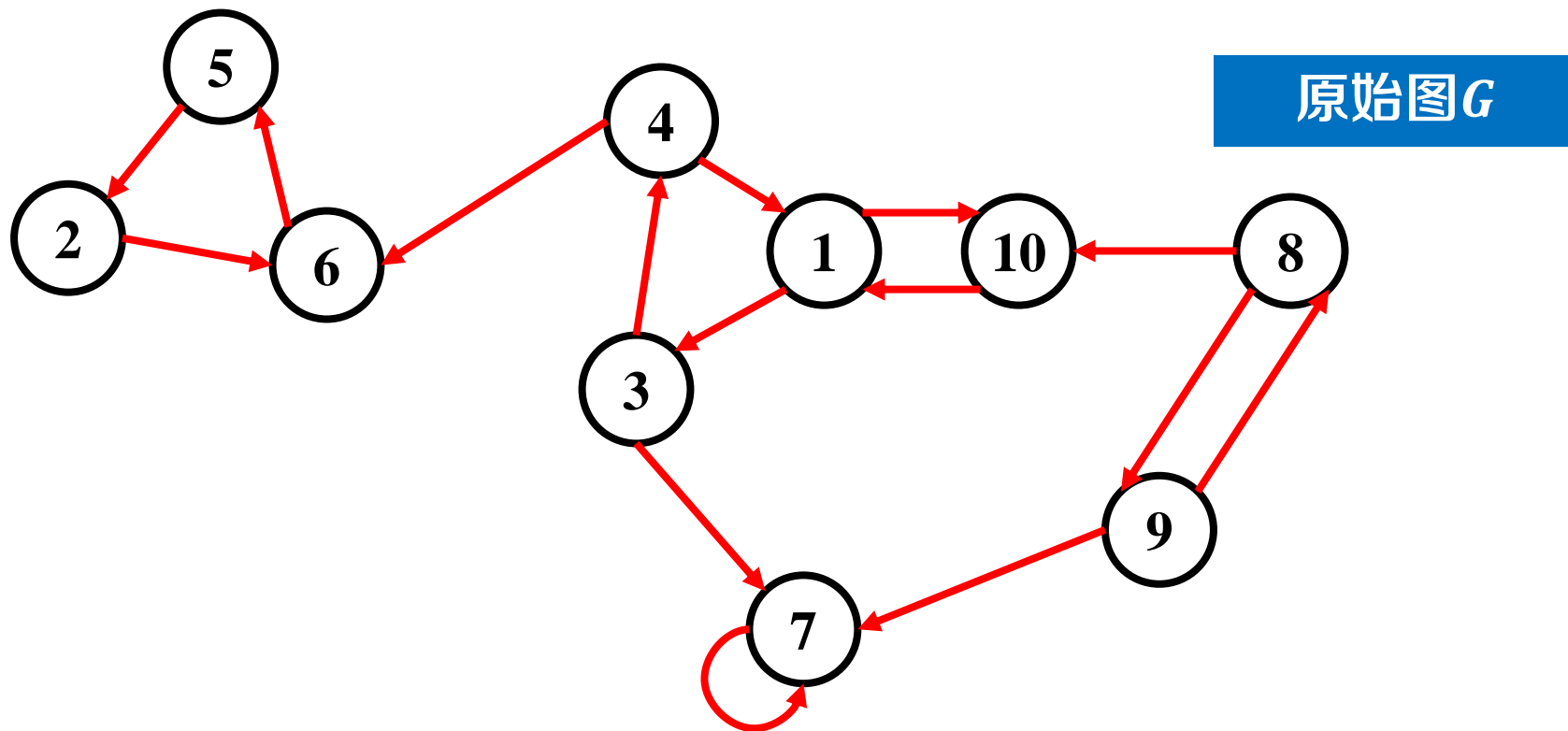


L	3	4	9	8	10	1	6	5	2	7
-----	---	---	---	---	----	---	---	---	---	---

算法框架与实例



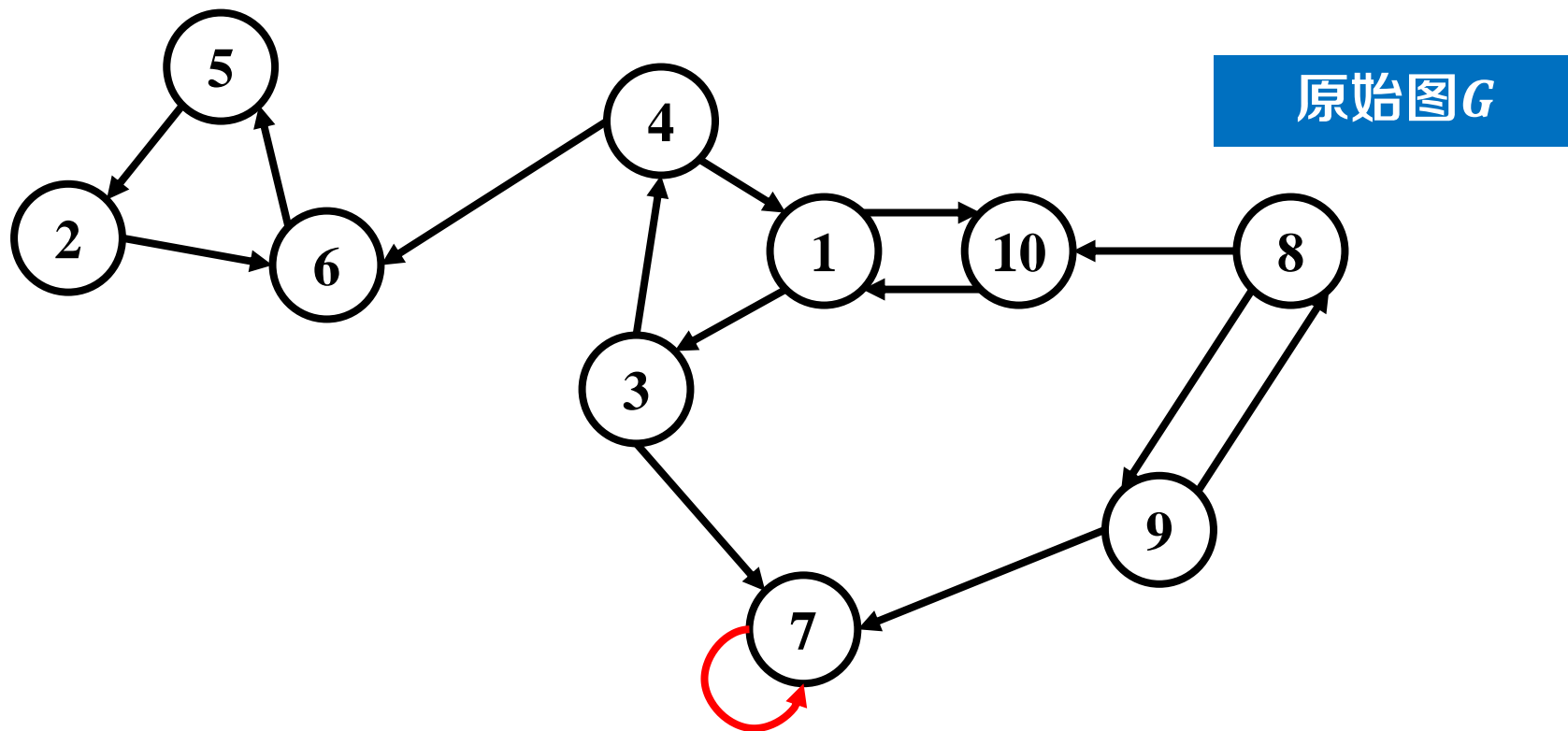
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



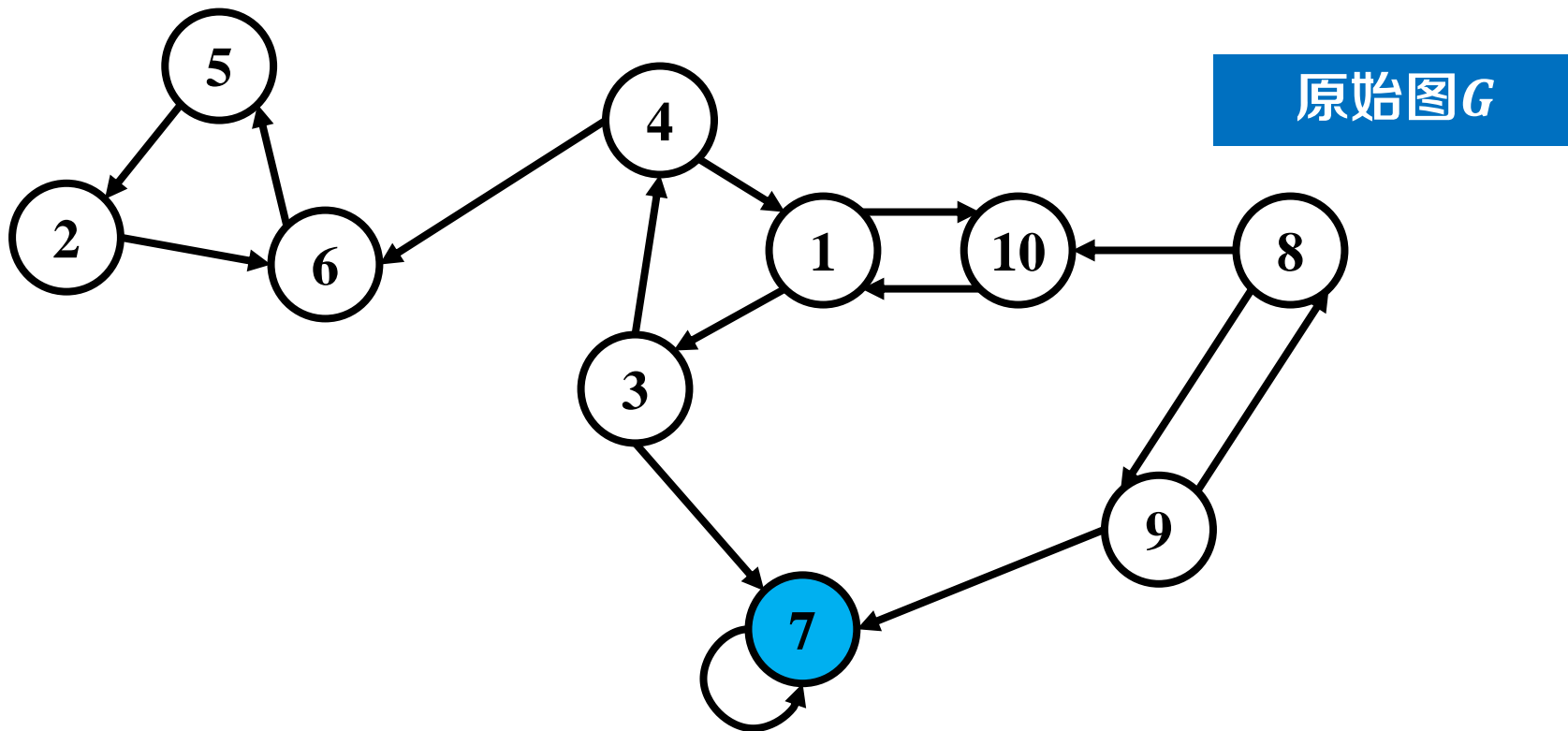
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



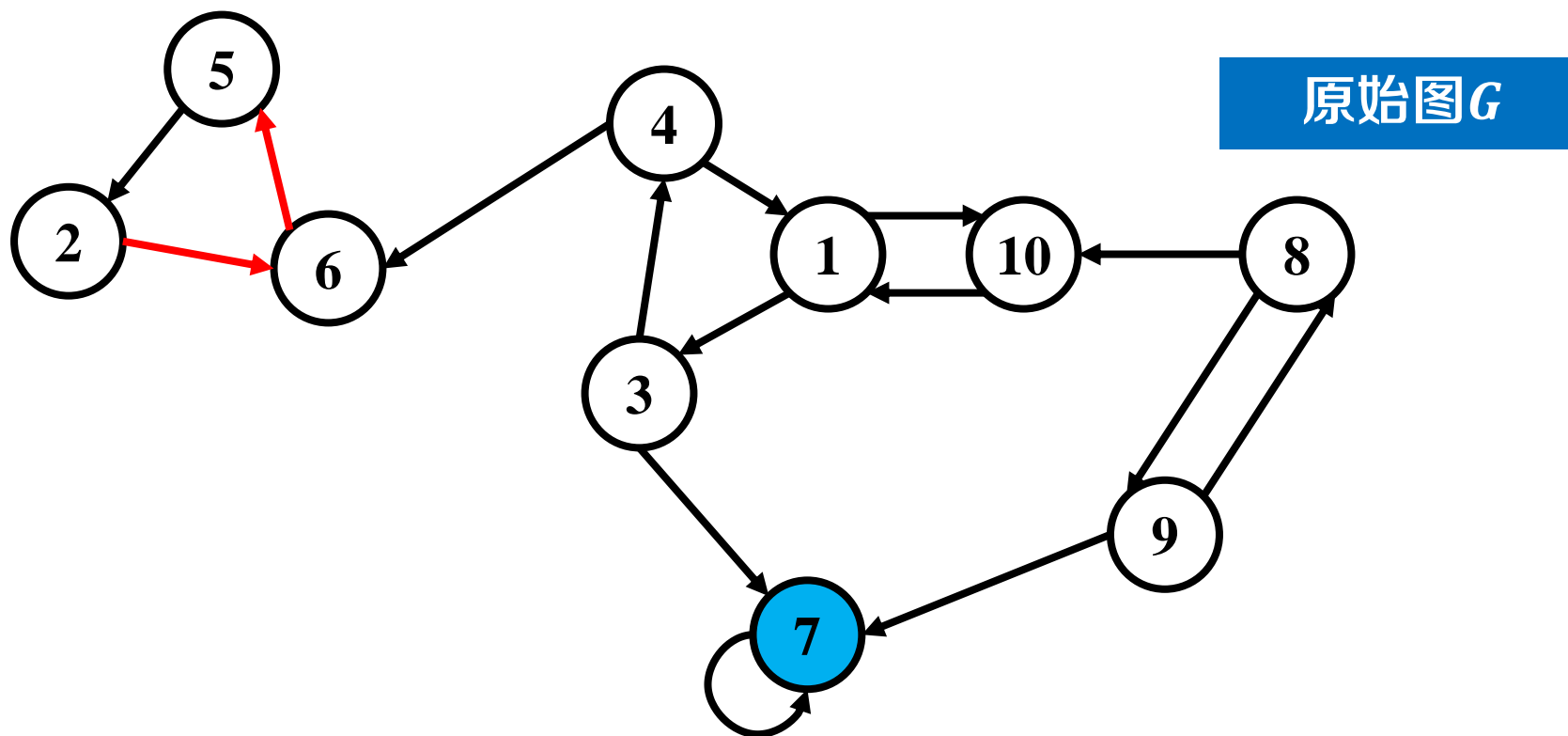
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



