Design and Analysis of Algorithms

Part IV: Graph Algorithms

Lecture 26: Strongly Connected Components

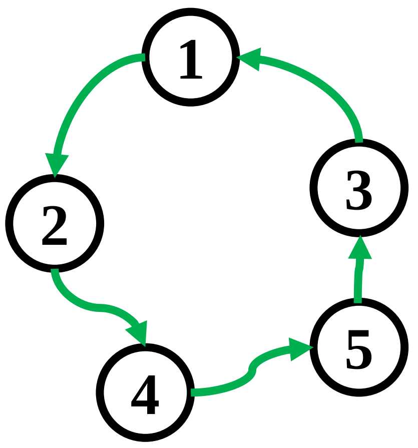
童咏昕

北京航空航天大学
计算机学院

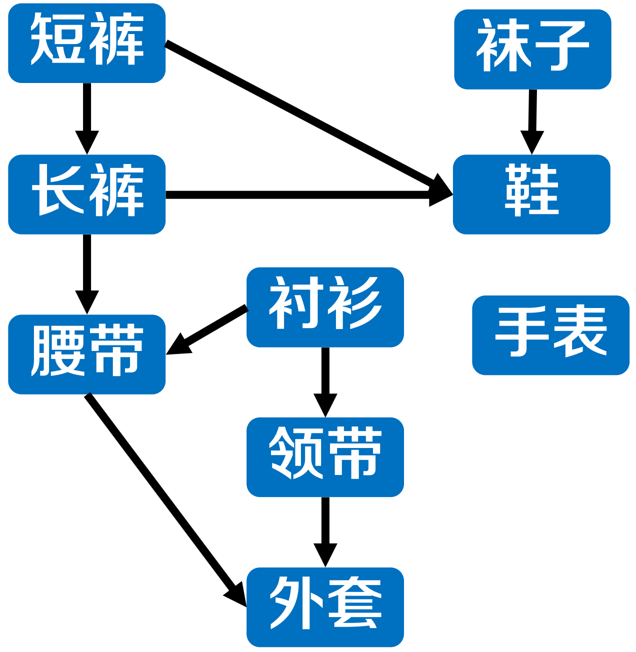
图算法篇概述

- 在算法课程第四部分“图算法”主题中，我们将主要聚焦于如下经典问题：
  - Basic Concepts in Graph Algorithms（图算法的基本概念）
  - Breadth-First Search (BFS，广度优先搜索）
  - Depth-First Search (DFS，深度优先搜索）
  - Cycle Detection（环路检测）
  - Topological Sort （拓扑排序）
  - Strongly Connected Components（强连通分量）
  - Minimum Spanning Trees (最小生成树)
  - Single Source Shortest Path (单源最短路径)
  - All-Pairs Shortest Paths (所有点对最短路径)
  - Bipartite Graph Matching（二分图匹配）
  - Maximum/Network Flows (最大流/网络流)

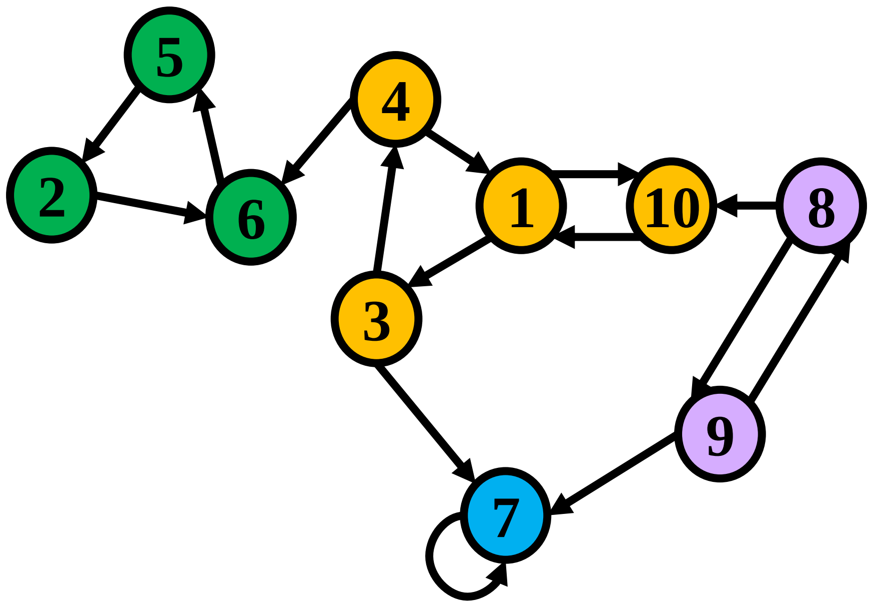
深度优先搜索应用



环路的存在性判断



拓扑排序



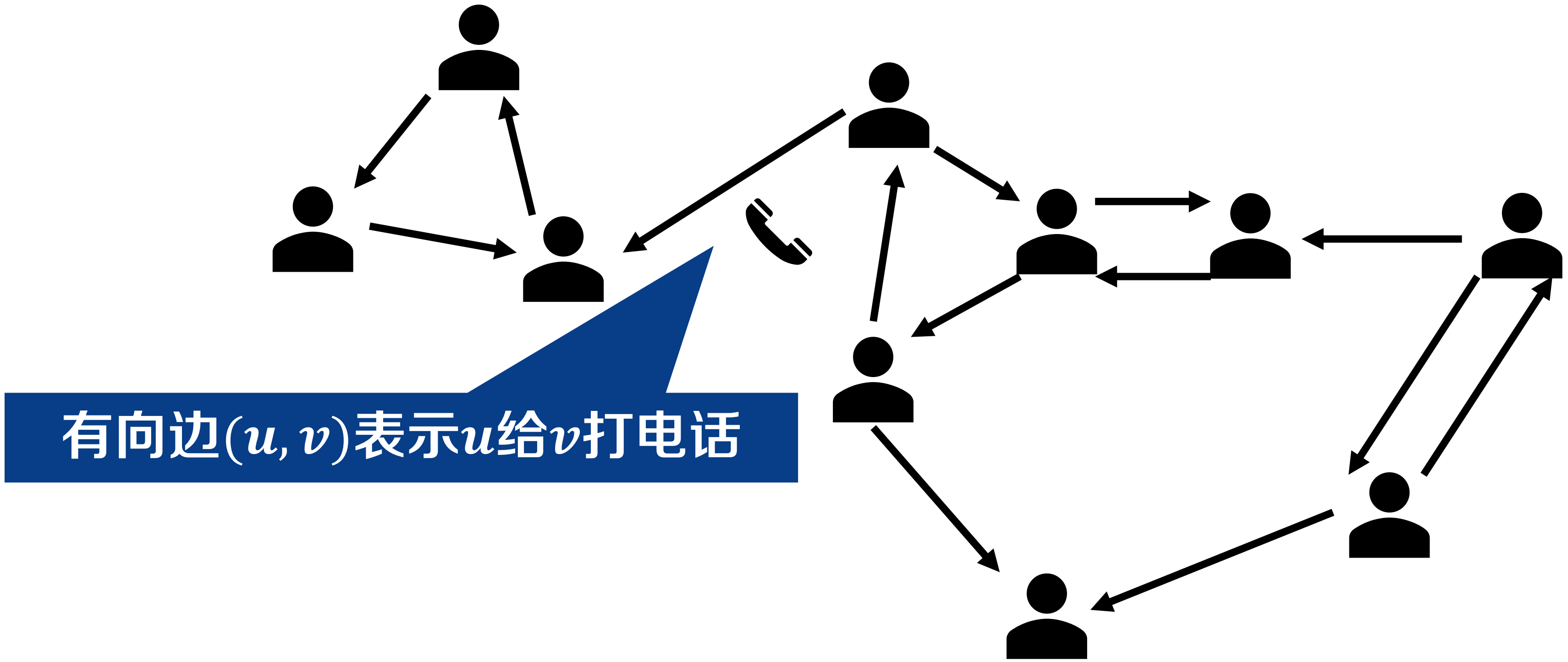
强连通分量

提纲

- 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

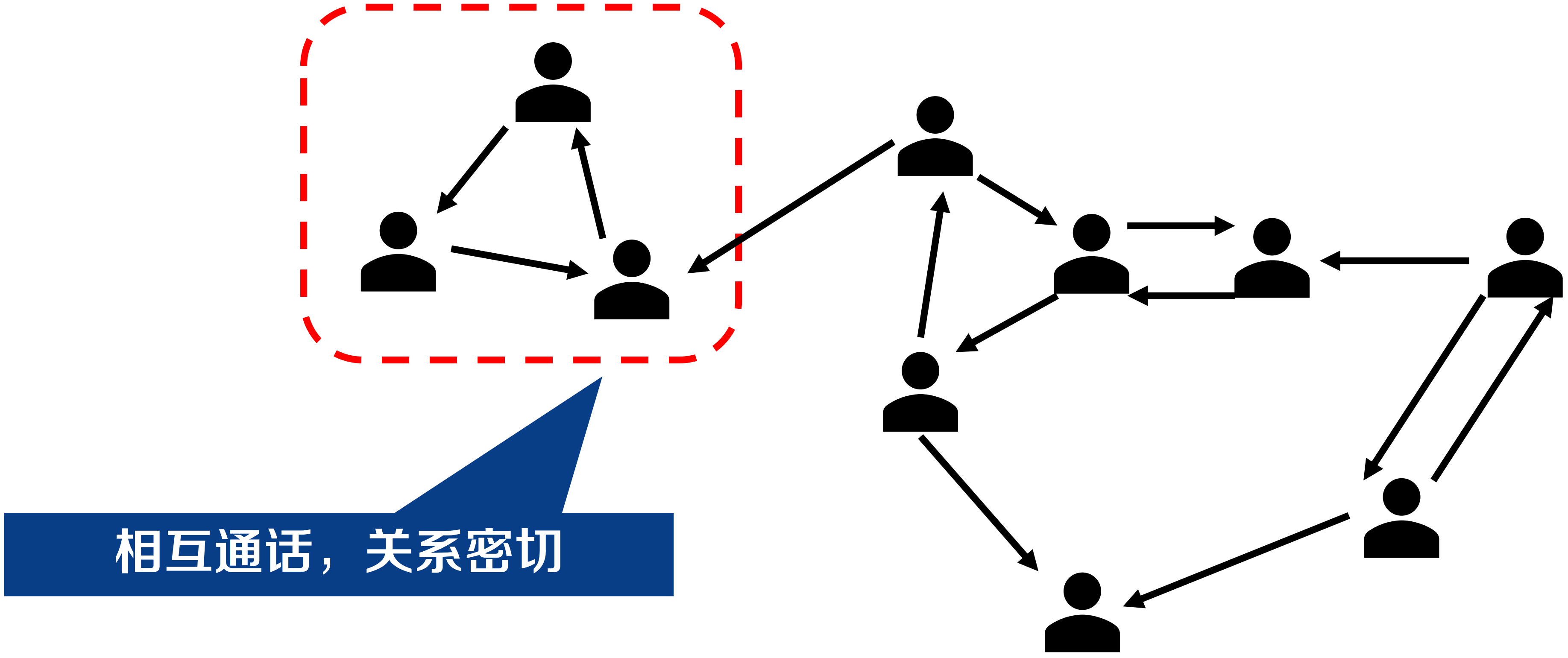
问题背景

- 社交圈划分
  - 如何把人群按通话记录划分成不同的社交圈?



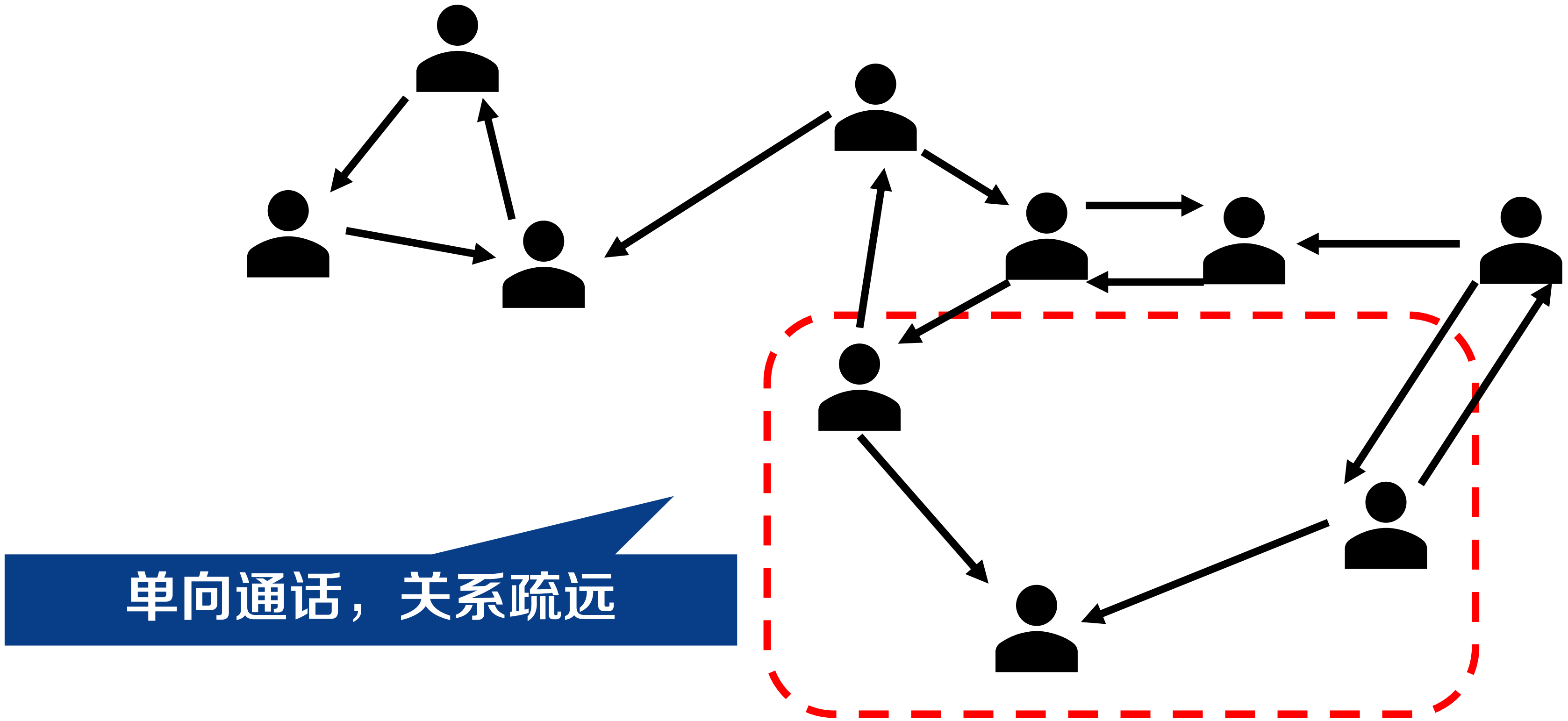
问题背景

- 社交圈划分
  - 如何把人群按通话记录划分成不同的社交圈?