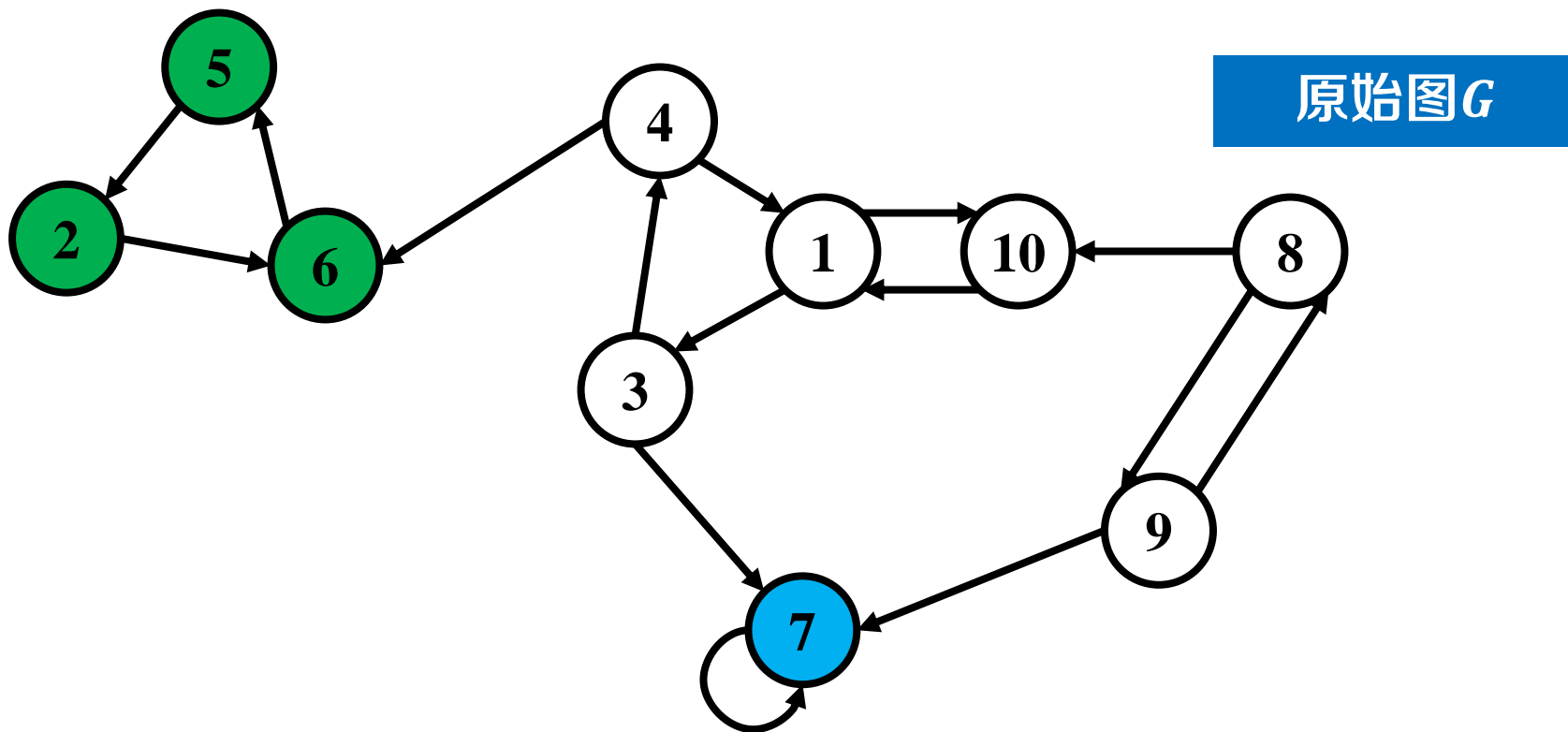
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



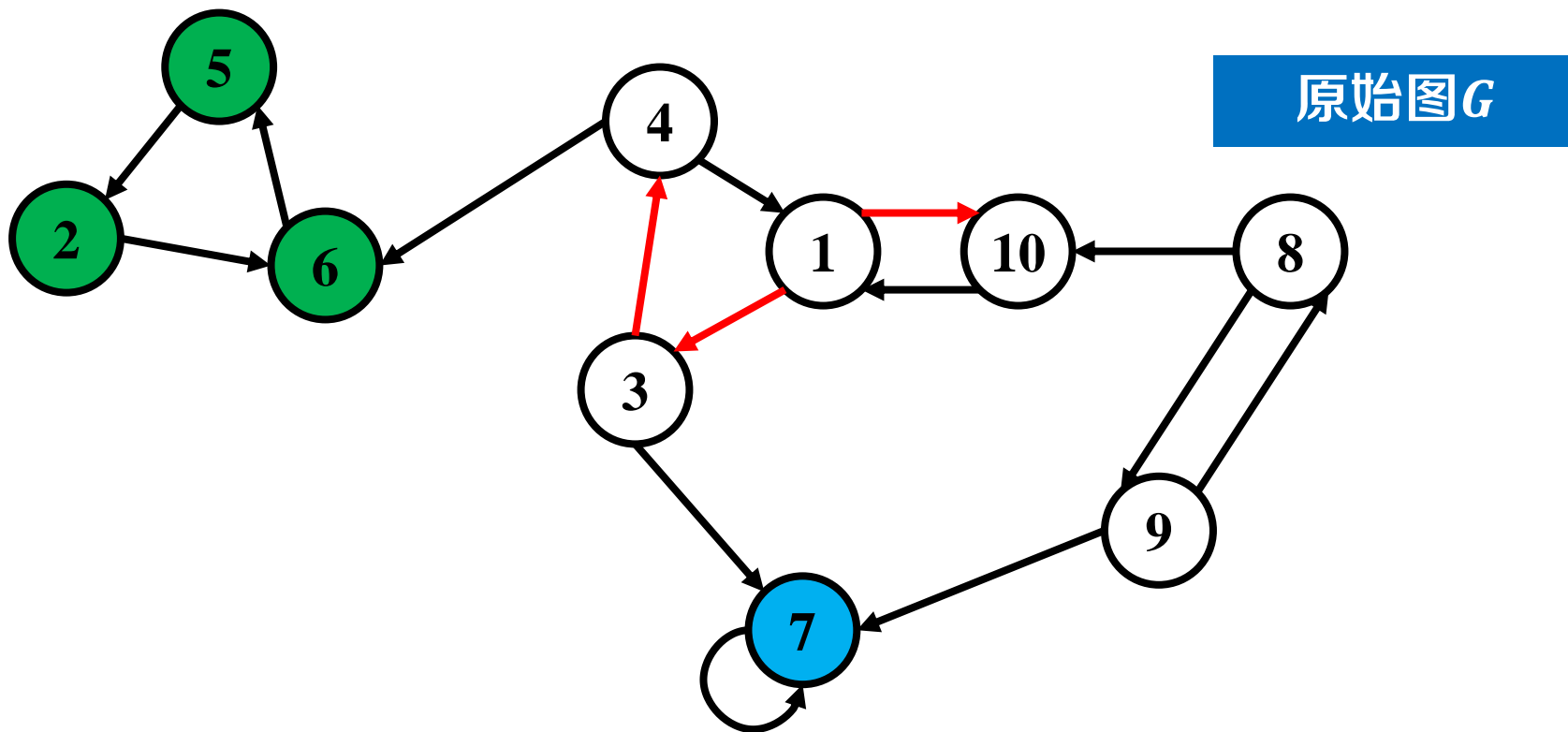
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



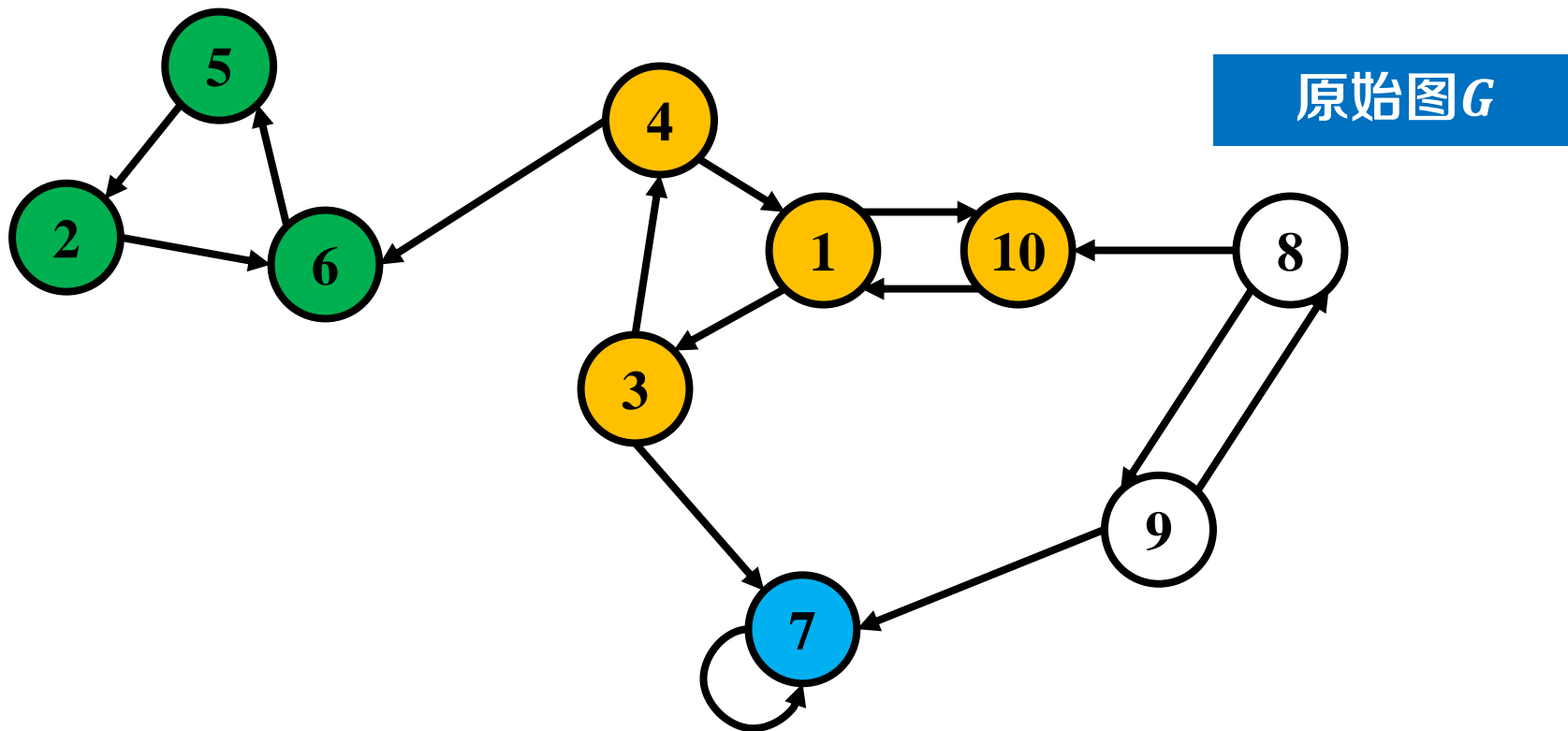
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



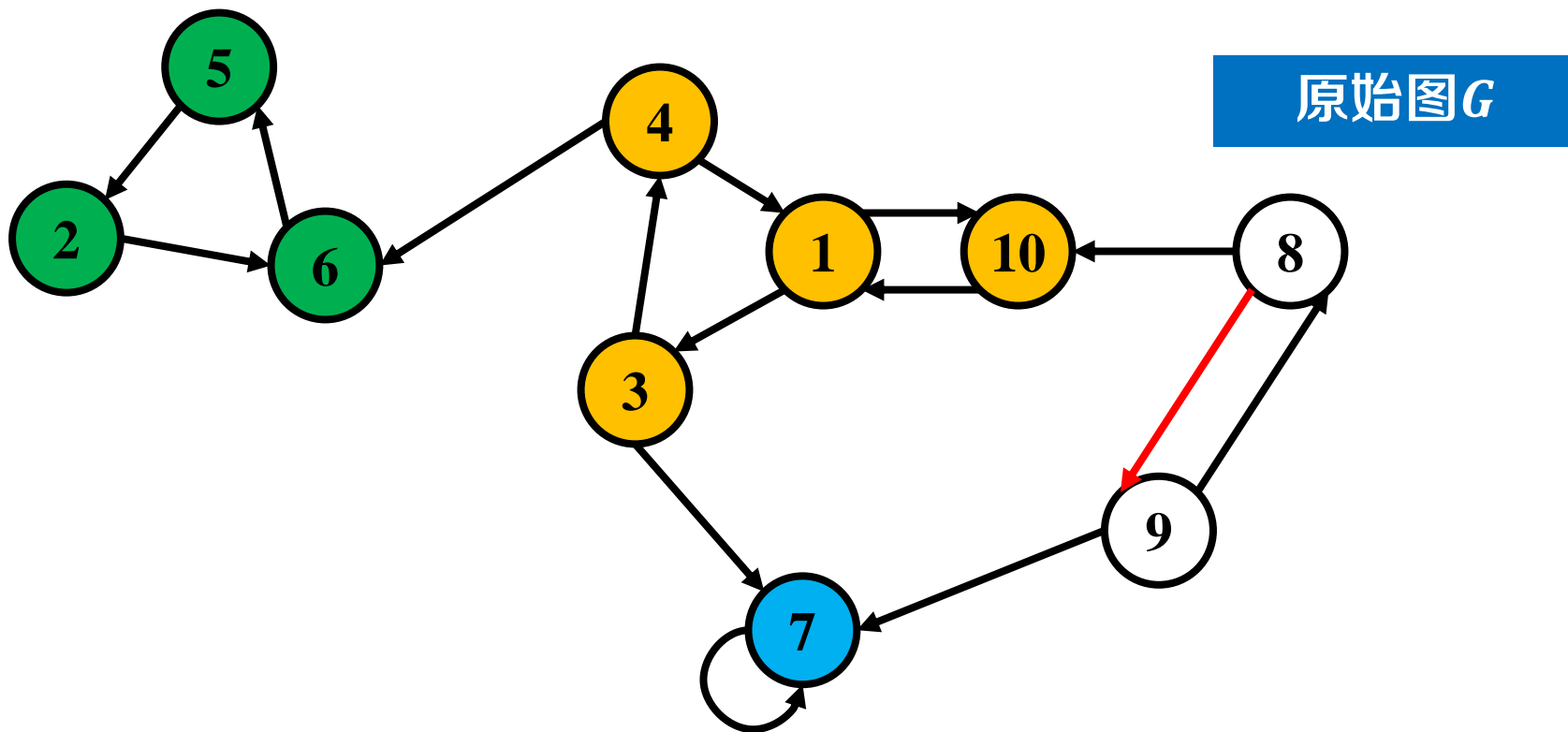
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



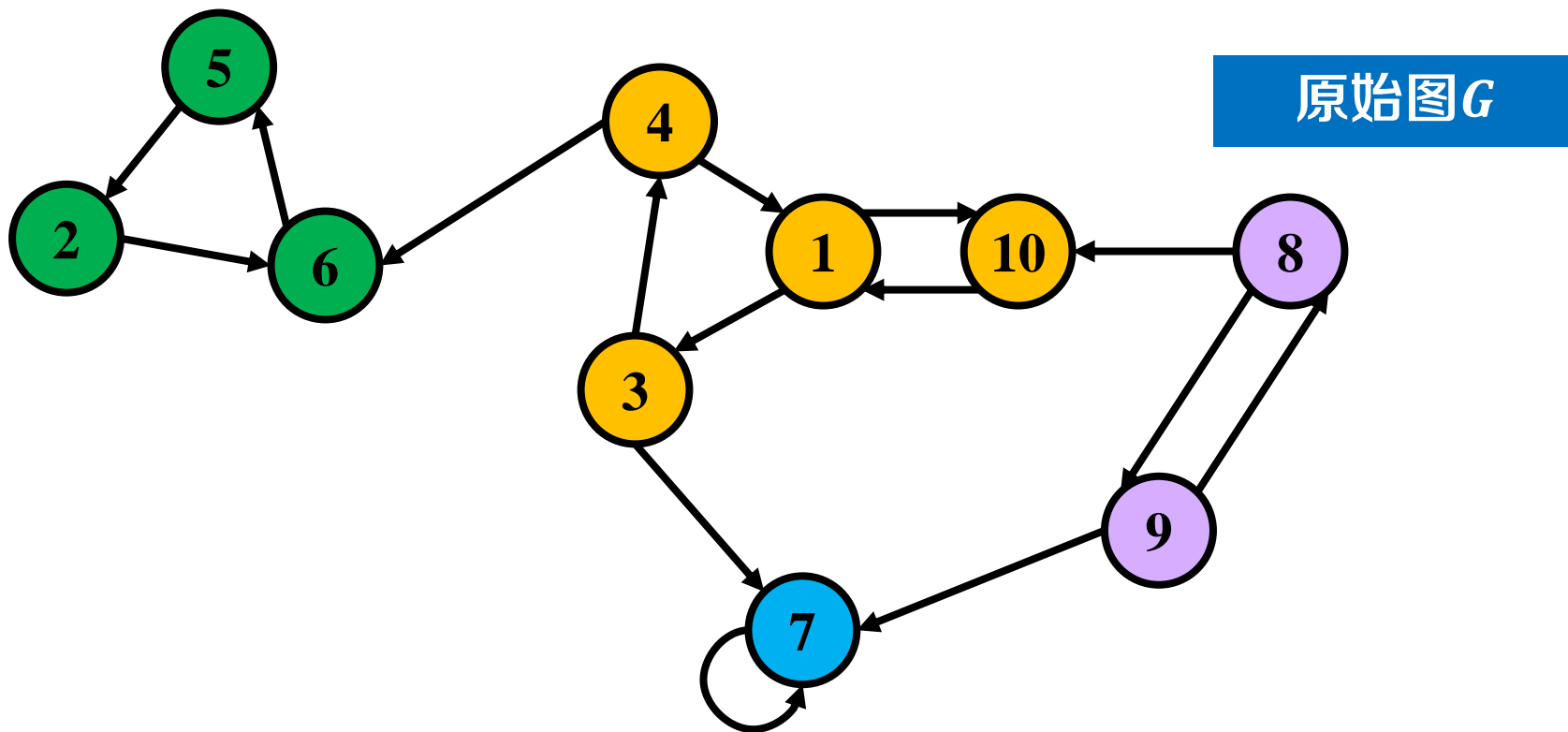
- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



算法框架与实例



- 步骤1: 把边反向, 得到反向图 G^R
- 步骤2: 在 G^R 上执行DFS, 得到顶点完成时刻顺序 L
- 步骤3: 在 G 上按 L 逆序执行DFS, 得到强连通分量



问题背景与定义

算法框架与实例

伪代码与复杂度

算法正确性证明

- Strongly-Connected-Component(G)

输入: 图 G

输出: 强连通分量

$R \leftarrow \{\}$

$G^R \leftarrow G.reverse()$

$L \leftarrow \text{DFS}(G^R)$

$color[1..V] \leftarrow WHITE$

for $i \leftarrow L.length()$ **downto** 1 **do**

$u \leftarrow L[i]$

if $color[u] = WHITE$ **then**

$L_{scc} \leftarrow \text{DFS-Visit}(G, u)$

$R \leftarrow R \cup set(L_{scc})$

end

end

return R

构造反向图

- Strongly-Connected-Component(G)

输入: 图 G

输出: 强连通分量

$R \leftarrow \{\}$

$G^R \leftarrow G.reverse()$

$L \leftarrow \text{DFS}(G^R)$

$color[1..V] \leftarrow WHITE$

for $i \leftarrow L.length()$ downto 1 do

$u \leftarrow L[i]$

 if $color[u] = WHITE$ then

$L_{scc} \leftarrow \text{DFS-Visit}(G, u)$

$R \leftarrow R \cup set(L_{scc})$

 end

end

return R

在反向图上执行DFS

- Strongly-Connected-Component(G)

输入: 图 G

输出: 强连通分量

$R \leftarrow \{\}$

$G^R \leftarrow G.reverse()$

$L \leftarrow \text{DFS}(G^R)$

$color[1..V] \leftarrow WHITE$

for $i \leftarrow L.length()$ downto 1 do

$u \leftarrow L[i]$

 if $color[u] = WHITE$ then

$L_{scc} \leftarrow \text{DFS-Visit}(G, u)$

$R \leftarrow R \cup set(L_{scc})$

 end

end

return R

按 L 逆序在原图执行DFS

- Strongly-Connected-Component(G)

输入: 图 G

输出: 强连通分量

$R \leftarrow \{\}$

$G^R \leftarrow G.reverse()$

$L \leftarrow \text{DFS}(G^R)$

$color[1..V] \leftarrow WHITE$

for $i \leftarrow L.length()$ downto 1 do

$u \leftarrow L[i]$

 if $color[u] = WHITE$ then

$L_{scc} \leftarrow \text{DFS-Visit}(G, u)$

$R \leftarrow R \cup set(L_{scc})$

 end

end

return R

如何在搜索过程中得到 L ?

- DFS(G)

```
输入: 图  $G$   
新建数组  $color[1..V], L[1..V]$   
for  $v \in V$  do  
    |  $color[v] \leftarrow WHITE$   
end  
for  $v \in V$  do  
    | if  $color[v] = WHITE$  then  
    |     |  $L' \leftarrow \text{DFS-Visit}(G, v)$   
    |     | 向  $L$  结尾追加  $L'$   
    |     end  
    end  
end  
return  $L$ 
```

- DFS-Visit(G, v)

```
输入: 图  $G$ , 顶点  $v$   
输出: 按完成时刻从早到晚排列的顶点  $L$   
 $color[v] \leftarrow GRAY$   
for  $w \in G.Adj[v]$  do  
    | if  $color[w] = WHITE$  then  
    |     |  $L \leftarrow \text{DFS-Visit}(G, w)$   
    |     end  
    end  
end  
 $color[v] \leftarrow BLACK$   
向  $L$  结尾追加顶点  $v$   
return  $L$ 
```

顶点按
完成时
刻排列

- Strongly-Connected-Component(G)