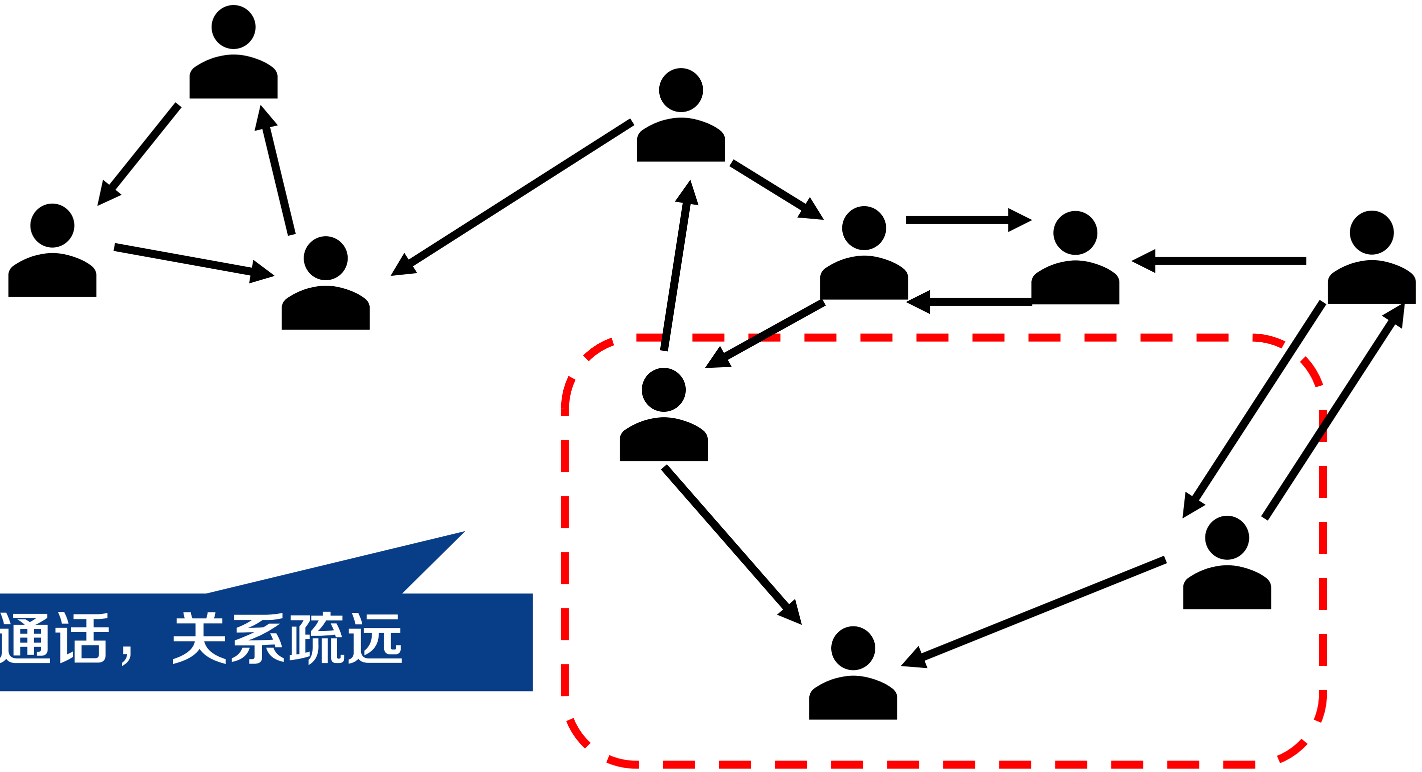
问题背景

- 社交圈划分
  - 如何把人群按通话记录划分成不同的社交圈?



问题背景

- 社交圈划分
  - 如何把人群按通话记录划分成不同的社交圈？

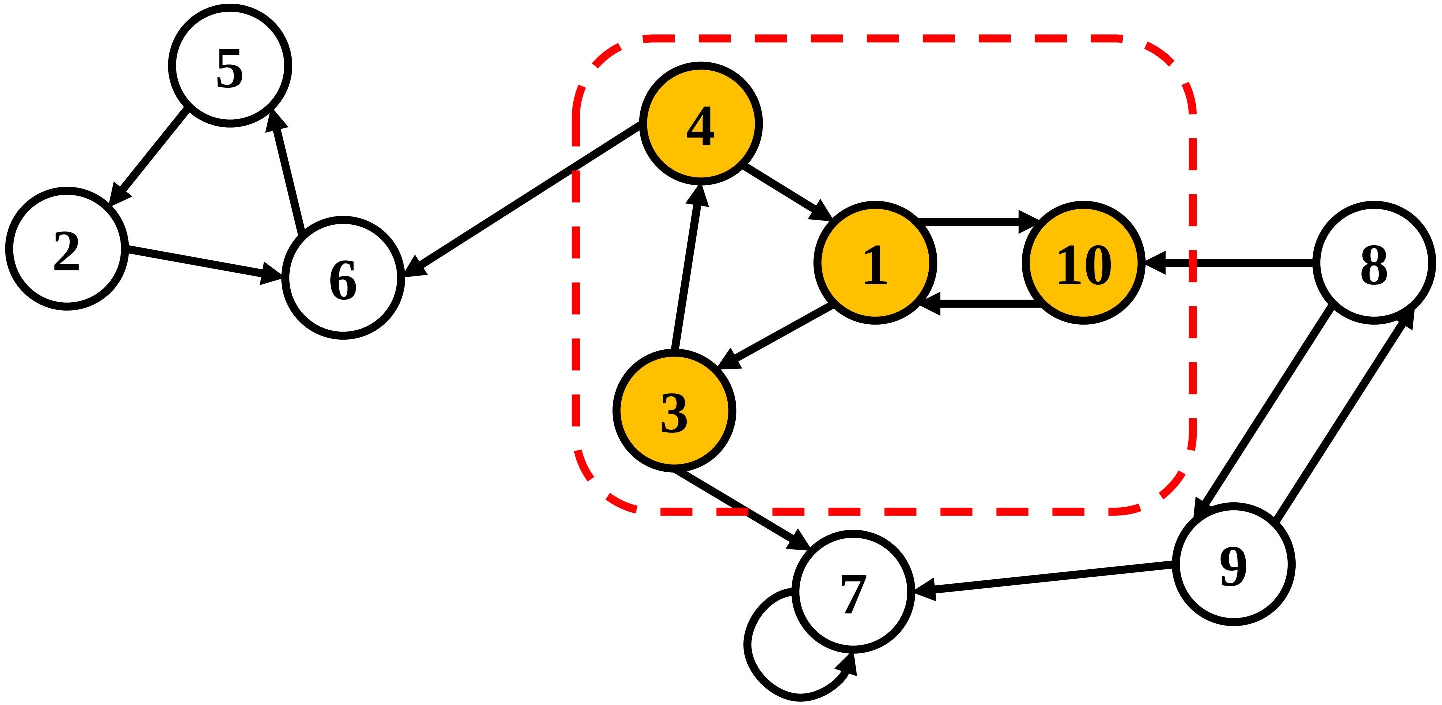


单向通话，关系疏远

问题：如何严格定义关系的亲密程度？

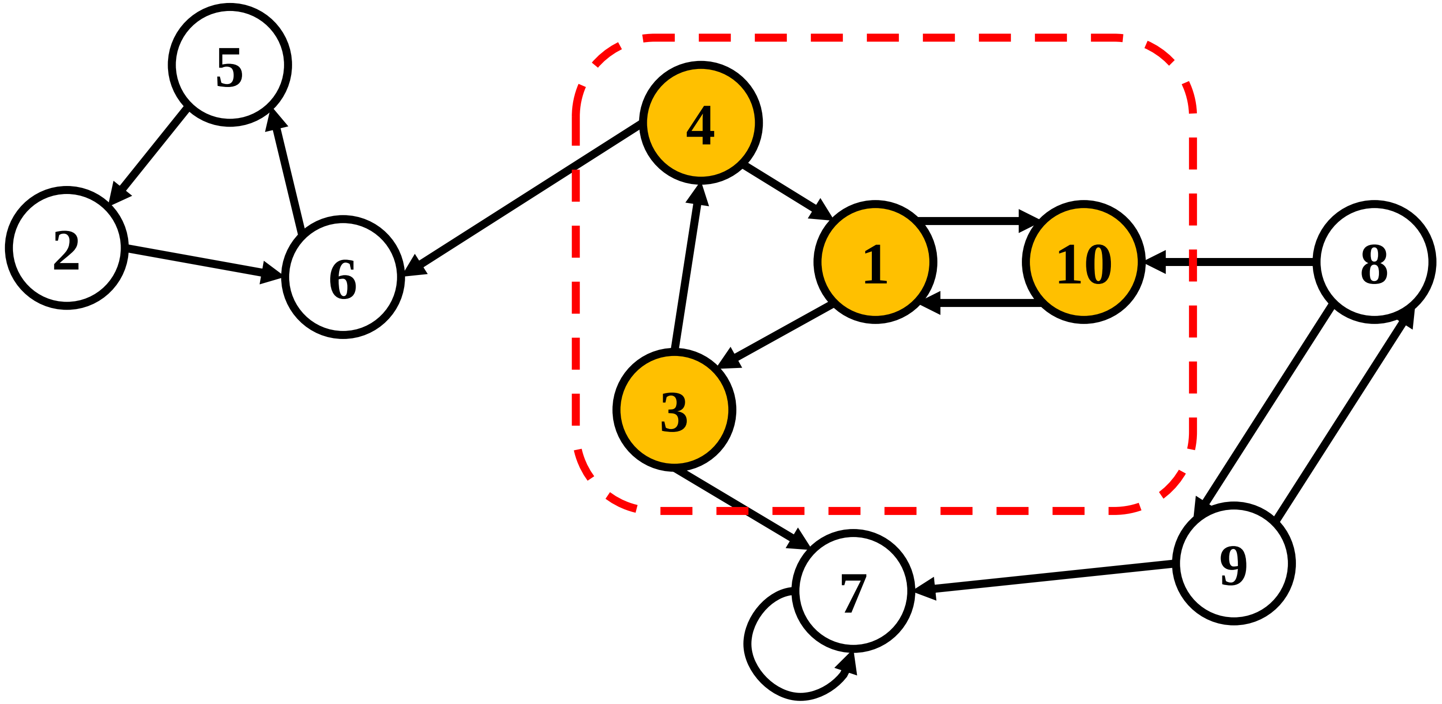
问题背景

- 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



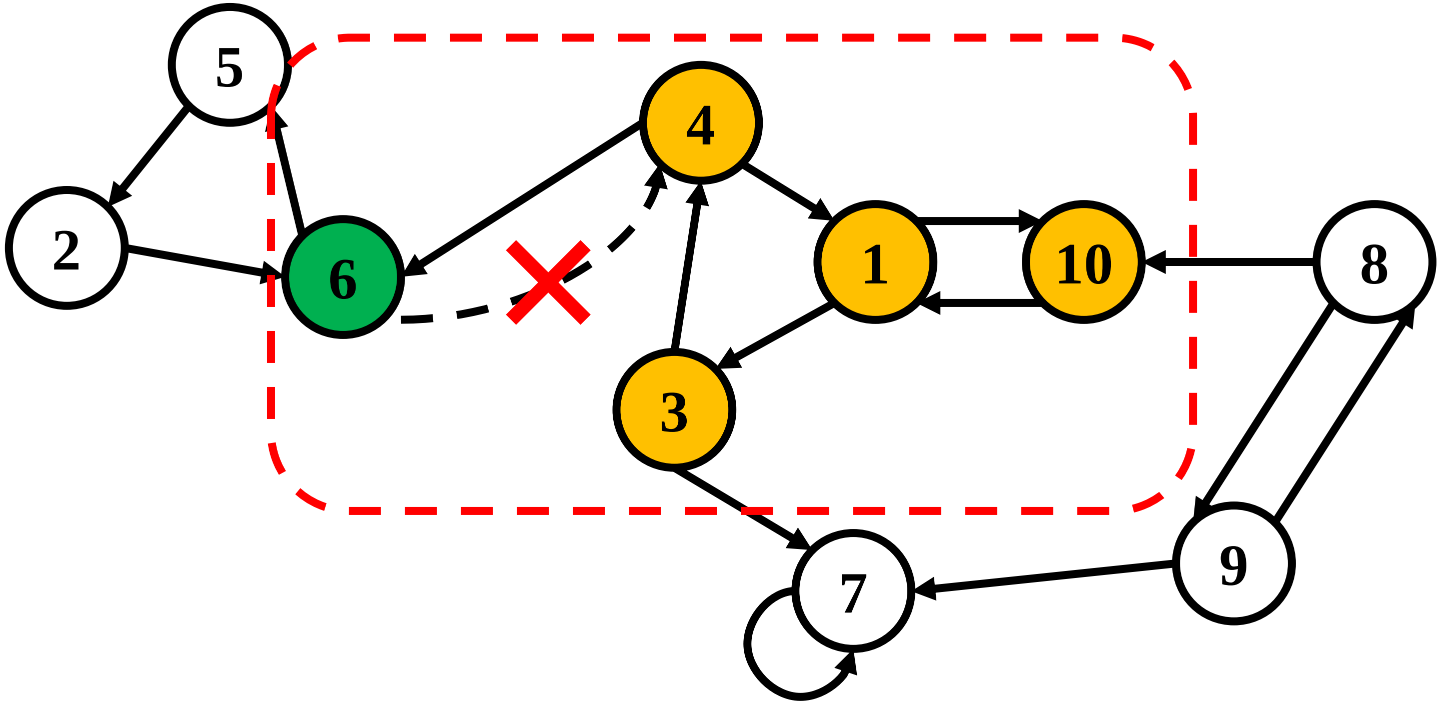
问题背景

- 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



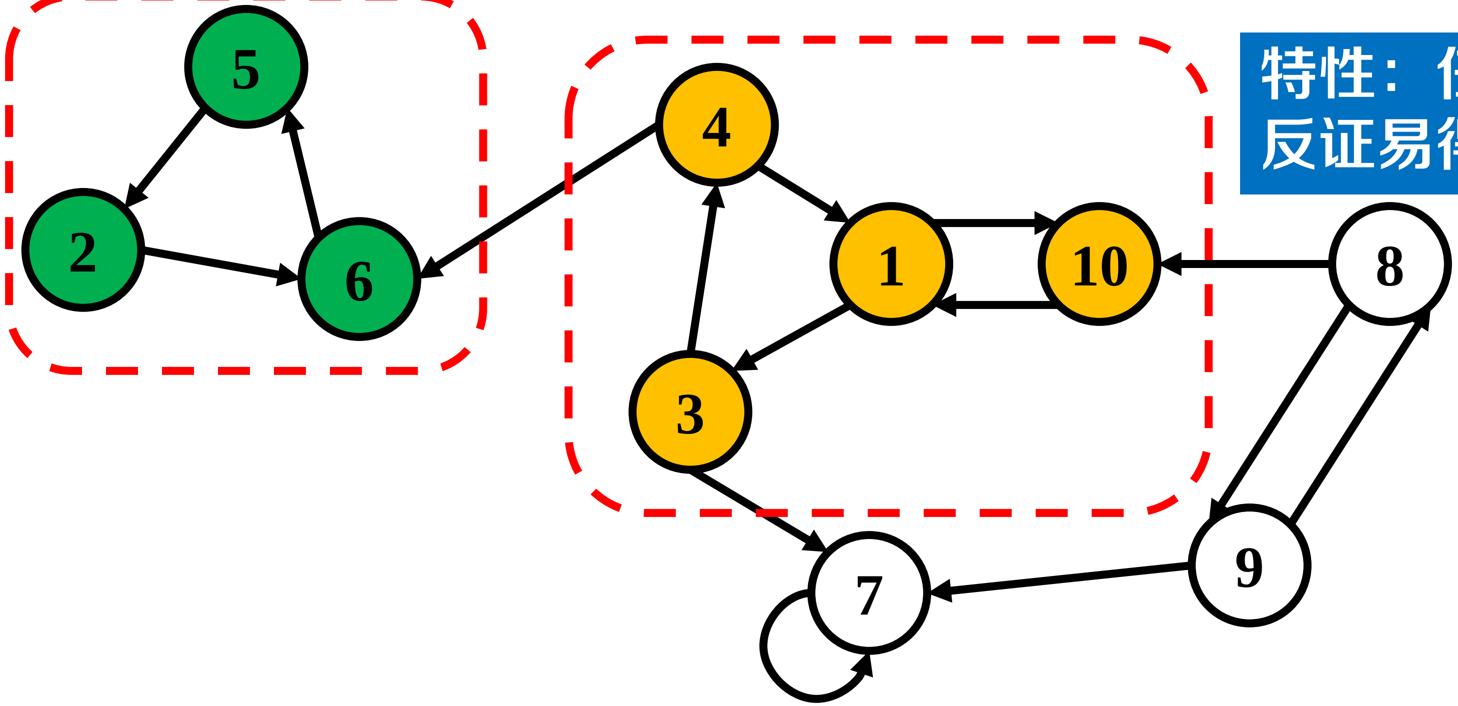
问题背景

- 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



问题背景

- 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

问题定义

强连通分量

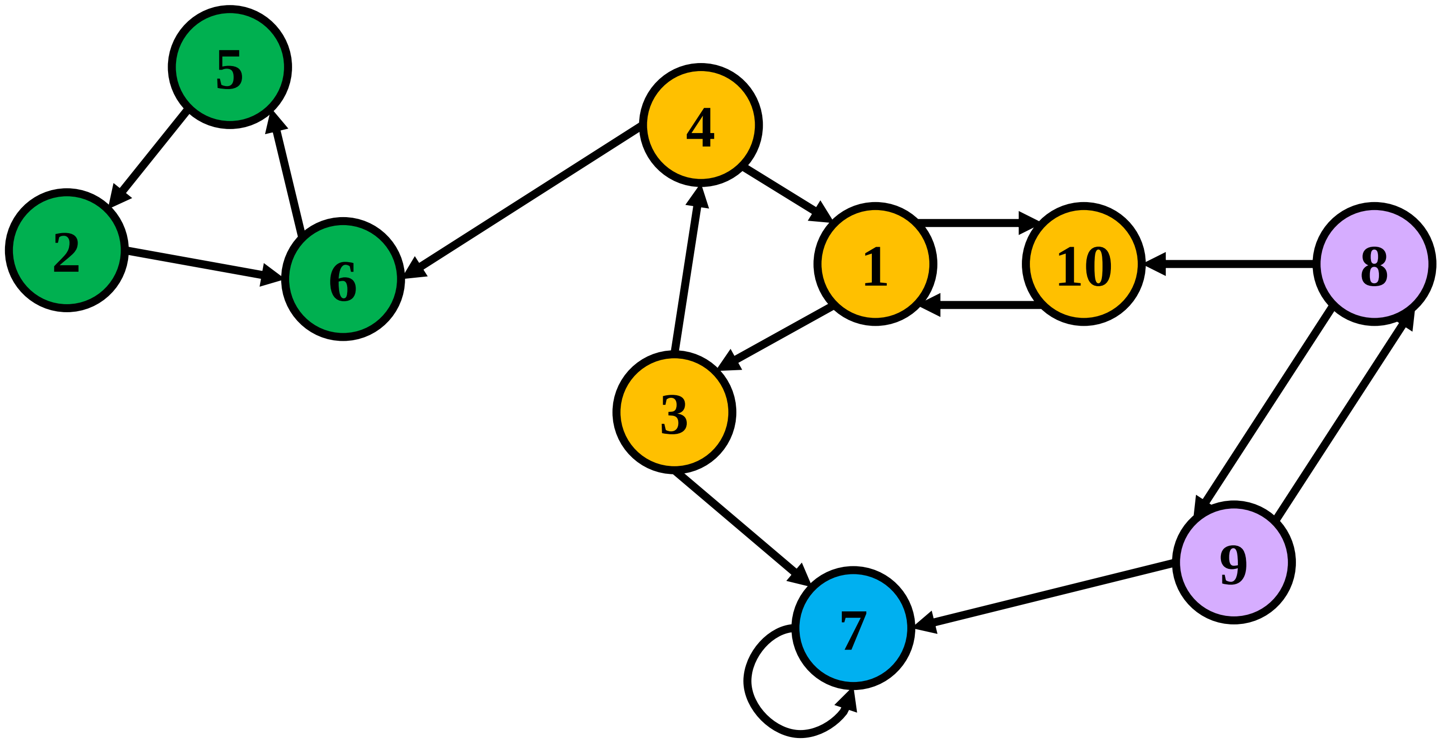
Strongly Connected Components

输入

- 有向图$G=<V,E>$

输出