输入: 图 G

输出: 强连通分量

$R \leftarrow \{\}$

$G^R \leftarrow G.reverse()$ ----- $O(|V| + |E|)$

$L \leftarrow \text{DFS}(G^R)$ ----- $O(|V| + |E|)$

第一次深度优先搜索

$color[1..V] \leftarrow WHITE$

for $i \leftarrow L.length()$ downto 1 do

$u \leftarrow L[i]$

 if $color[u] = WHITE$ then

$L_{scc} \leftarrow \text{DFS-Visit}(G, u)$

$R \leftarrow R \cup set(L_{scc})$

 end

end

return R

} $O(|V| + |E|)$ 第二次深度优先搜索

时间复杂度: $O(|V| + |E|)$

问题背景与定义

算法框架与实例

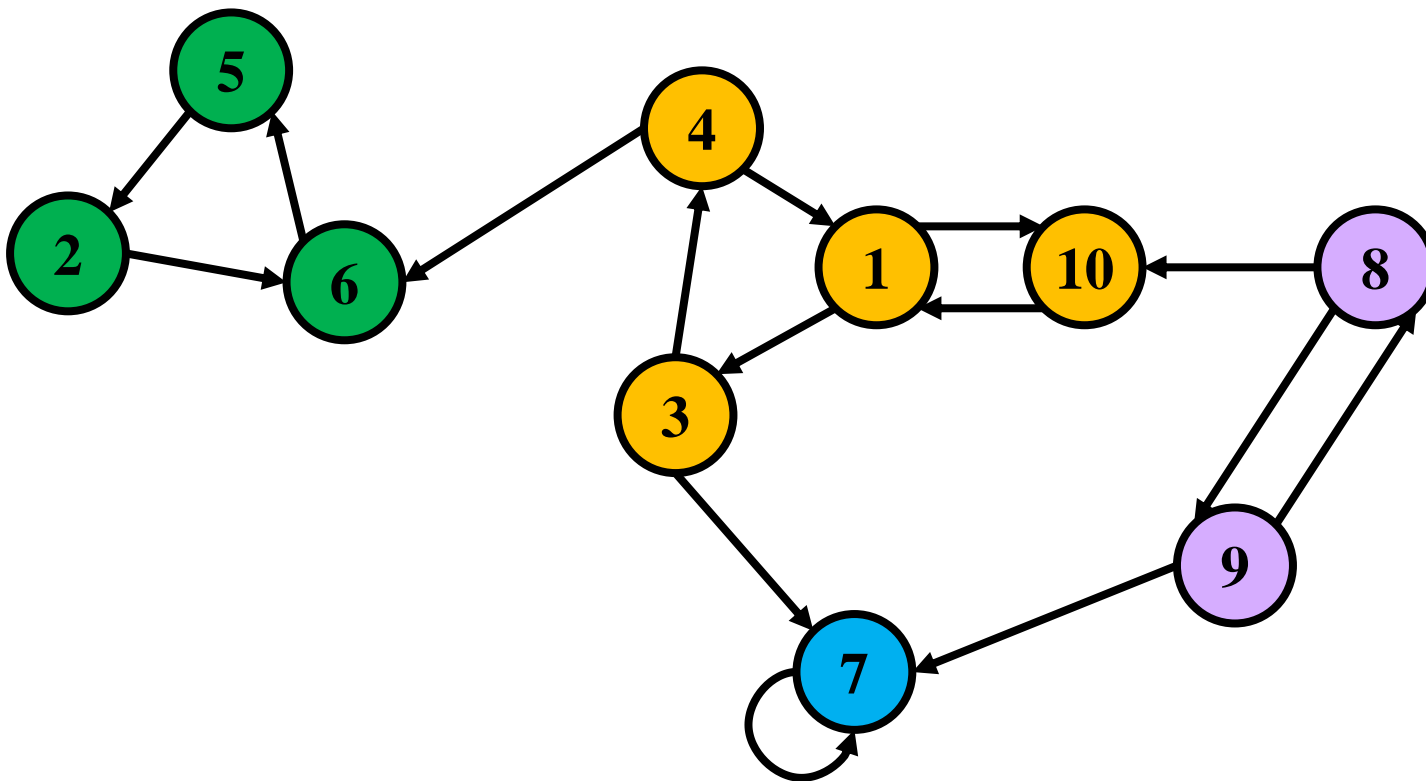
伪代码与复杂度

算法正确性证明

正确性证明



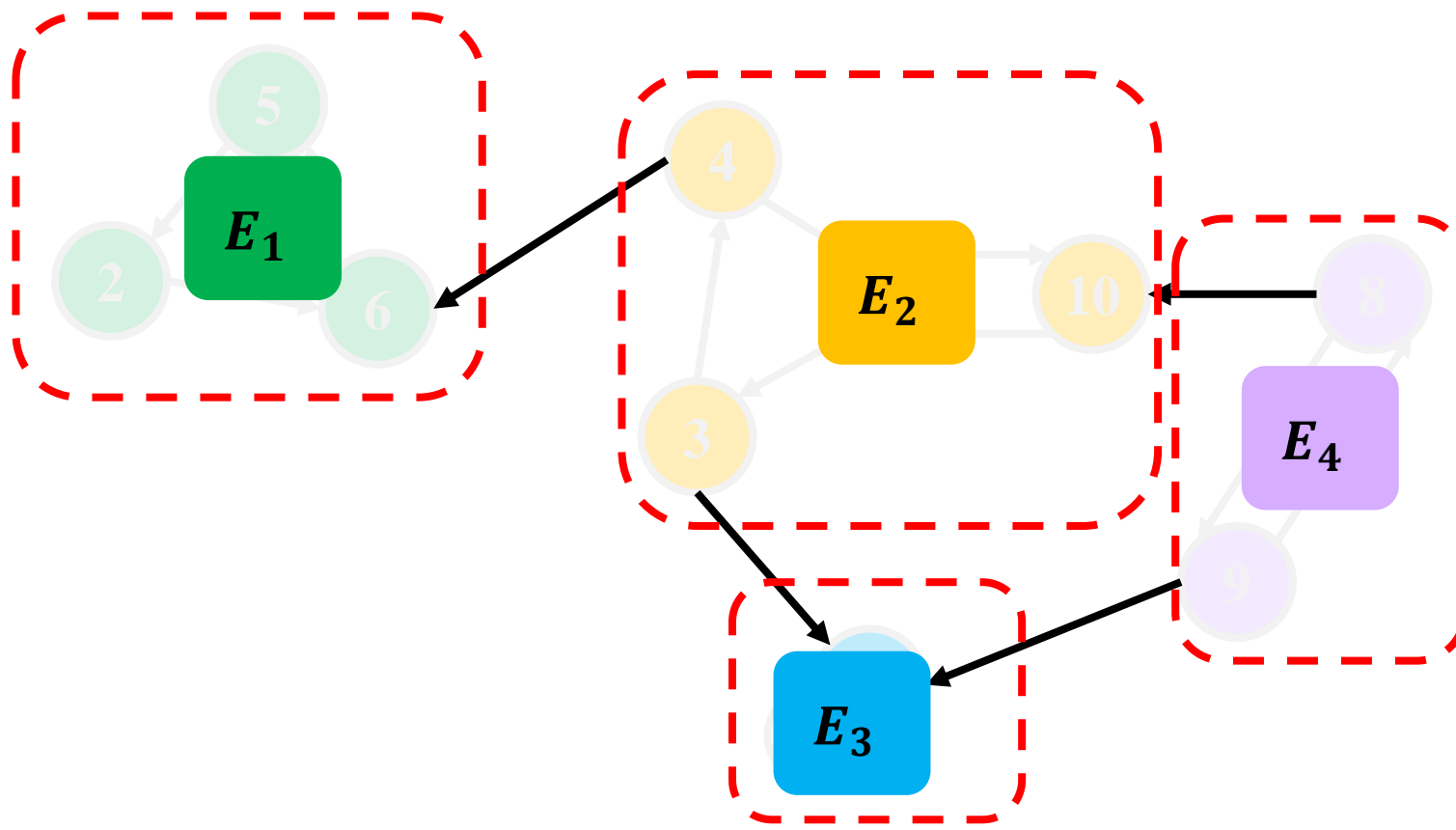
- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图



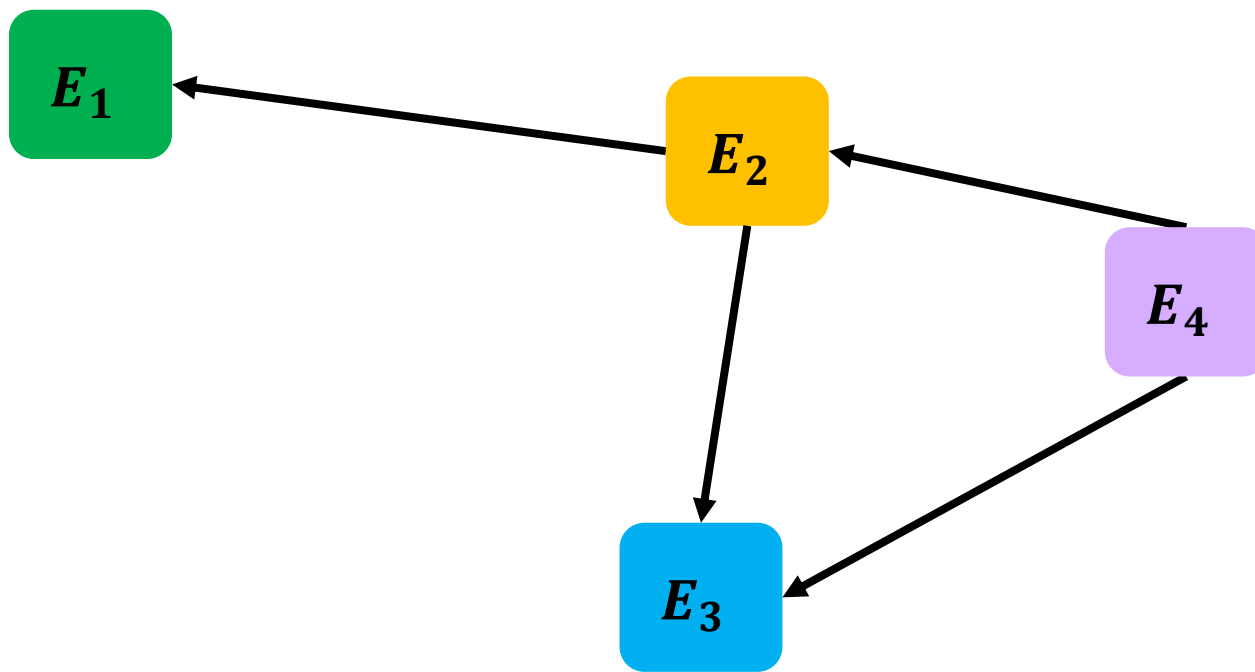
正确性证明



- 强连通分量图 G^{SCC} : 把强连通分量看作一个点，得到有向图



- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图

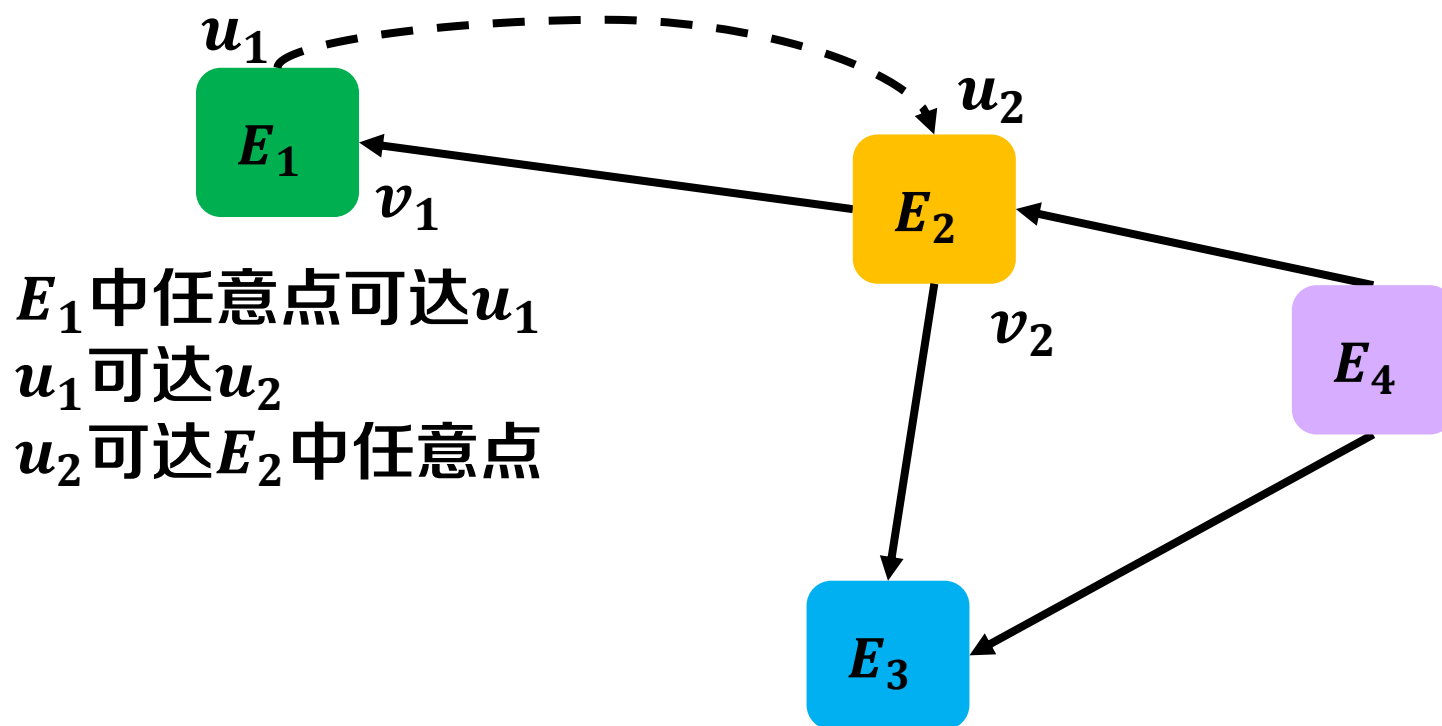


正确性证明



- 强连通分量图 G^{SCC} ：把强连通分量看作一个点，得到有向图
 - 性质： G^{SCC} 一定是有向无环图
 - 反证：若存在环，两强连通分量中顶点相互可达，与**最大性**矛盾。