- 图的所有强连通分量$C_1,C_2,\dots C_n$

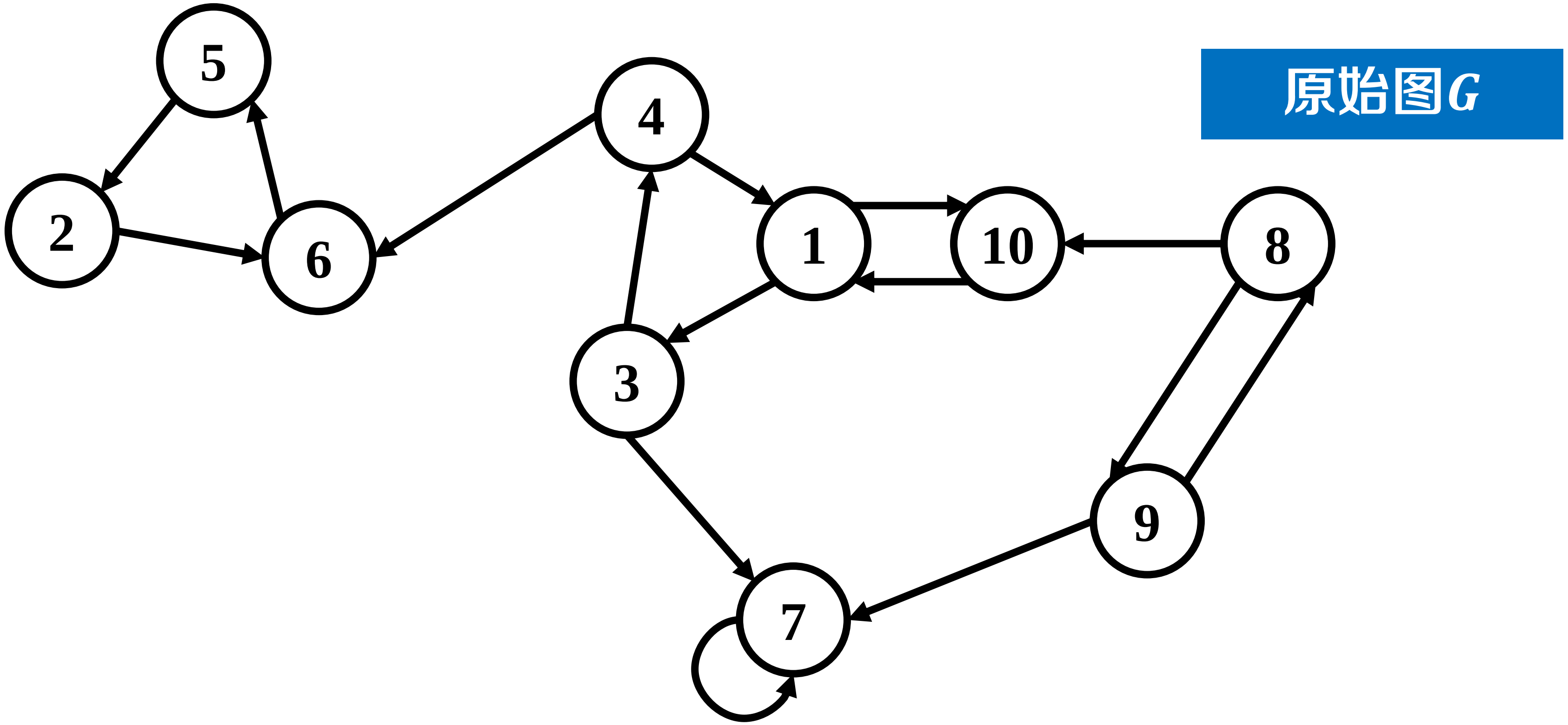


提纲

- 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

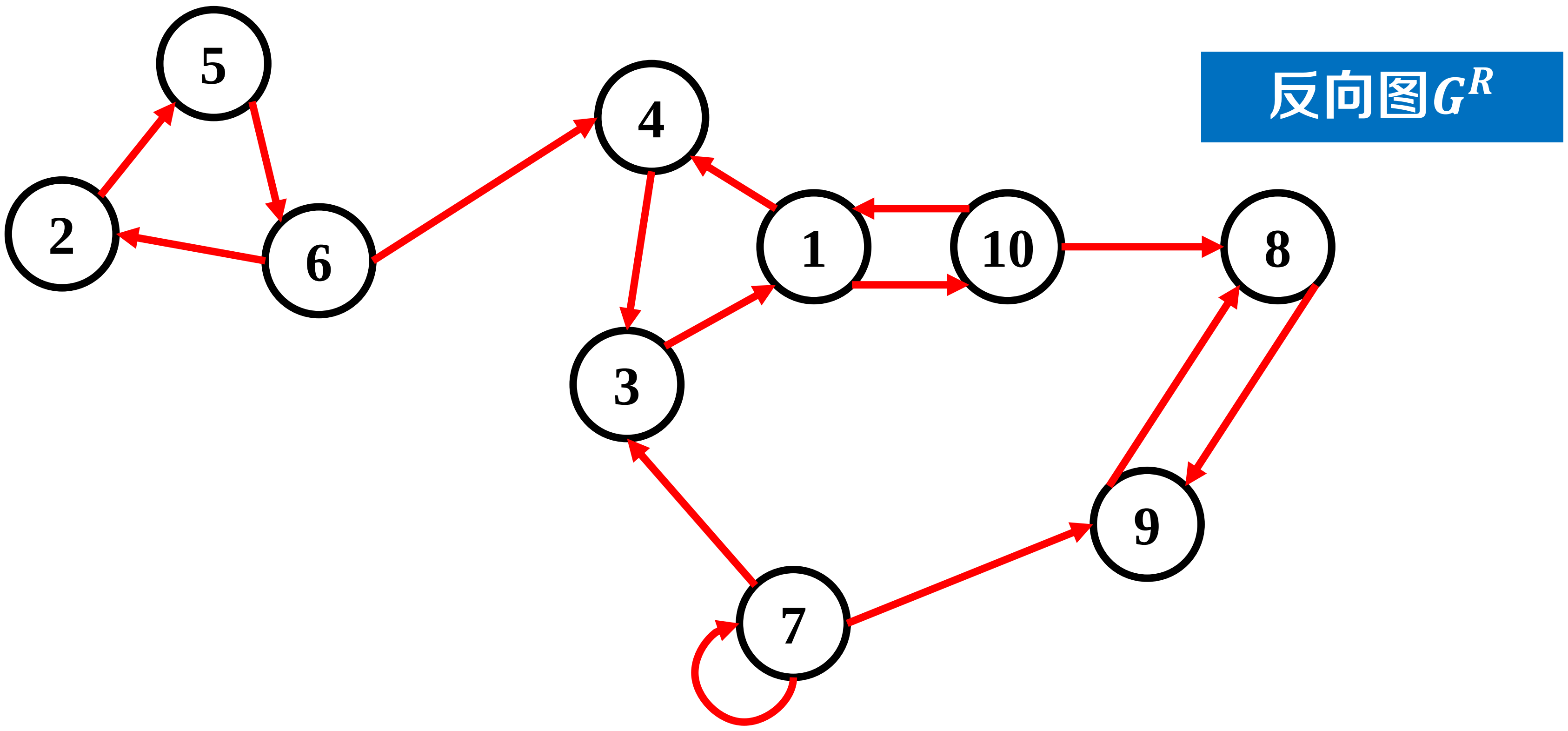
算法框架与实例

- 步骤1：把边反向，得到反向图G^R



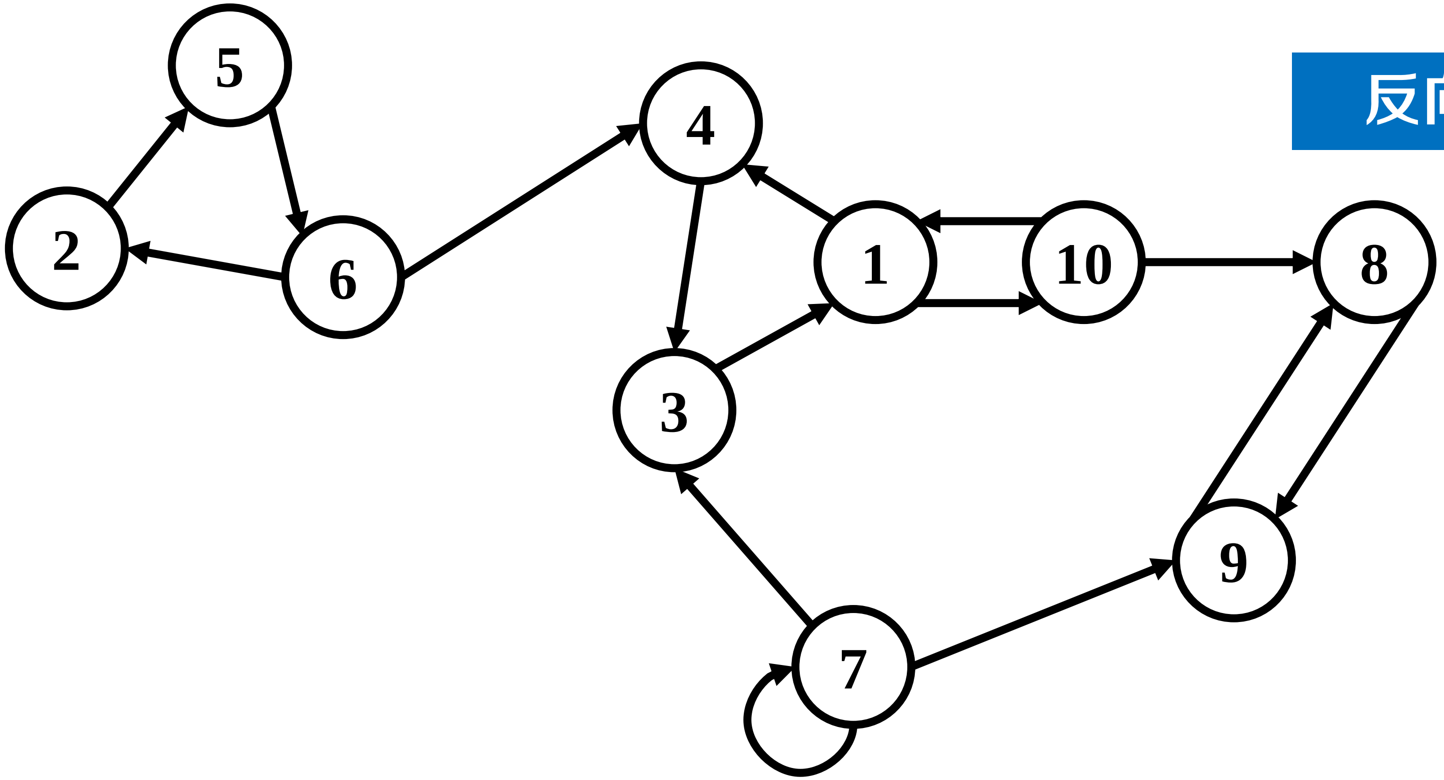
算法框架与实例

- 步骤1：把边反向，得到反向图G^R



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

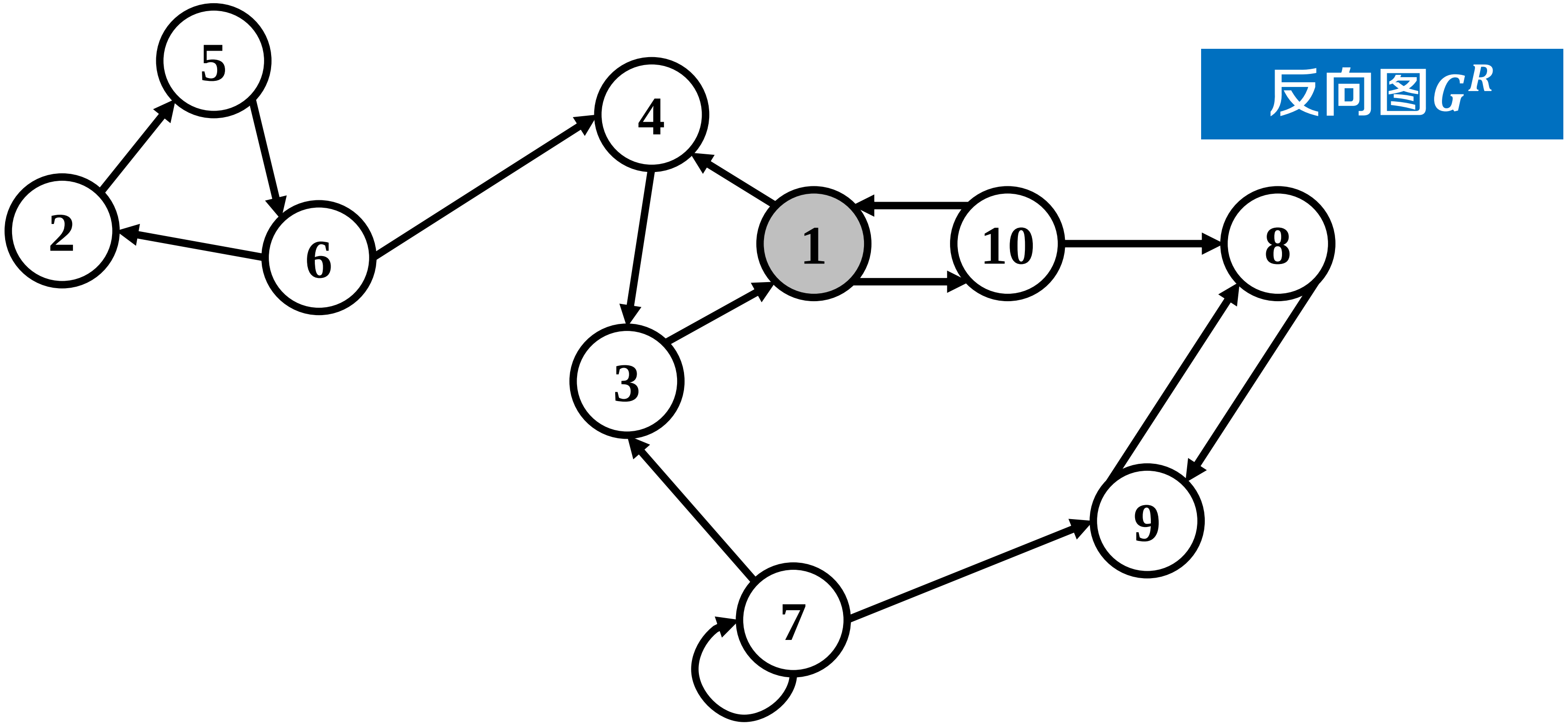


反向图G^R

L										

算法框架与实例

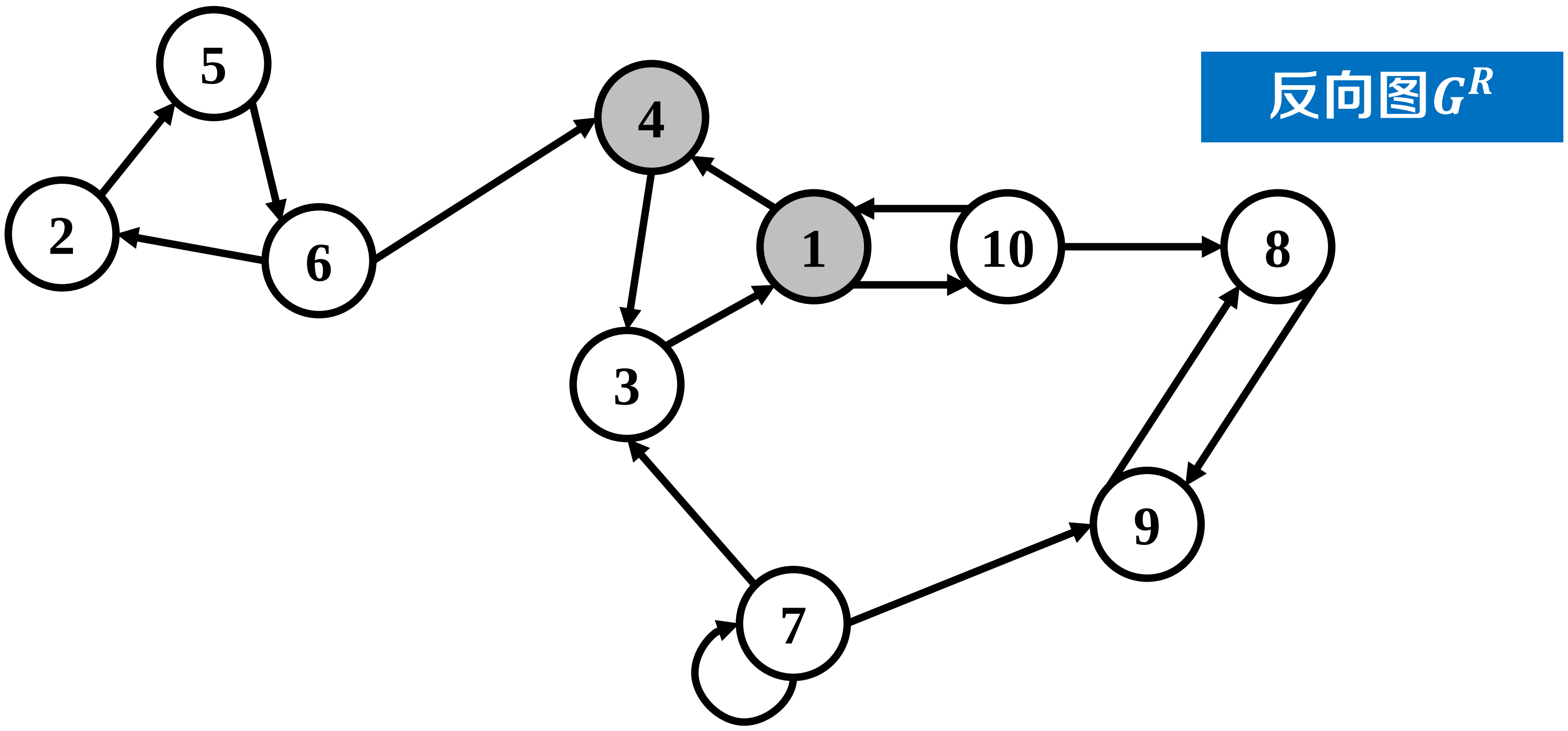
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L										

算法框架与实例

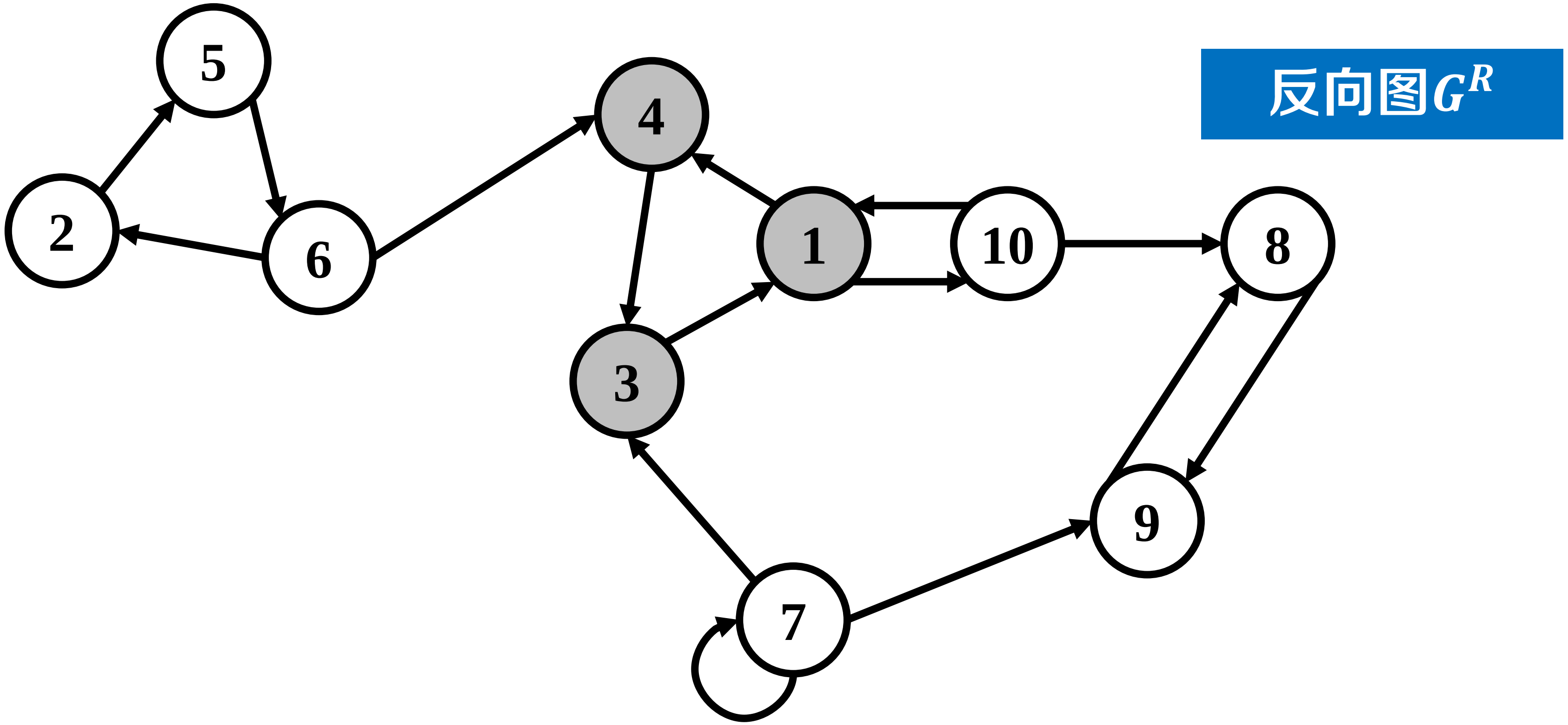
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L										