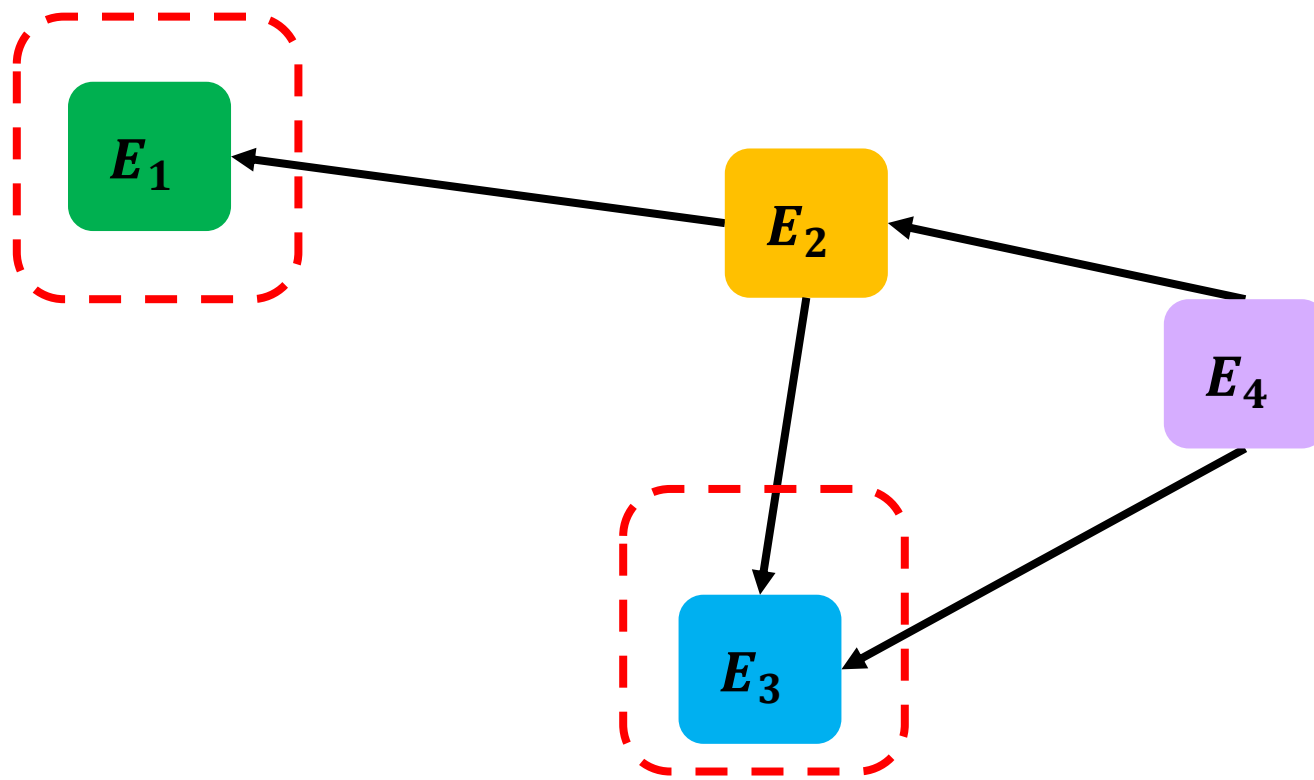
最大性：加入新顶点，不保证相互可达



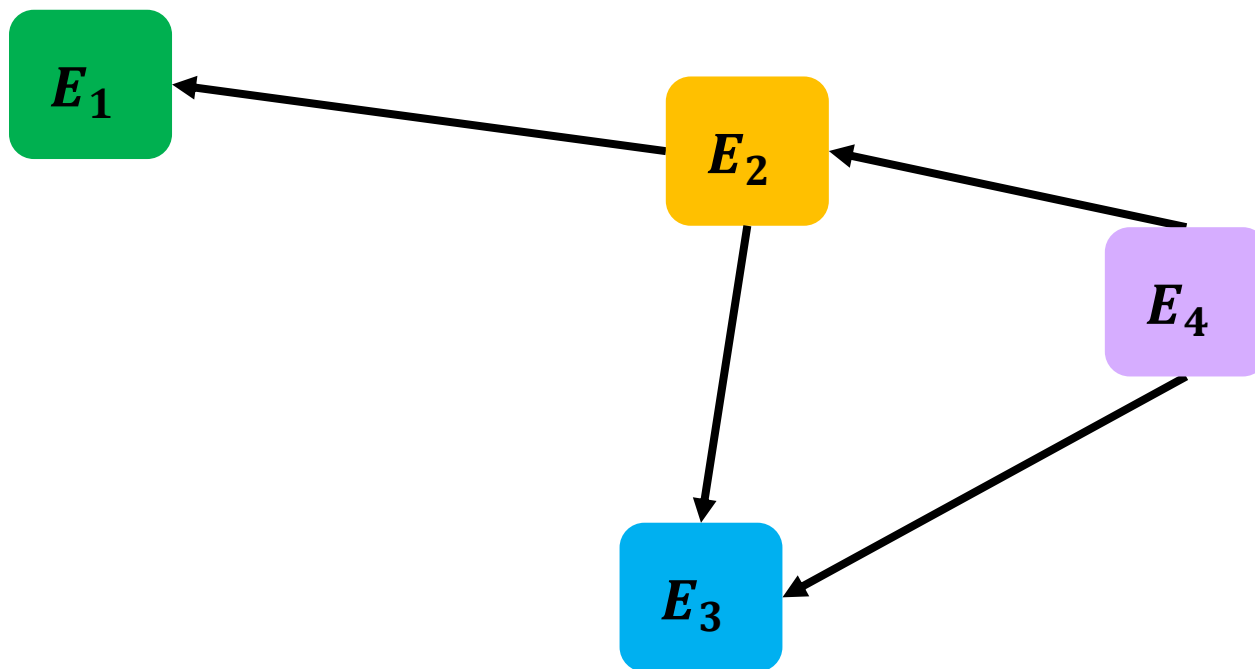
正确性证明



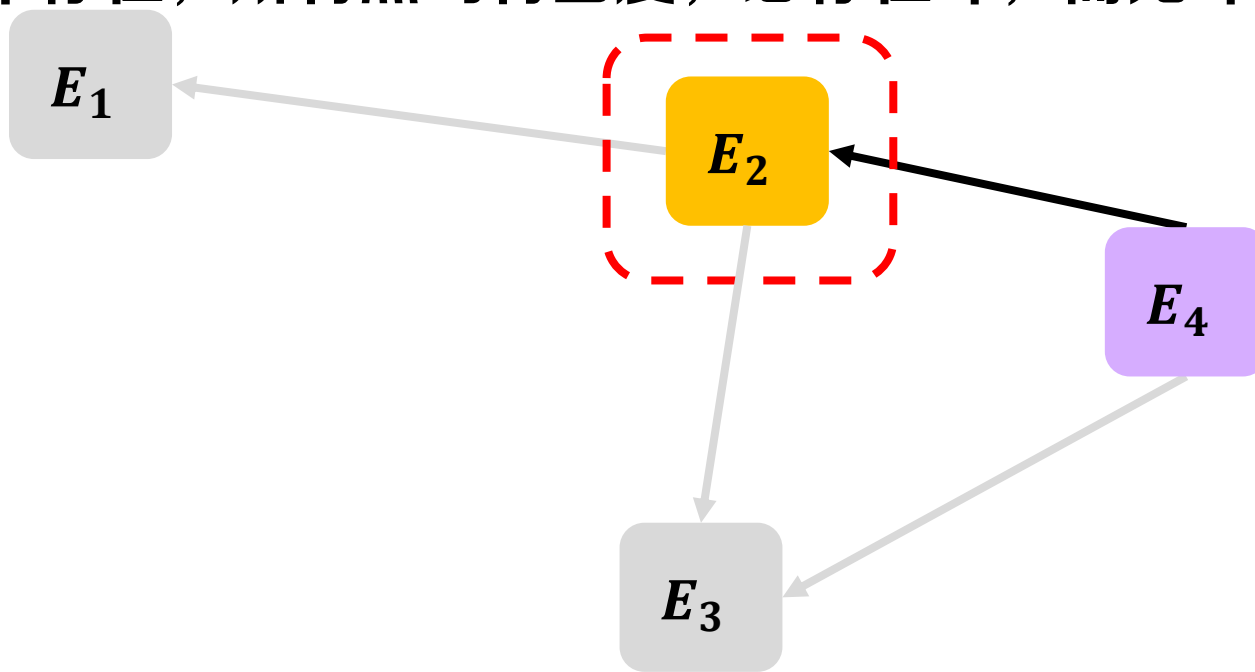
- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点



- 强连通分量图 G^{SCC} ：把强连通分量看作一个点，得到有向图
 - 性质： G^{SCC} 一定是有向无环图
- SCC_{Sink} ： G^{SCC} 中出度为0的点
 - 性质1： G^{SCC} 中存在至少一个 SCC_{Sink}
 - 反证：若不存在，所有点均有出度，必存在环，矛盾



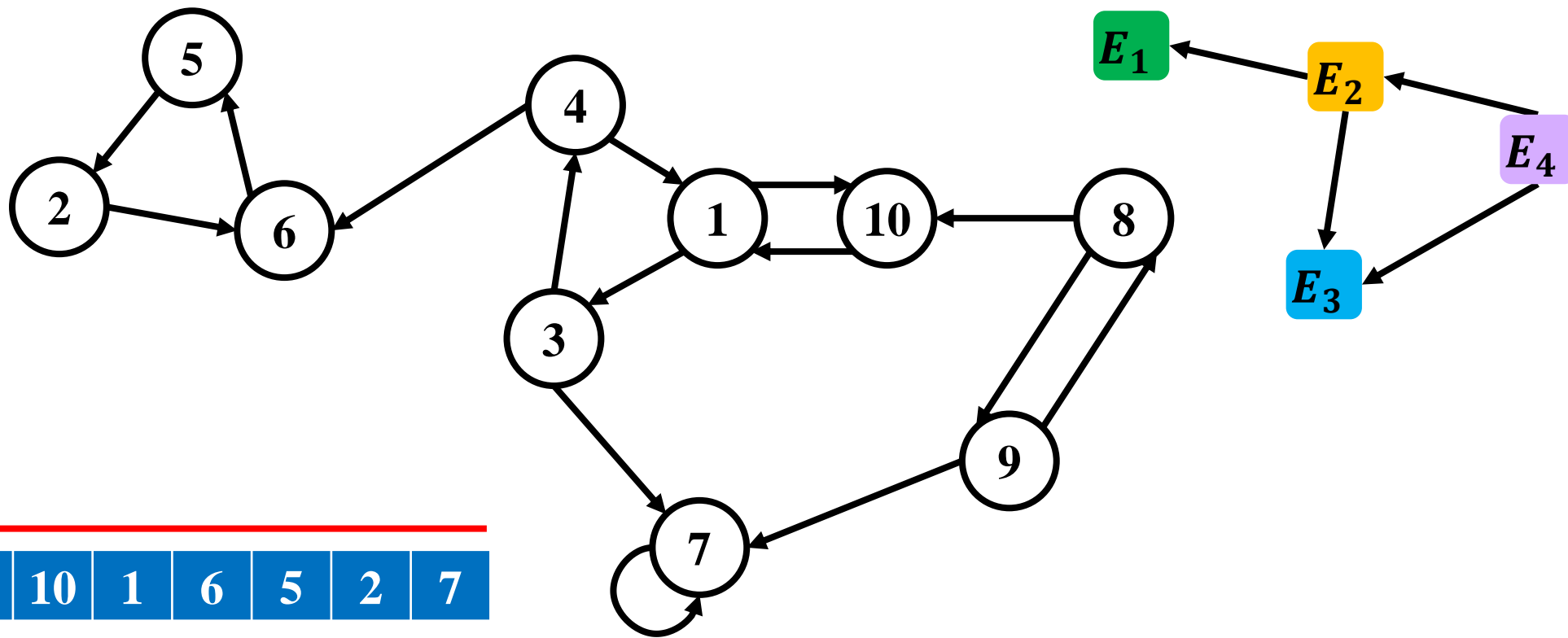
- 强连通分量图 G^{SCC} ：把强连通分量看作一个点，得到有向图
 - 性质： G^{SCC} 一定是有向无环图
- SCC_{Sink} ： G^{SCC} 中出度为0的点
 - 性质1： G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2： 删除 SCC_{Sink} ，会产生新的 SCC_{Sink}
 - 反证： 若不存在，所有点均有出度，必存在环；而无环图子图必无环，矛盾



正确性证明

- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点
 - 性质1: G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2: 删除 SCC_{Sink} , 会产生新的 SCC_{Sink}