算法框架与实例

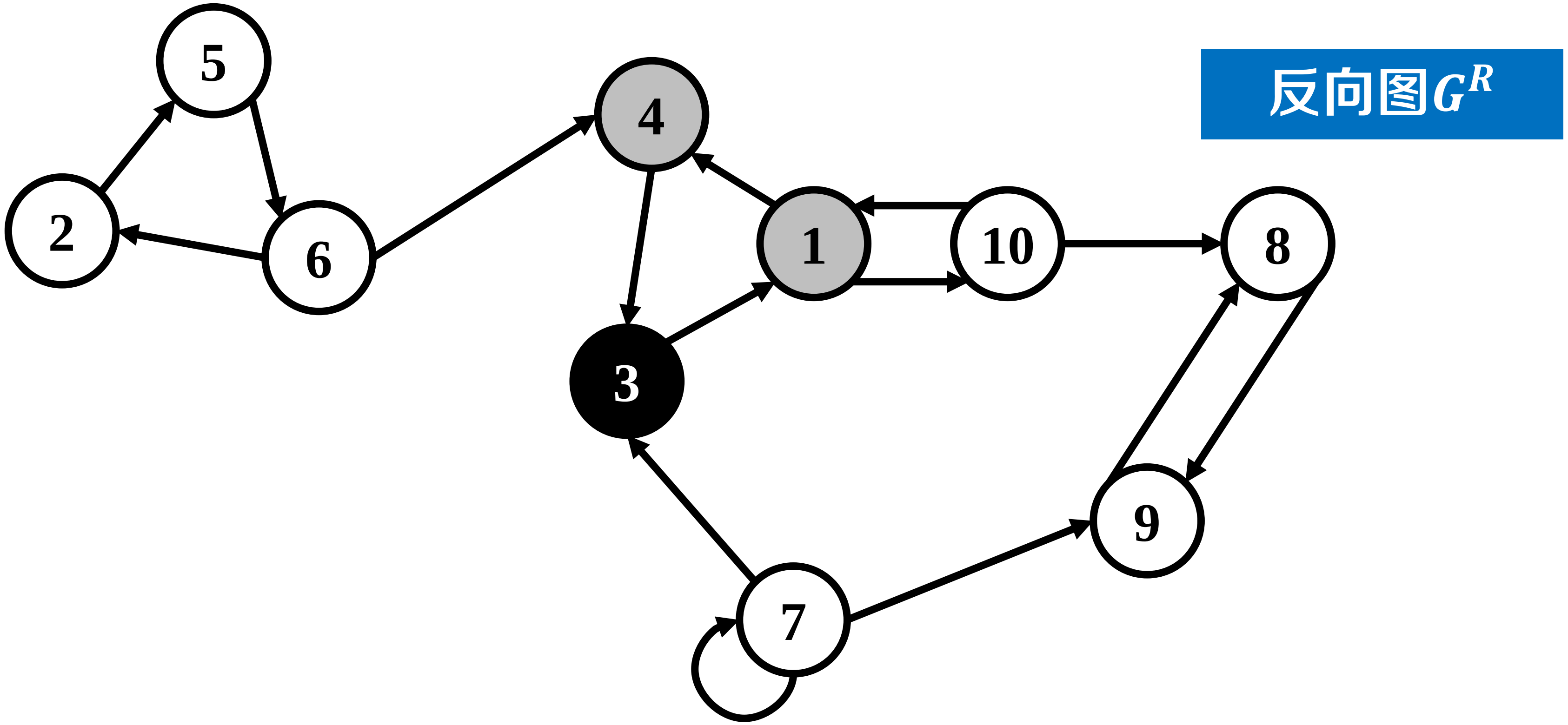
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L										

算法框架与实例

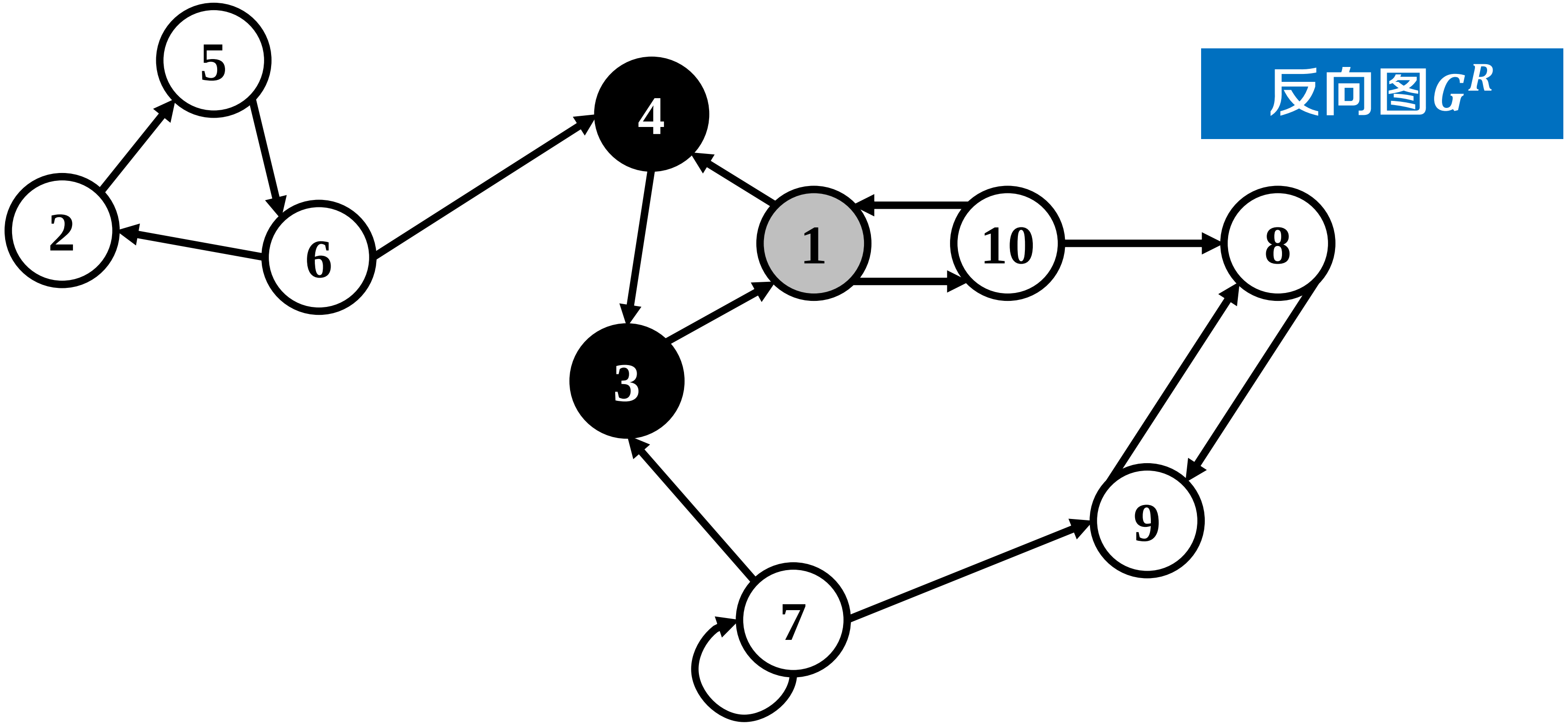
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L	3									

算法框架与实例

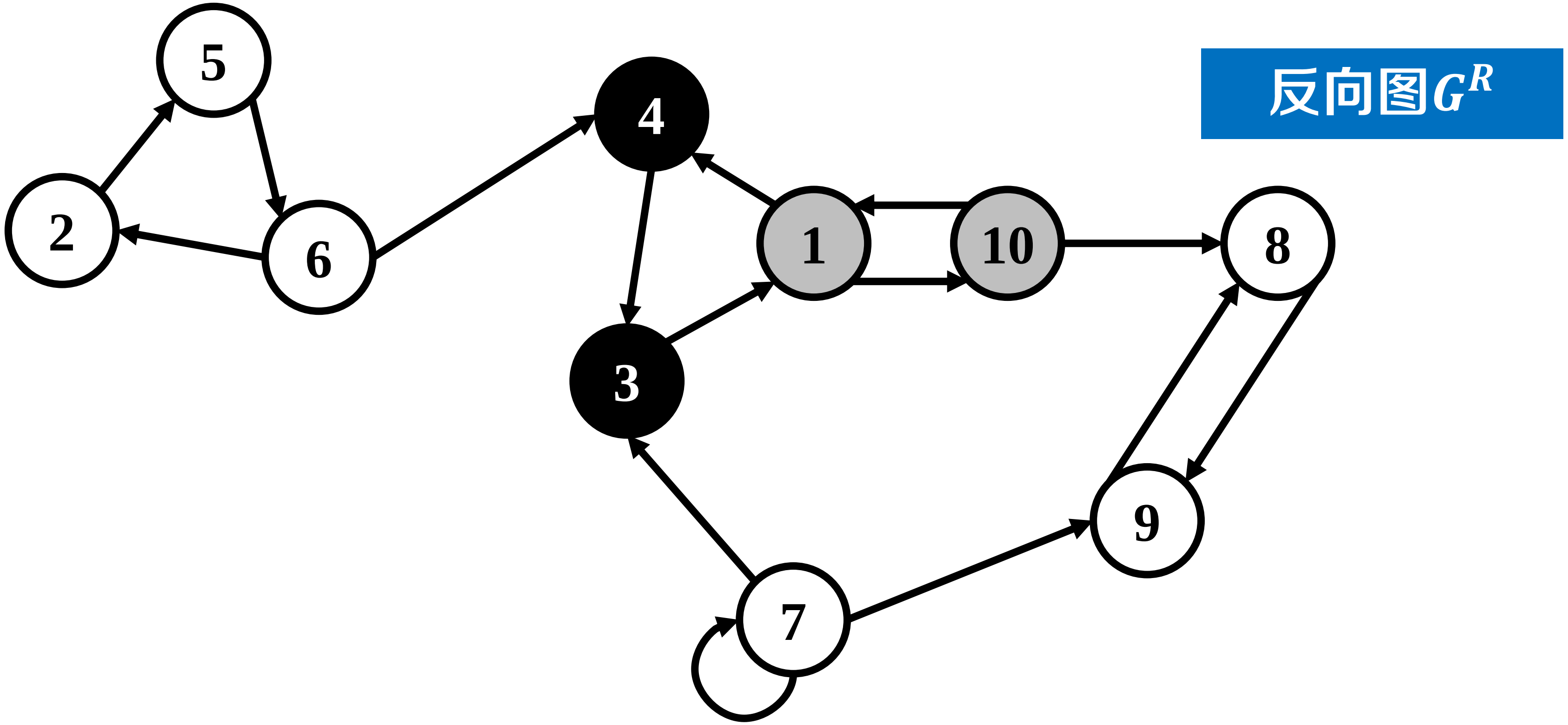
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L	3	4								

算法框架与实例

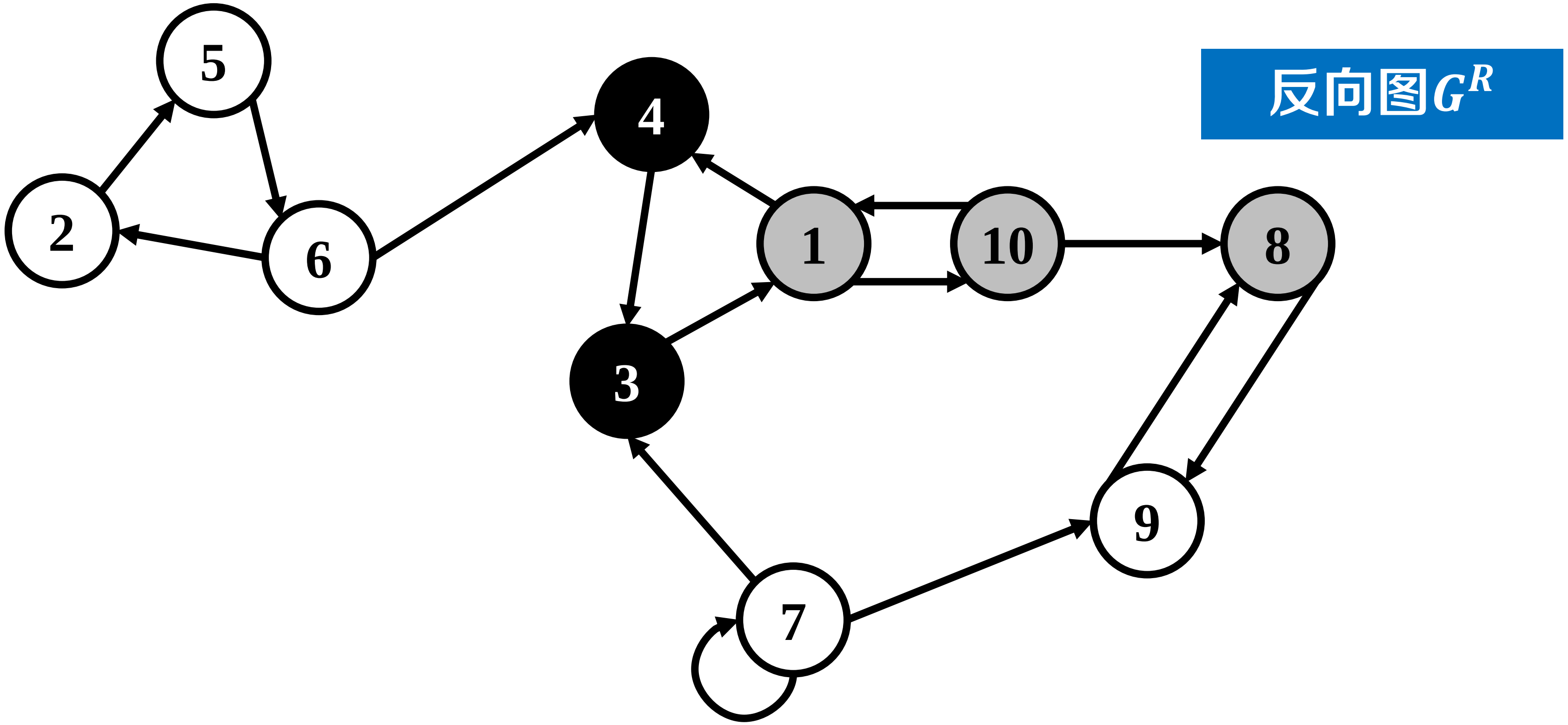
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L	3	4								

算法框架与实例

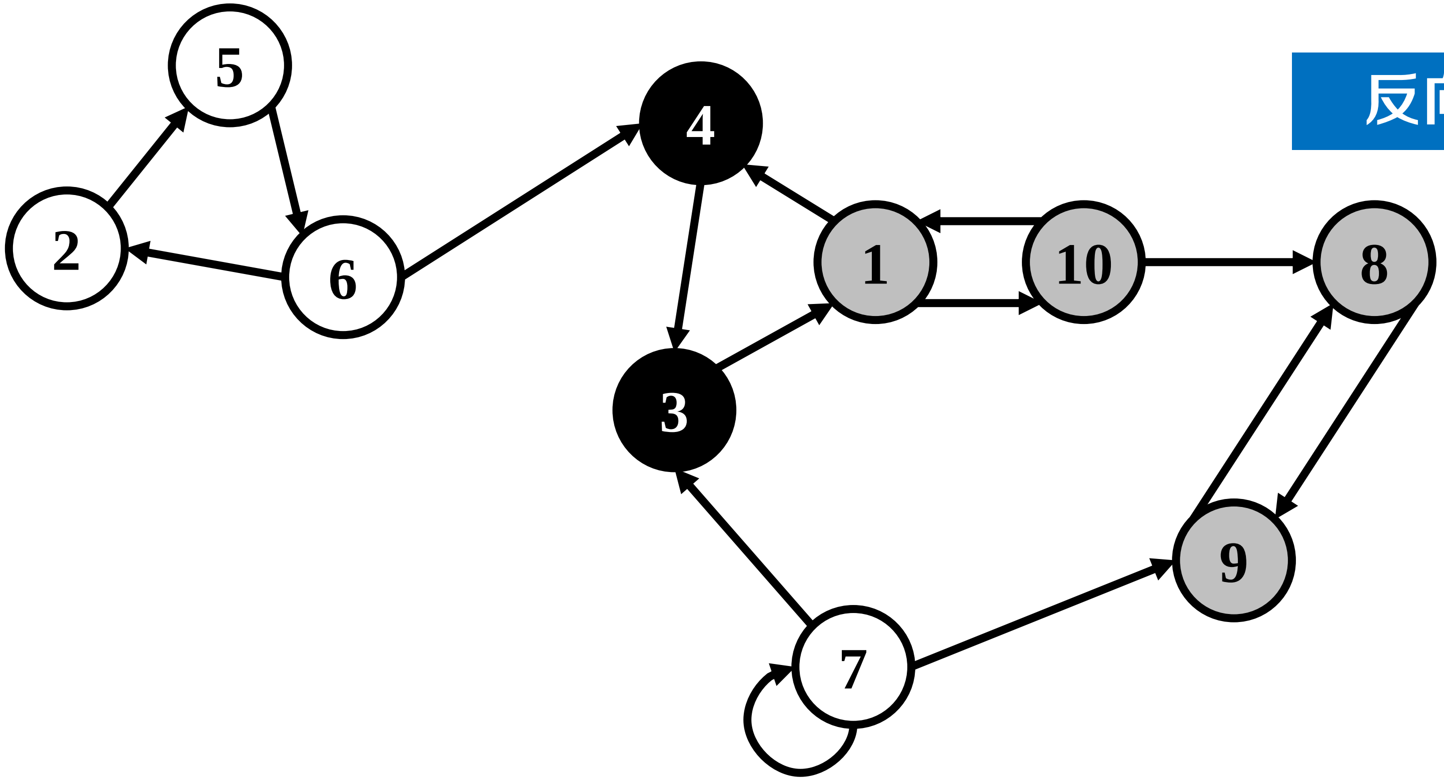
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



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

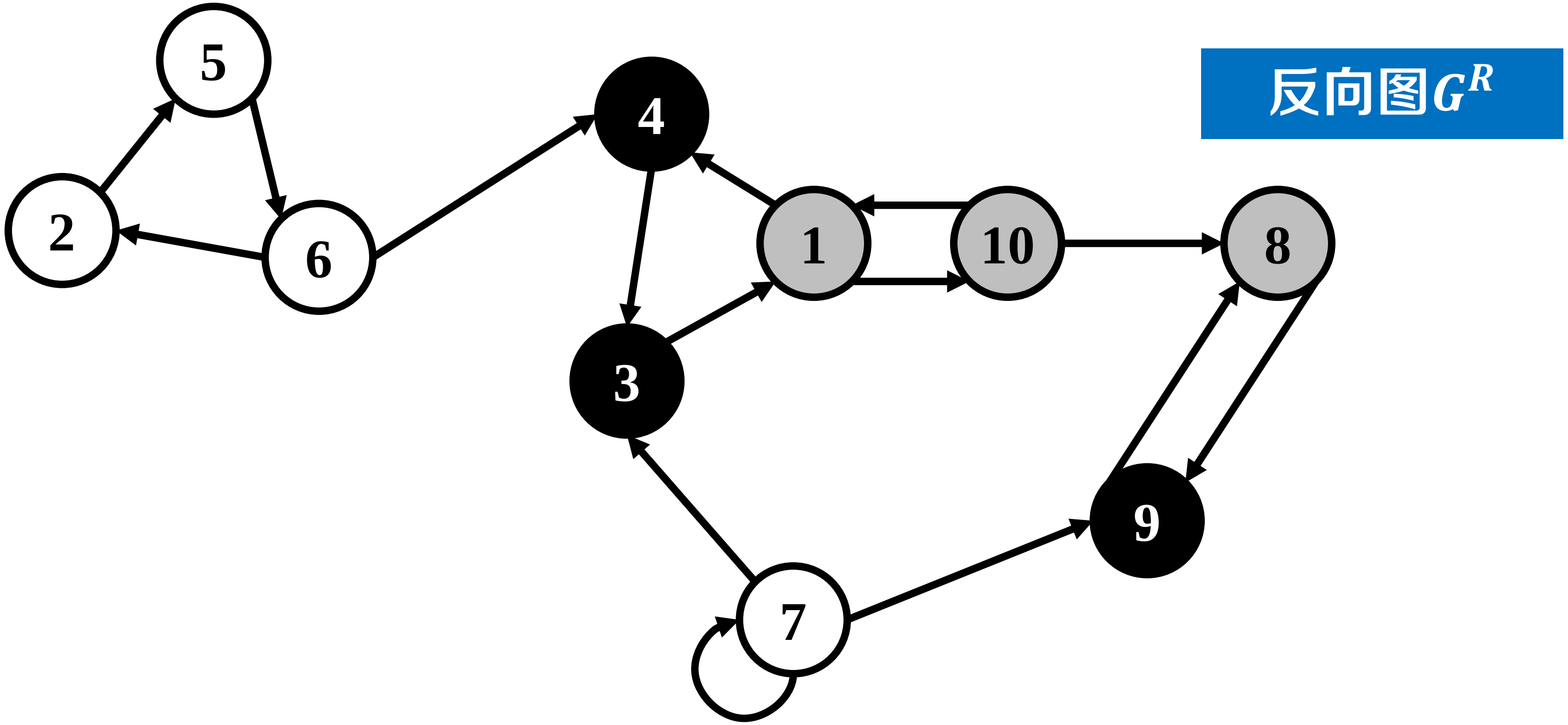


反向图G^R

L	3	4								

算法框架与实例

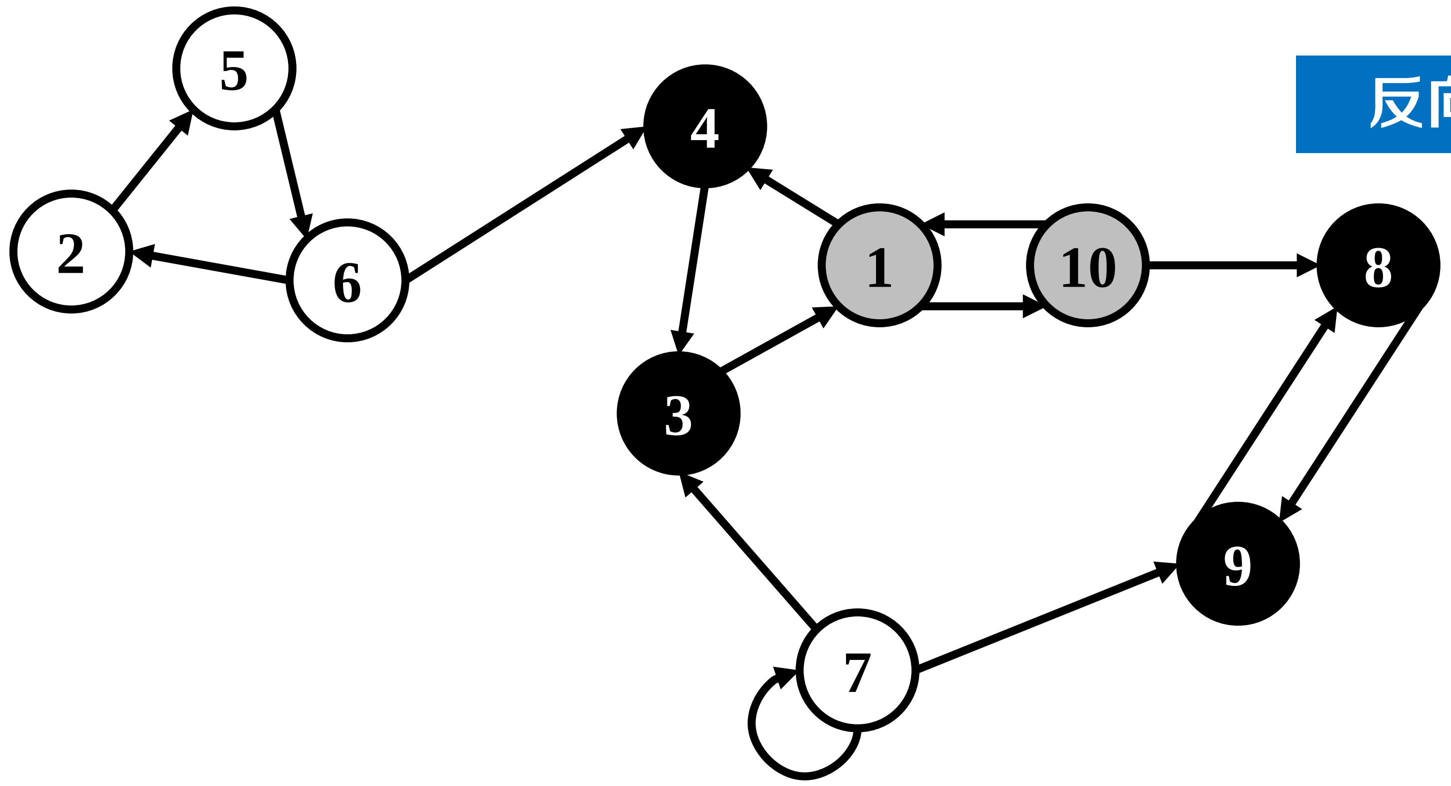
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