结合性质观察
算法第2次DFS

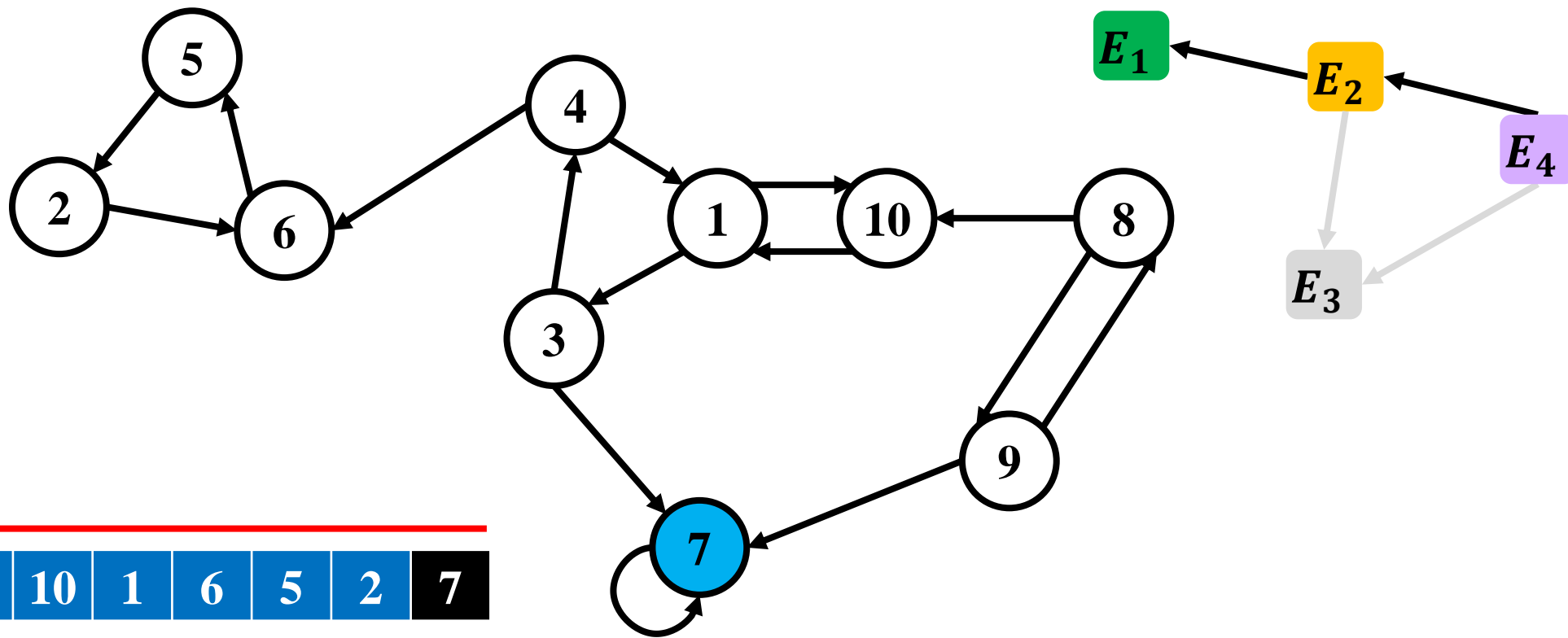


L	3	4	9	8	10	1	6	5	2	7
-----	---	---	---	---	----	---	---	---	---	---

正确性证明

- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点
 - 性质1: G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2: 删除 SCC_{Sink} , 会产生新的 SCC_{Sink}

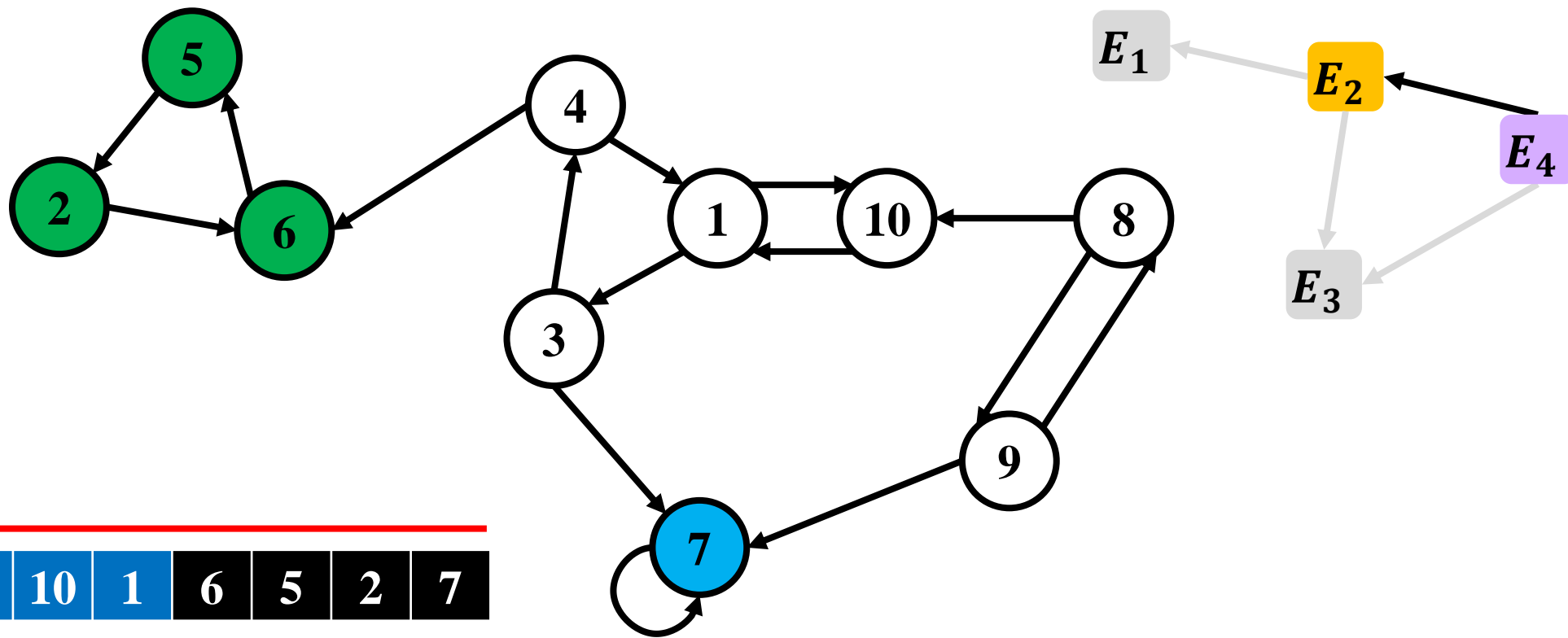
结合性质观察
算法第2次DFS



正确性证明

- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点
 - 性质1: G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2: 删除 SCC_{Sink} , 会产生新的 SCC_{Sink}

结合性质观察
算法第2次DFS

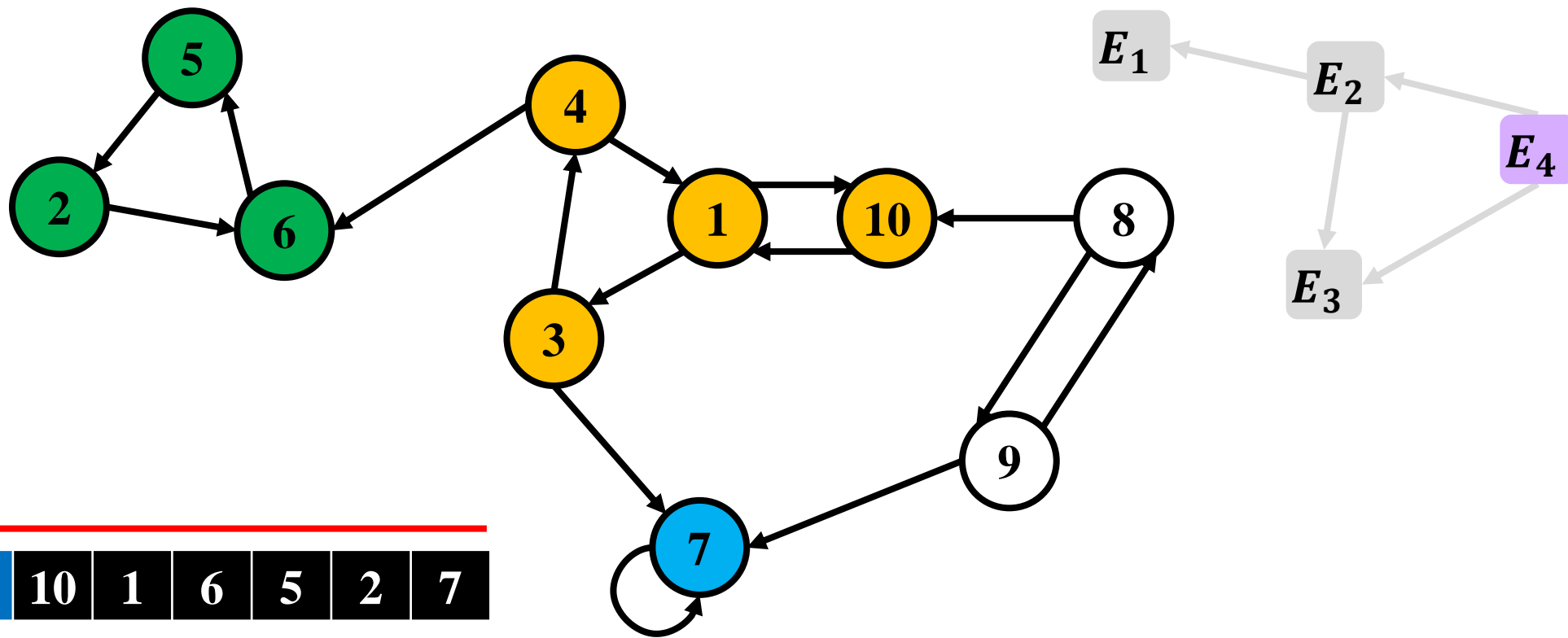


L	3	4	9	8	10	1	6	5	2	7
-----	---	---	---	---	----	---	---	---	---	---

正确性证明

- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点
 - 性质1: G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2: 删除 SCC_{Sink} , 会产生新的 SCC_{Sink}

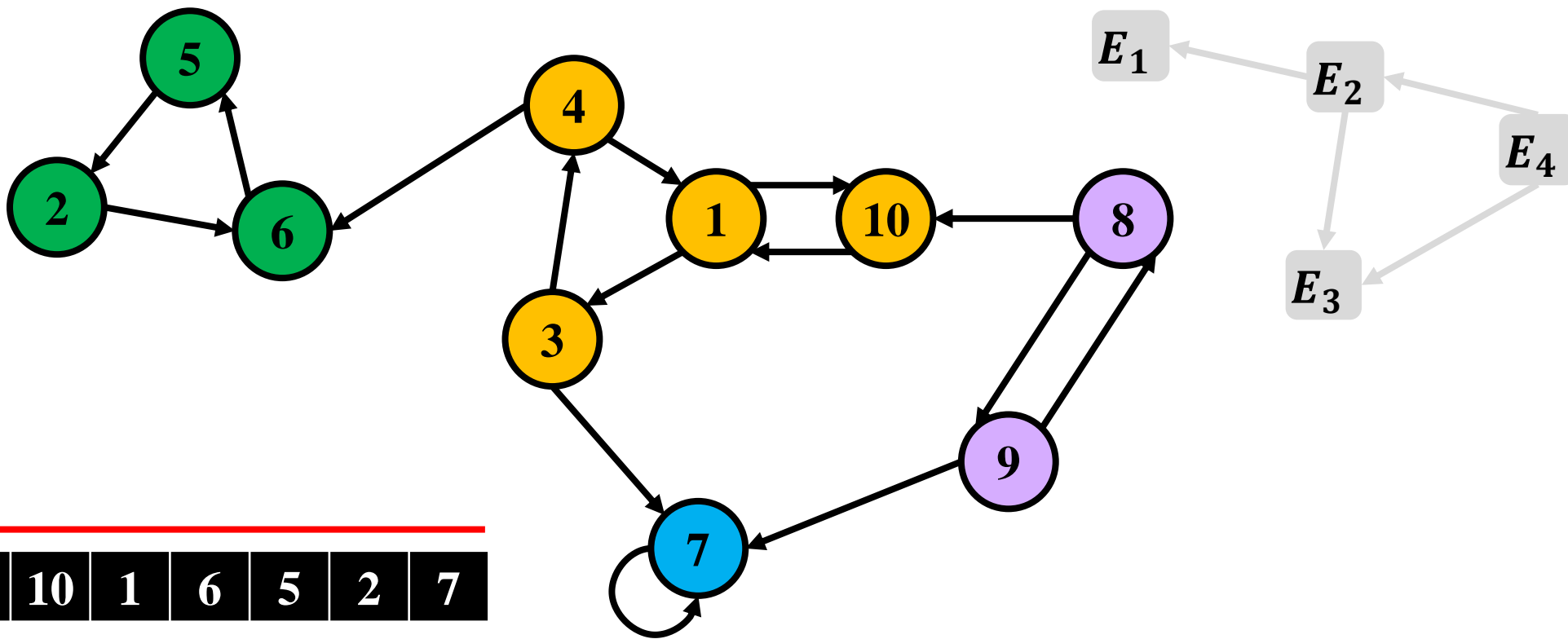
结合性质观察
算法第2次DFS



正确性证明

- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点
 - 性质1: G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2: 删除 SCC_{Sink} , 会产生新的 SCC_{Sink}