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

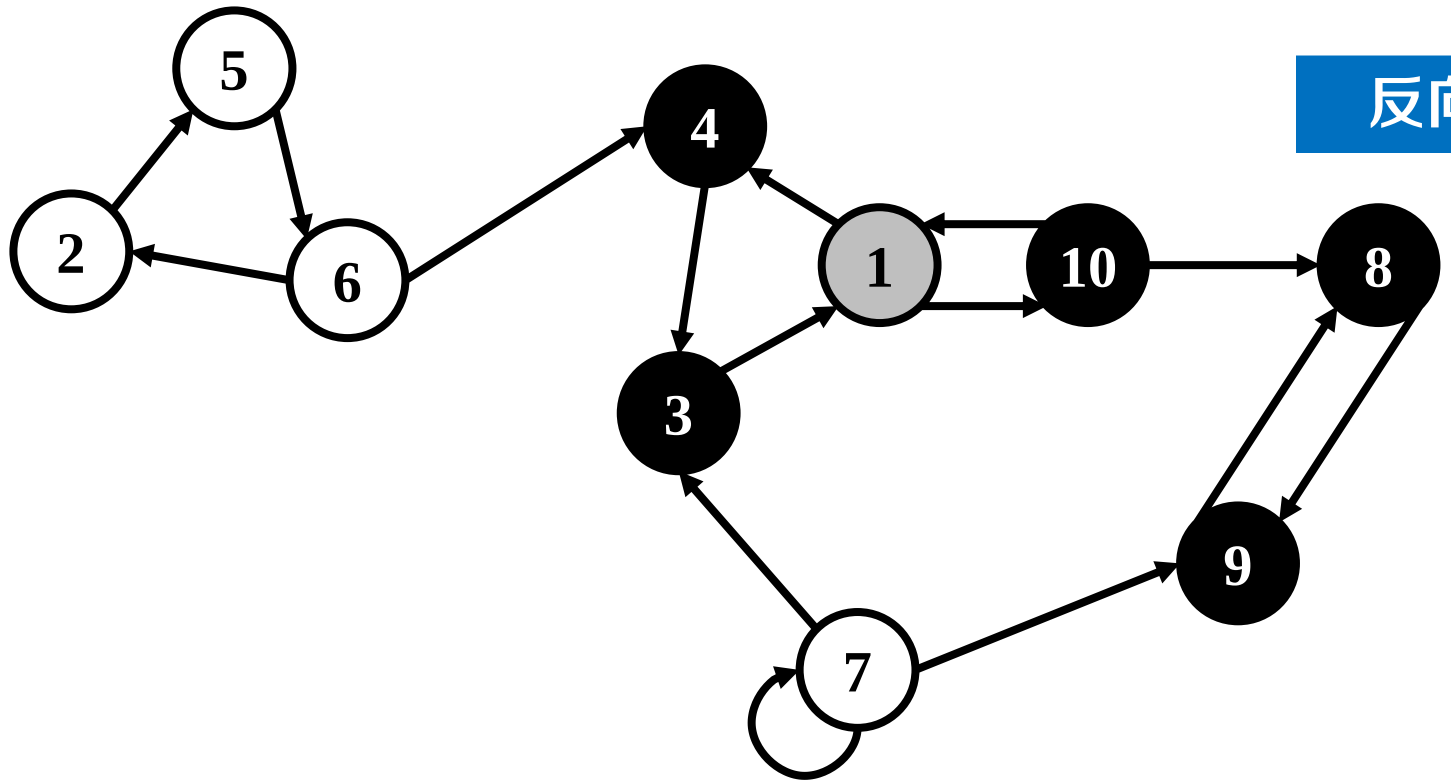


反向图G^R

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

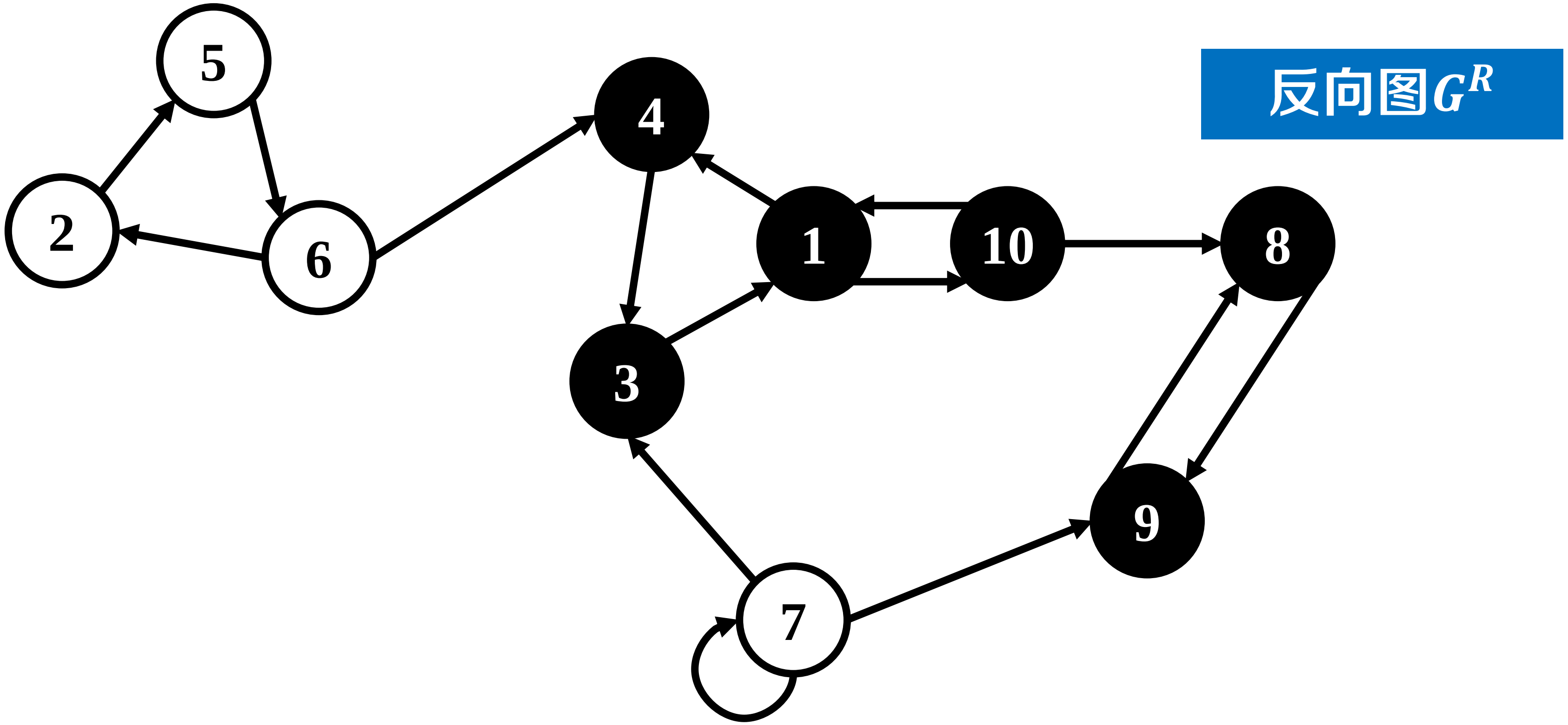


反向图G^R

L	3	4	9	8	10					

算法框架与实例

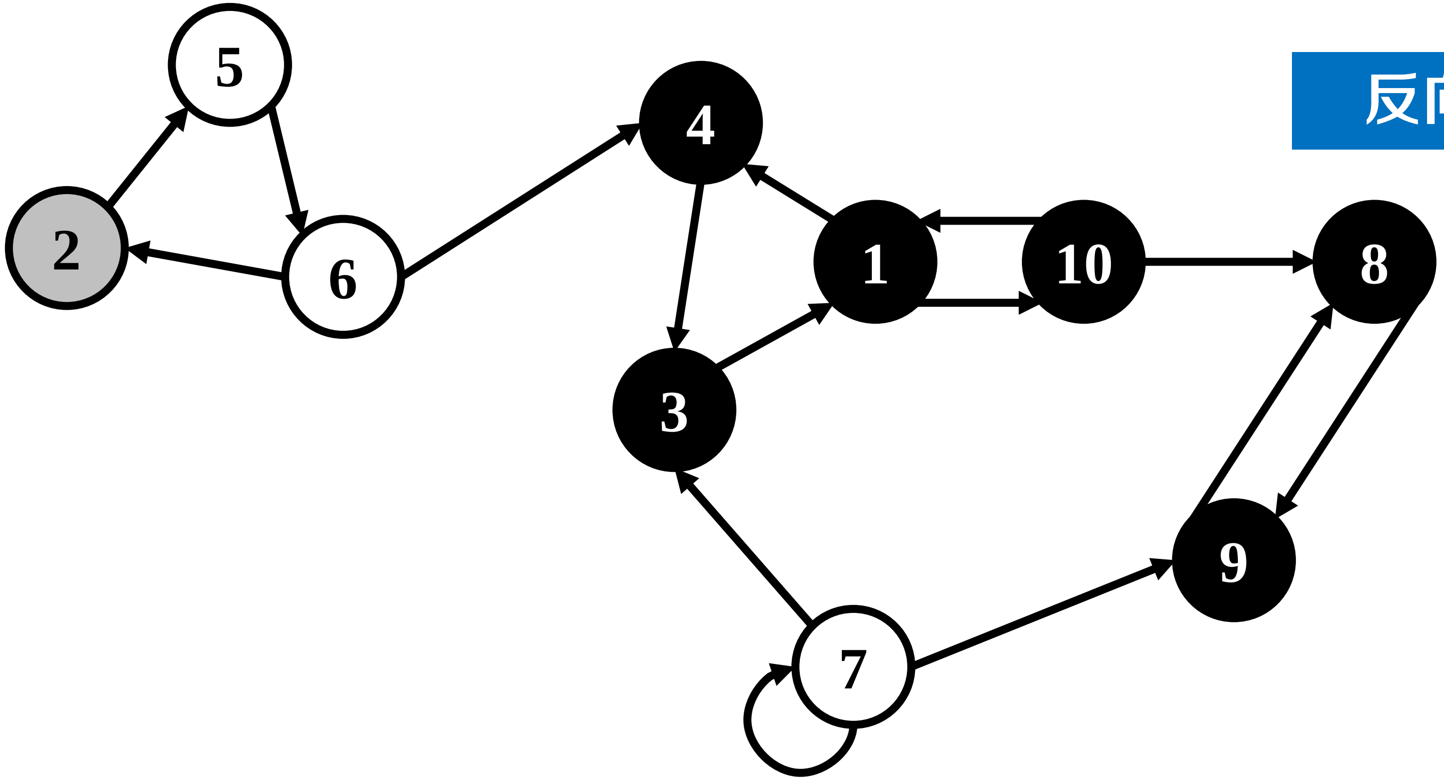
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