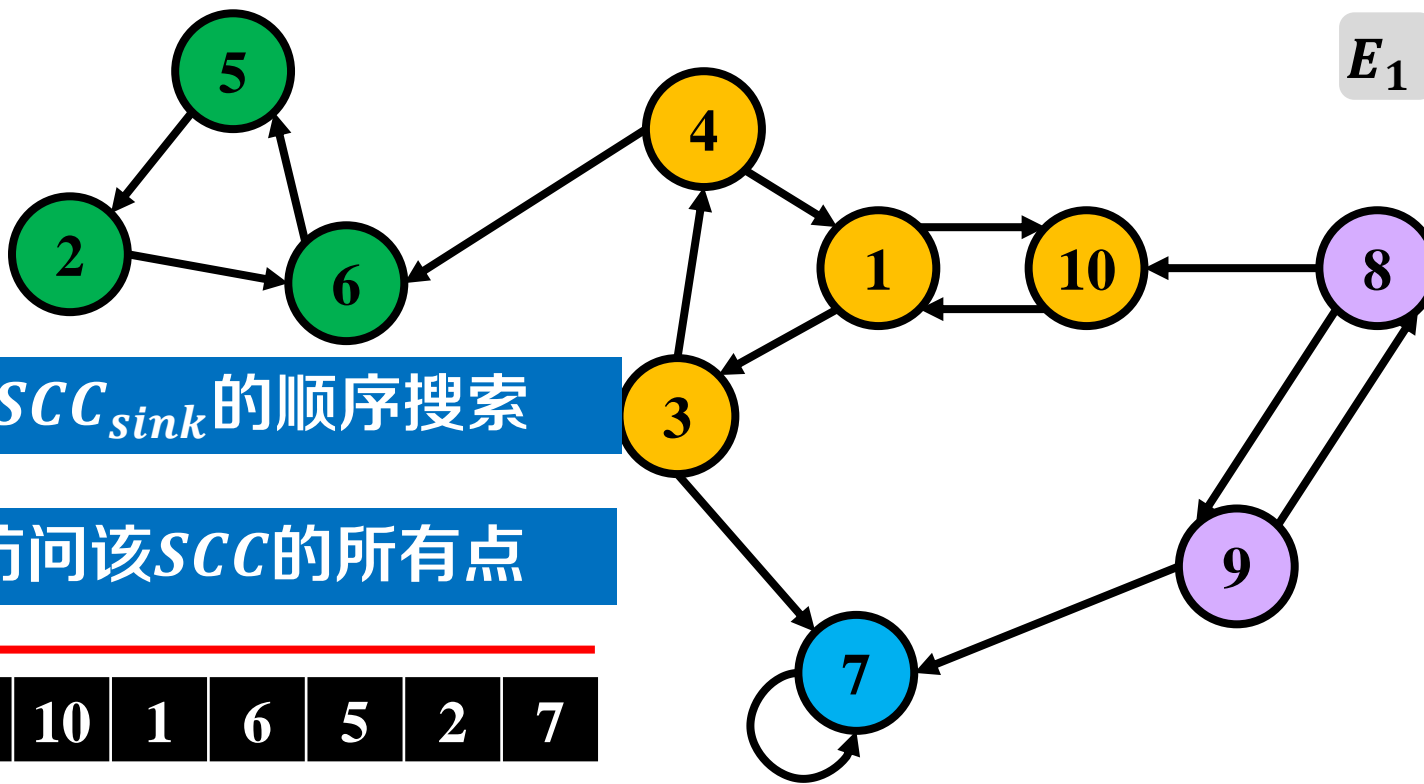
结合性质观察
算法第2次DFS



正确性证明

- 强连通分量图 G^{SCC} : 把强连通分量看作一个点, 得到有向图
 - 性质: G^{SCC} 一定是有向无环图
- SCC_{Sink} : G^{SCC} 中出度为0的点
 - 性质1: G^{SCC} 中存在至少一个 SCC_{Sink}
 - 性质2: 删除 SCC_{Sink} , 会产生新的 SCC_{Sink}

结合性质观察
算法第2次DFS



第2次DFS按照 SCC_{sink} 的顺序搜索

每次搜索恰好访问该 SCC 的所有点

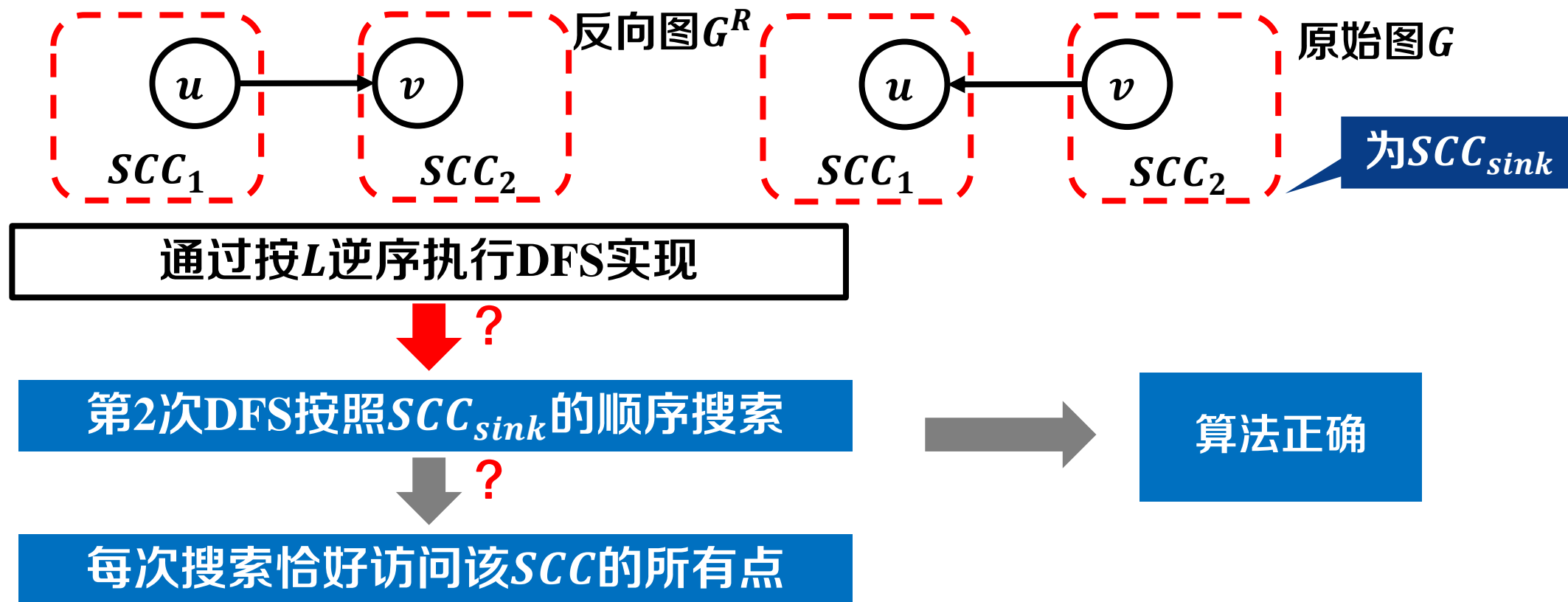




正确性证明



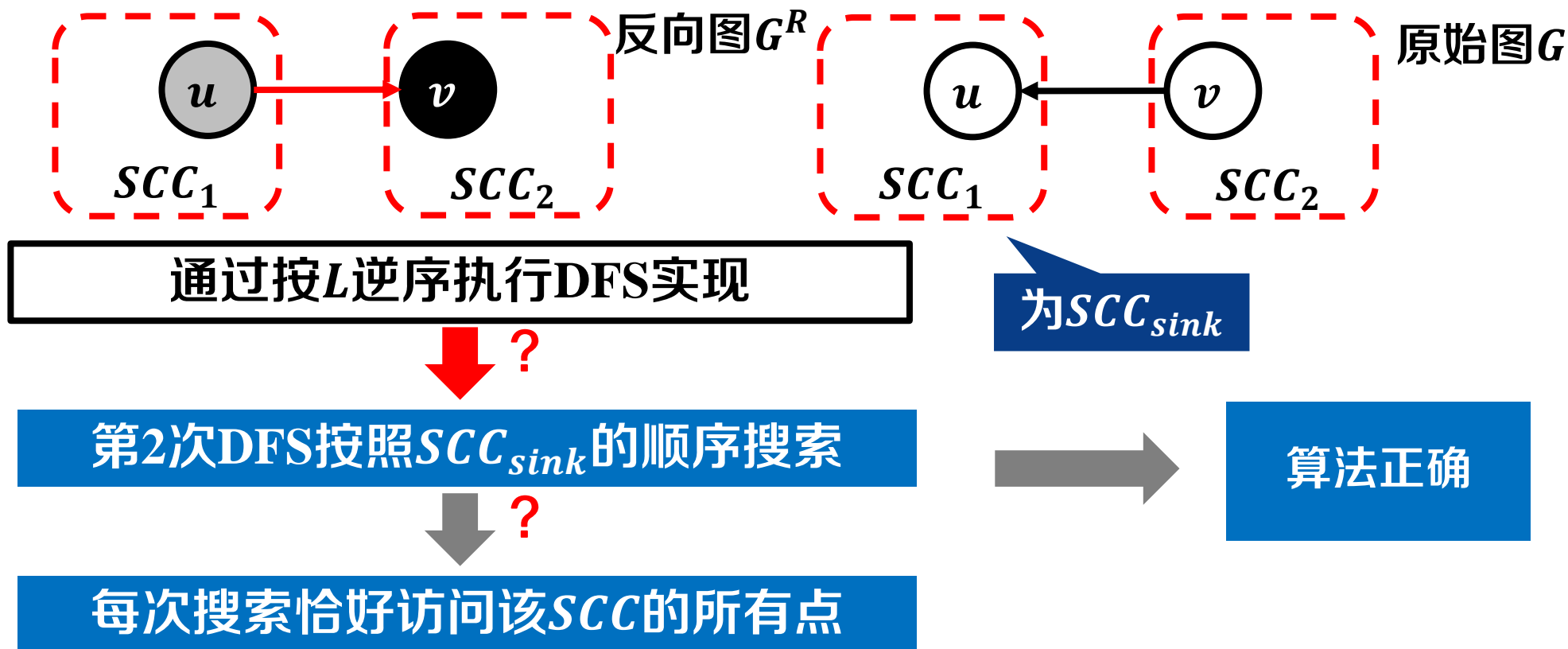
- 给定反向图 G^R ，存在边 (u, v) ， $u \in SCC_1$ ， $v \in SCC_2$



正确性证明



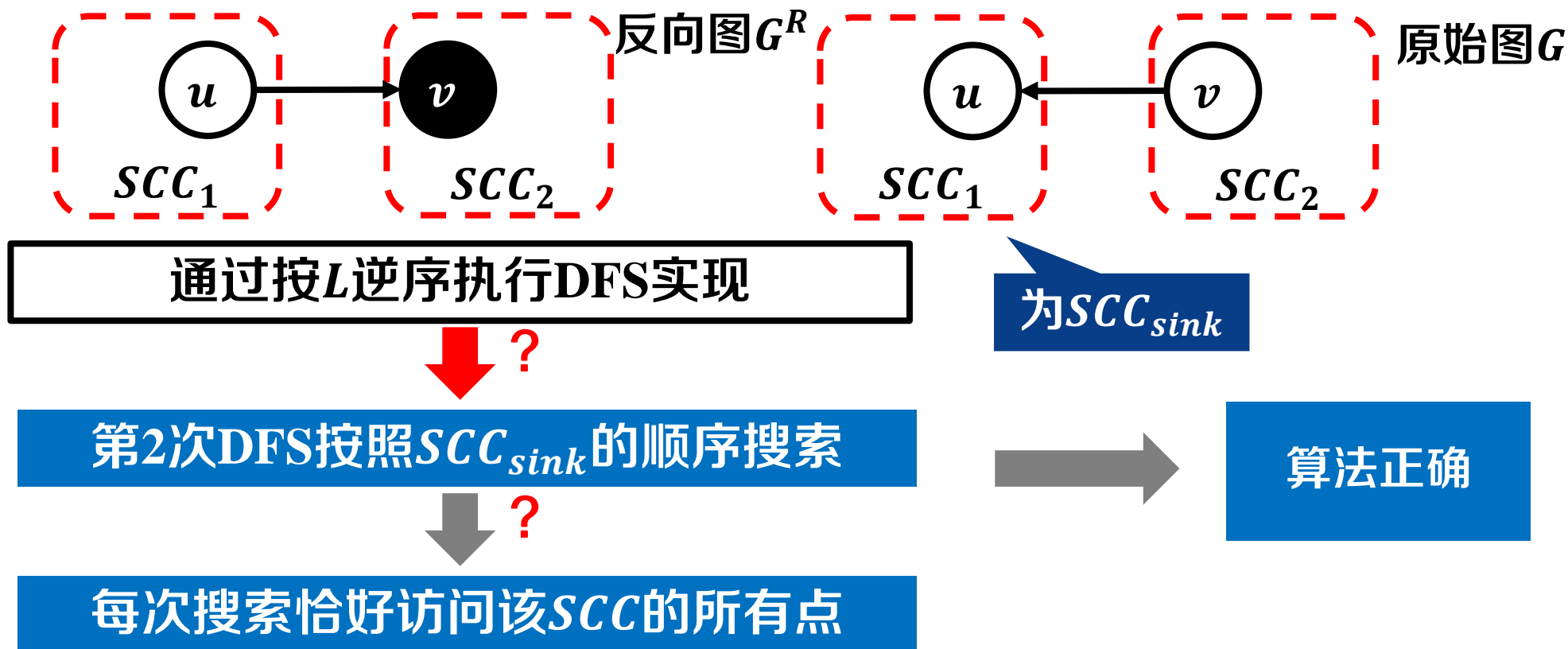
- 给定反向图 G^R , 存在边 (u, v) , $u \in SCC_1$, $v \in SCC_2$
 - 若先搜索 u , 会从 u 搜索 v , 则 $f(v) < f(u)$



正确性证明



- 给定反向图 G^R ，存在边 (u, v) ， $u \in SCC_1$ ， $v \in SCC_2$
 - 若先搜索 u ，会从 u 搜索 v ，则 $f(v) < f(u)$
 - 若先搜索 v ， v 搜索完成才开始搜索 u ，则 $f(v) < f(u)$



正确性证明

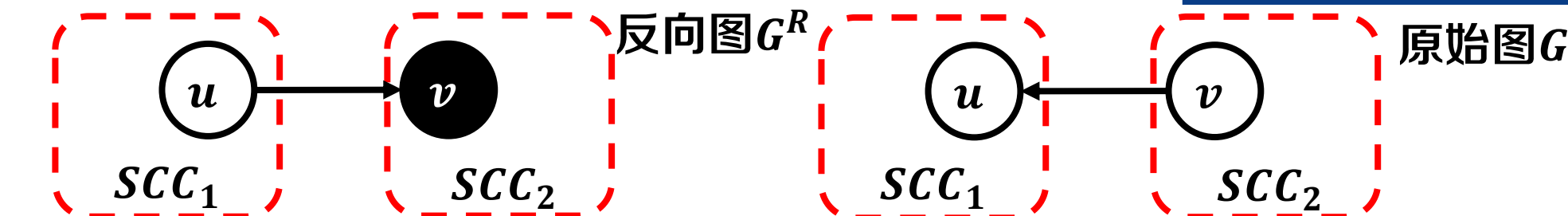


- 给定反向图 G^R , 存在边 (u, v) , $u \in SCC_1$, $v \in SCC_2$
 - 若先搜索 u , 会从 u 搜索 v , 则 $f(v) < f(u)$
 - 若先搜索 v , v 搜索完成才开始搜索 u , 则 $f(v) < f(u)$

f 越来越小



L : G^R 上DFS完成时刻顺序



通过按 L 逆序执行DFS实现

为 SCC_{sink}

第2次DFS按照 SCC_{sink} 的顺序搜索

算法正确

每次搜索恰好访问该 SCC 的所有点

正确性证明

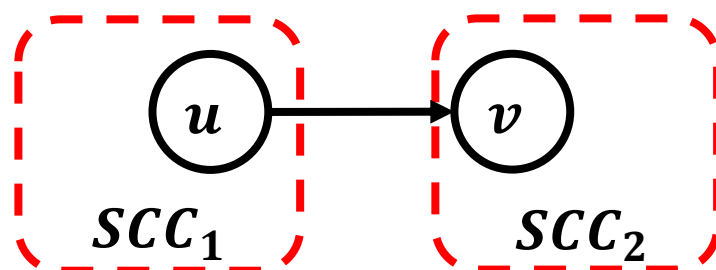


- 给定反向图 G^R , 存在边 (u, v) , $u \in SCC_1$, $v \in SCC_2$
 - 若先搜索 u , 会从 u 搜索 v , 则 $f(v) < f(u)$
 - 若先搜索 v , v 搜索完成才开始搜索 u , 则 $f(v) < f(u)$
 - 所以按 L 的逆序, 总是先搜索 u , 符合 SCC_{sink} 的顺序

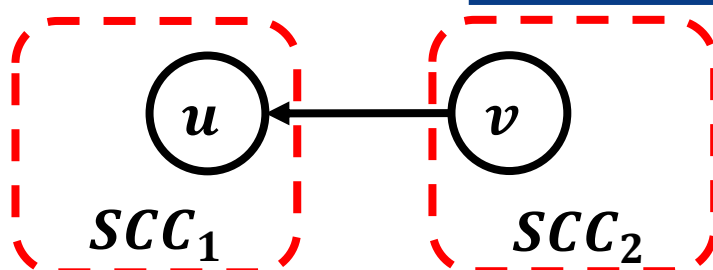
f 越来越小



L : G^R 上DFS完成时刻顺序



反向图 G^R



原始图 G

通过按 L 逆序执行DFS实现

为 SCC_{sink}

第2次DFS按照 SCC_{sink} 的顺序搜索

每次搜索恰好访问该 SCC 的所有点

算法正确

正确性证明

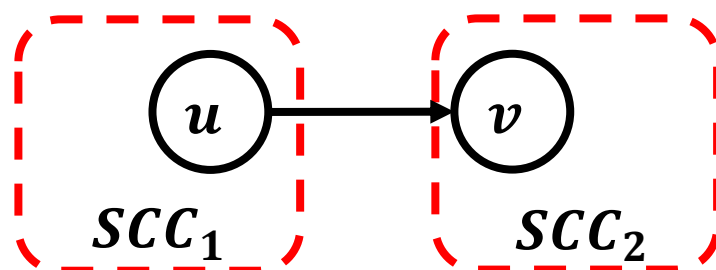


- 给定反向图 G^R , 存在边 (u, v) , $u \in SCC_1$, $v \in SCC_2$
 - 若先搜索 u , 会从 u 搜索 v , 则 $f(v) < f(u)$
 - 若先搜索 v , v 搜索完成才开始搜索 u , 则 $f(v) < f(u)$
 - 所以按 L 的逆序, 总是先搜索 u , 符合 SCC_{sink} 的顺序

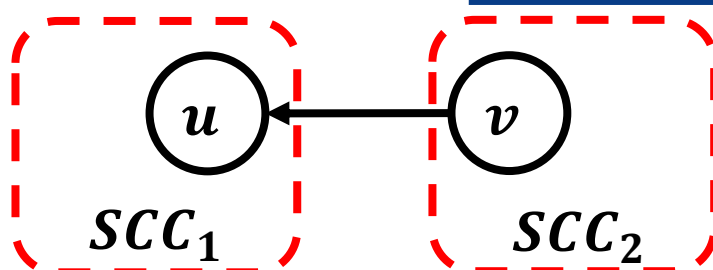
f 越来越小



L : G^R 上DFS完成时刻顺序



反向图 G^R



原始图 G

通过按 L 逆序执行DFS实现

为 SCC_{sink}

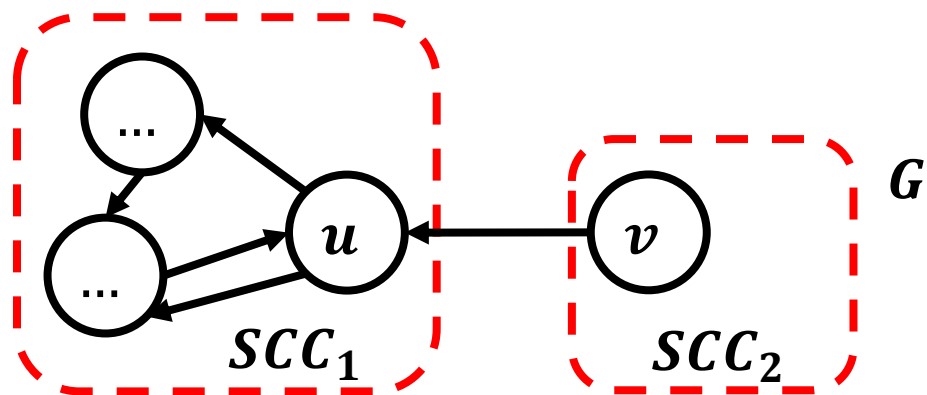
第2次DFS按照 SCC_{sink} 的顺序搜索

每次搜索恰好访问该 SCC 的所有点

算法正确

正确性证明

- 强连通分量内，顶点相互可达 \longrightarrow DFS可以访问到该SCC所有点
- SCC_{sink} 出度为0 \longrightarrow DFS不会访问该SCC以外的点



通过按 L 逆序执行DFS实现



第2次DFS按照 SCC_{sink} 的顺序搜索



每次搜索恰好访问该SCC的所有点

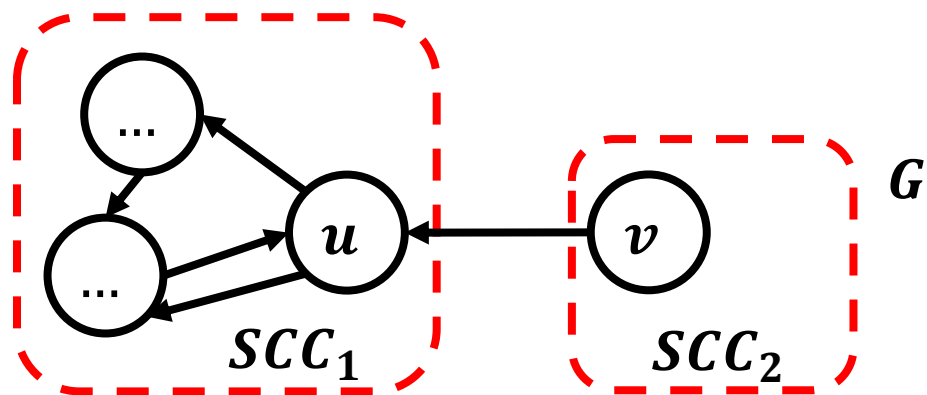


算法正确

正确性证明



- 强连通分量内，顶点相互可达 \longrightarrow DFS可以访问到该SCC所有点
- SCC_{sink} 出度为0 \longrightarrow DFS不会访问该SCC以外的点



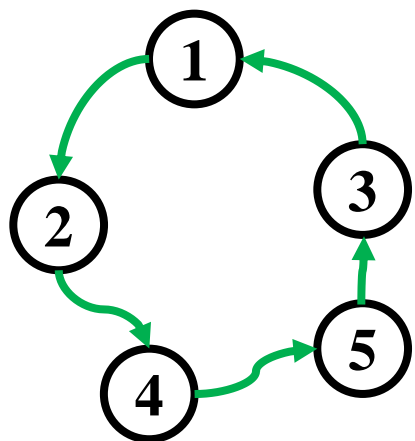
通过按 L 逆序执行DFS实现

第2次DFS按照 SCC_{sink} 的顺序搜索

每次搜索恰好访问该SCC的所有点

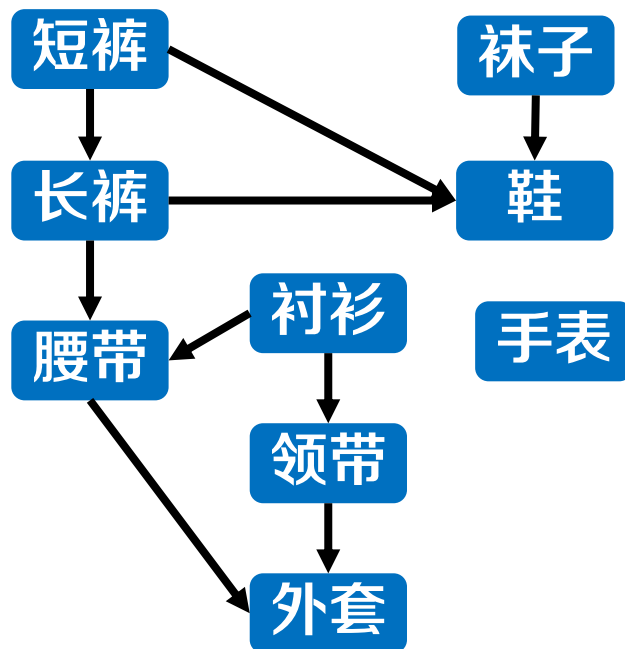
算法正确

- 深度优先搜索的应用



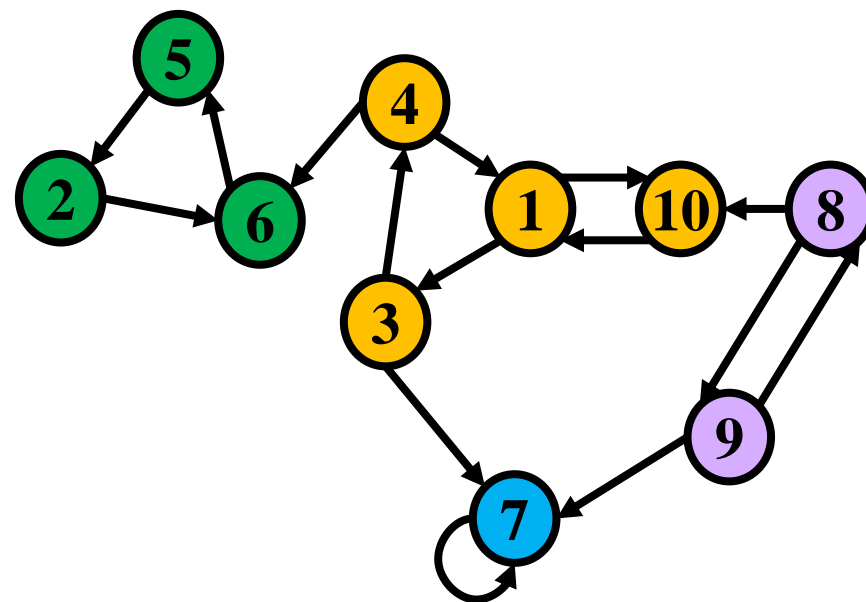
环的存在性判定

利用深度优先搜索边的性质



拓扑排序

利用深度优先搜索点的性质（括号化定理）



强连通分量