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

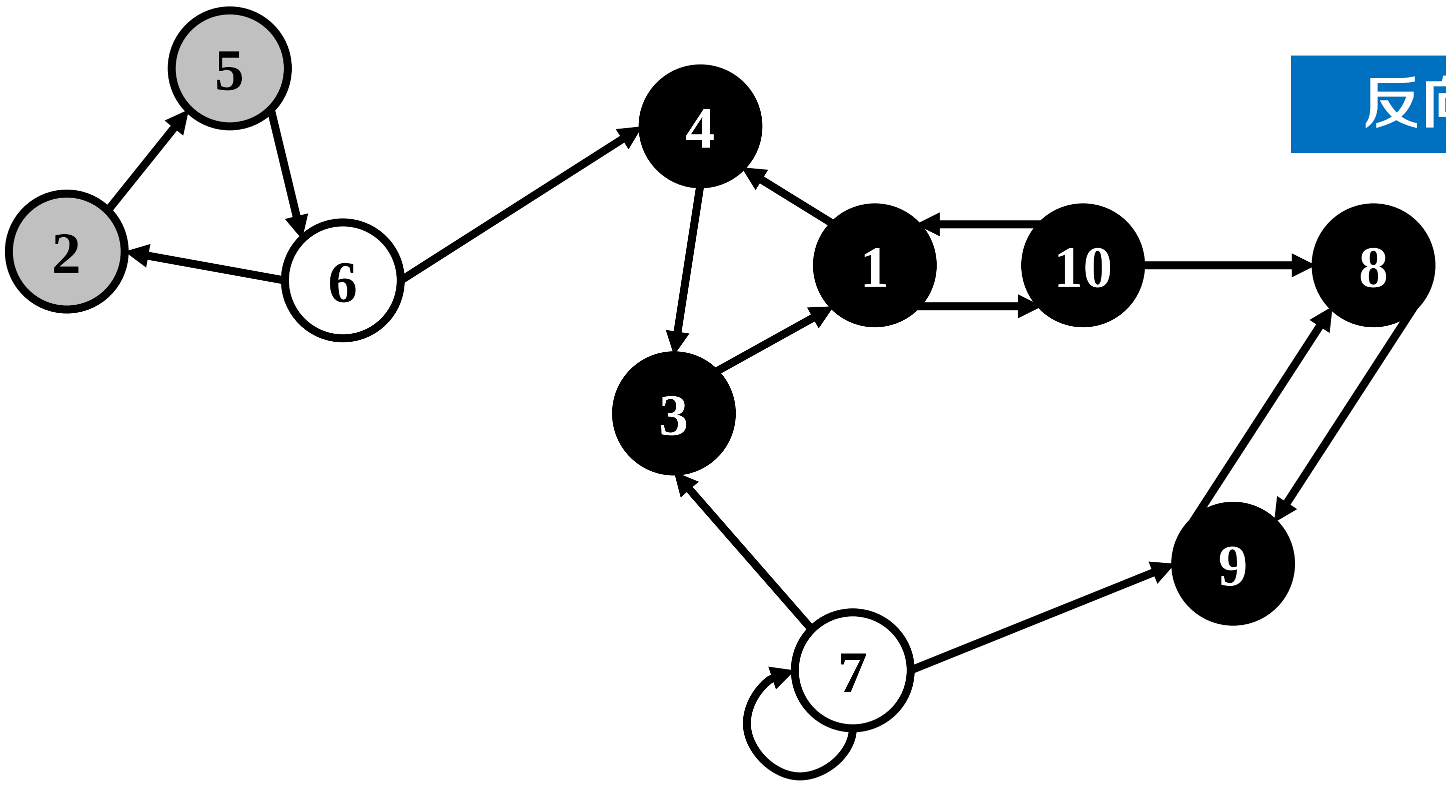


反向图G^R

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

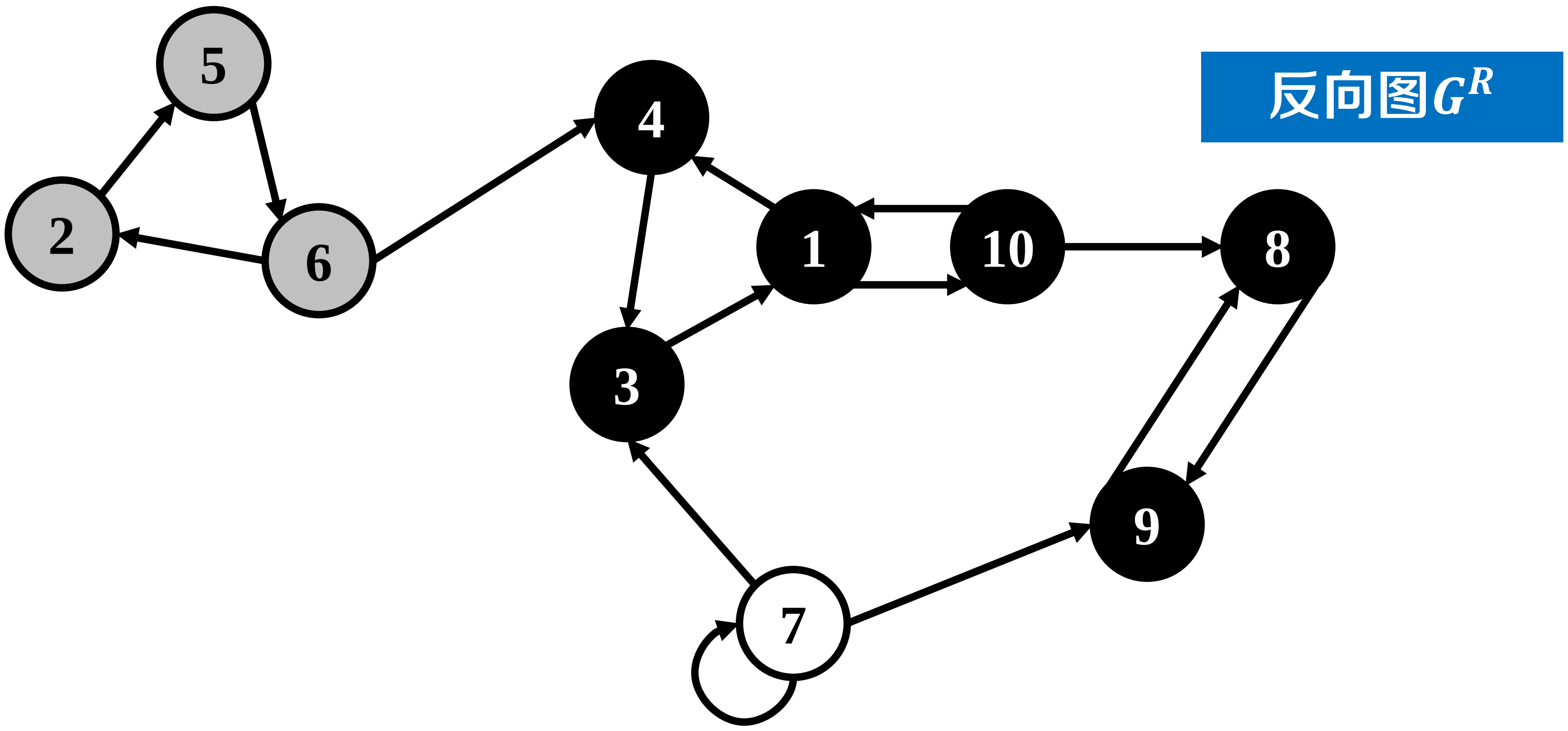


反向图G^R

L	3	4	9	8	10	1				

算法框架与实例

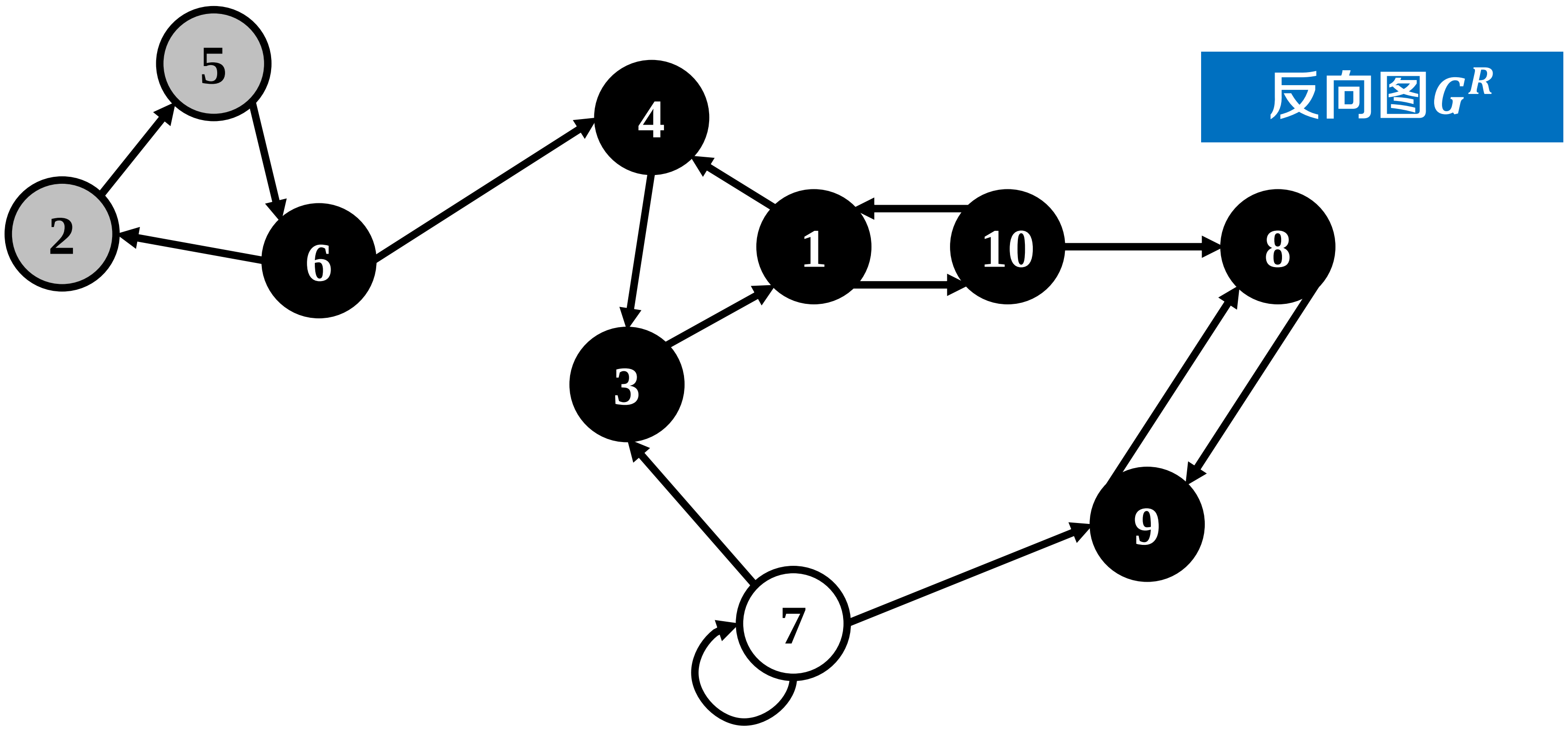
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L	3	4	9	8	10	1				

算法框架与实例

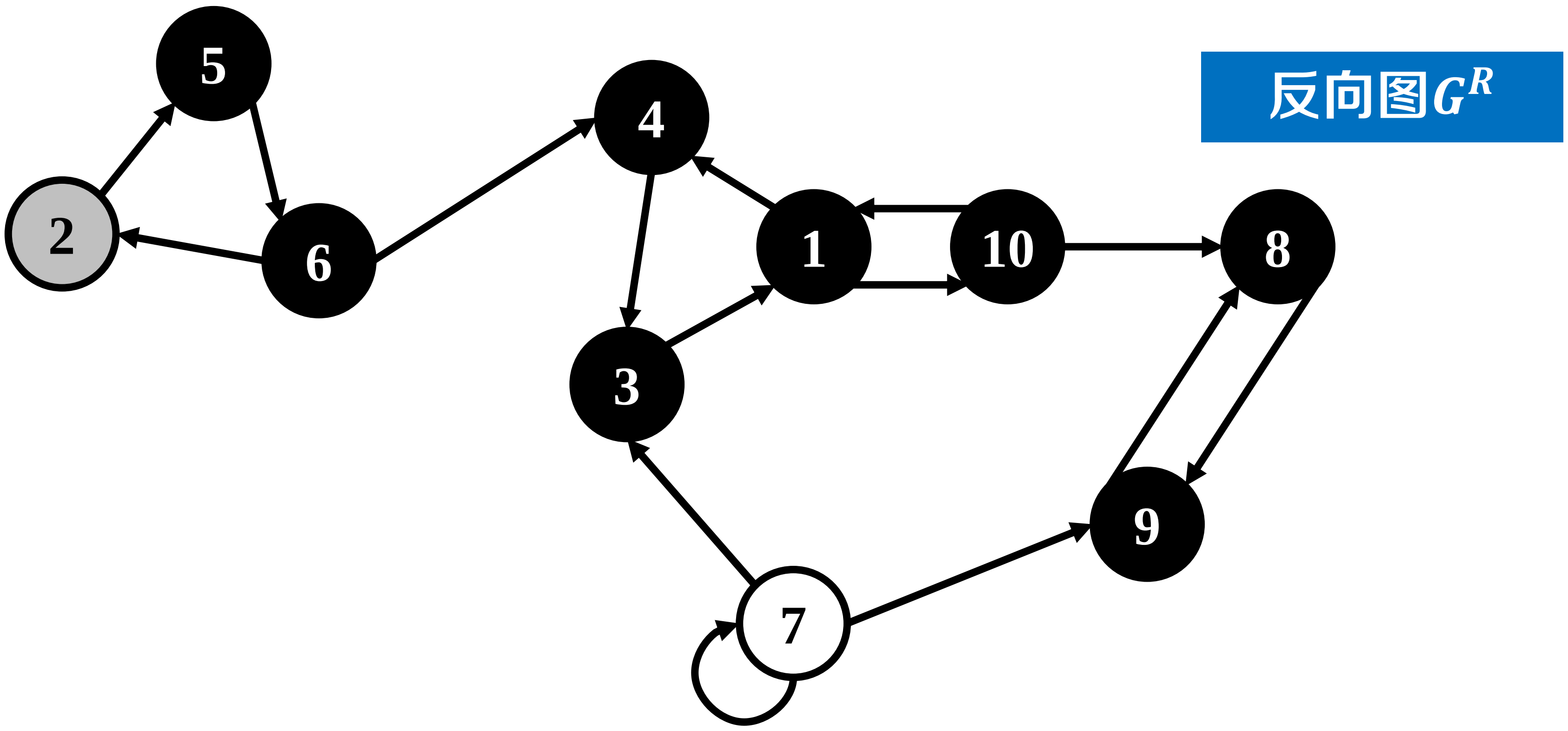
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



L	3	4	9	8	10	1	6			

算法框架与实例

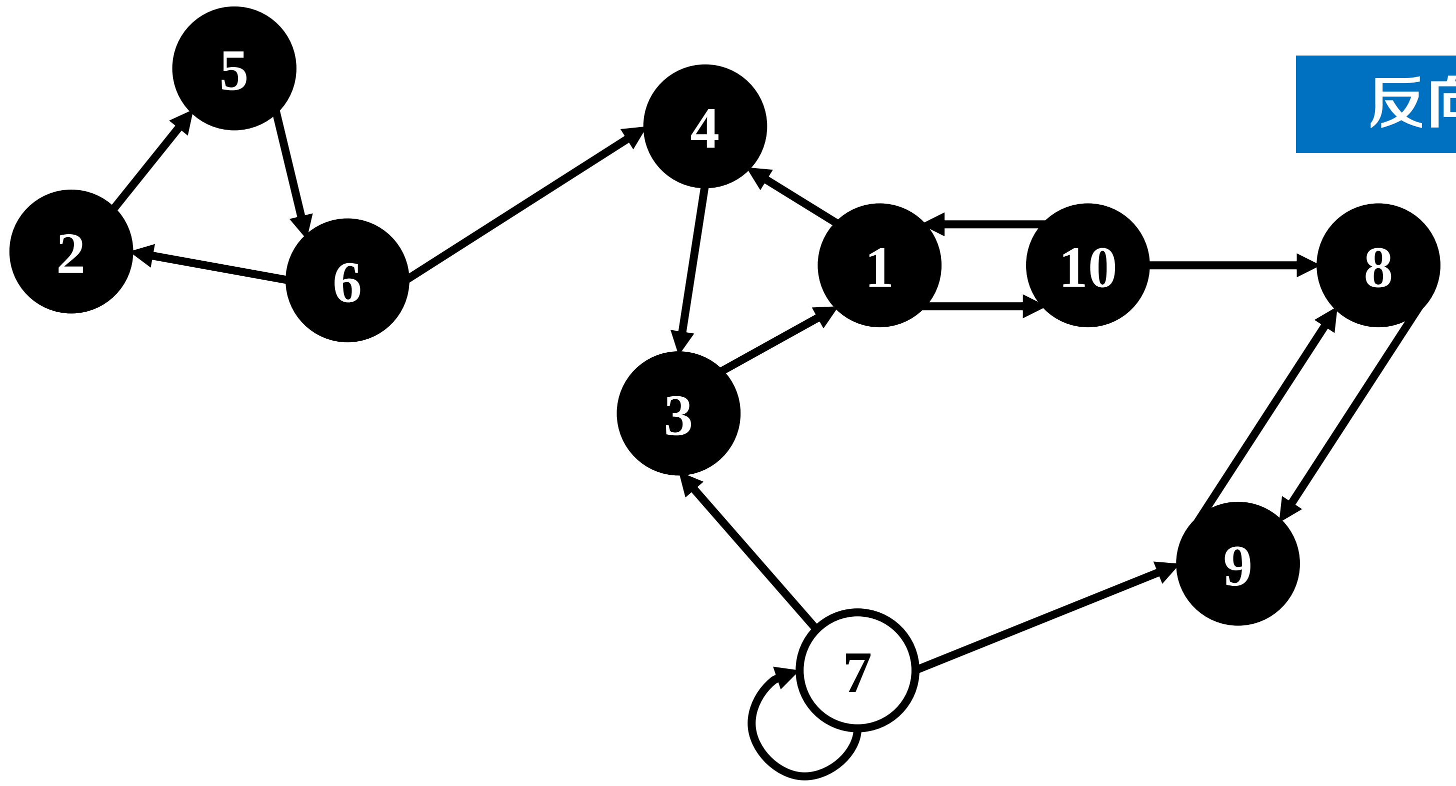
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



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

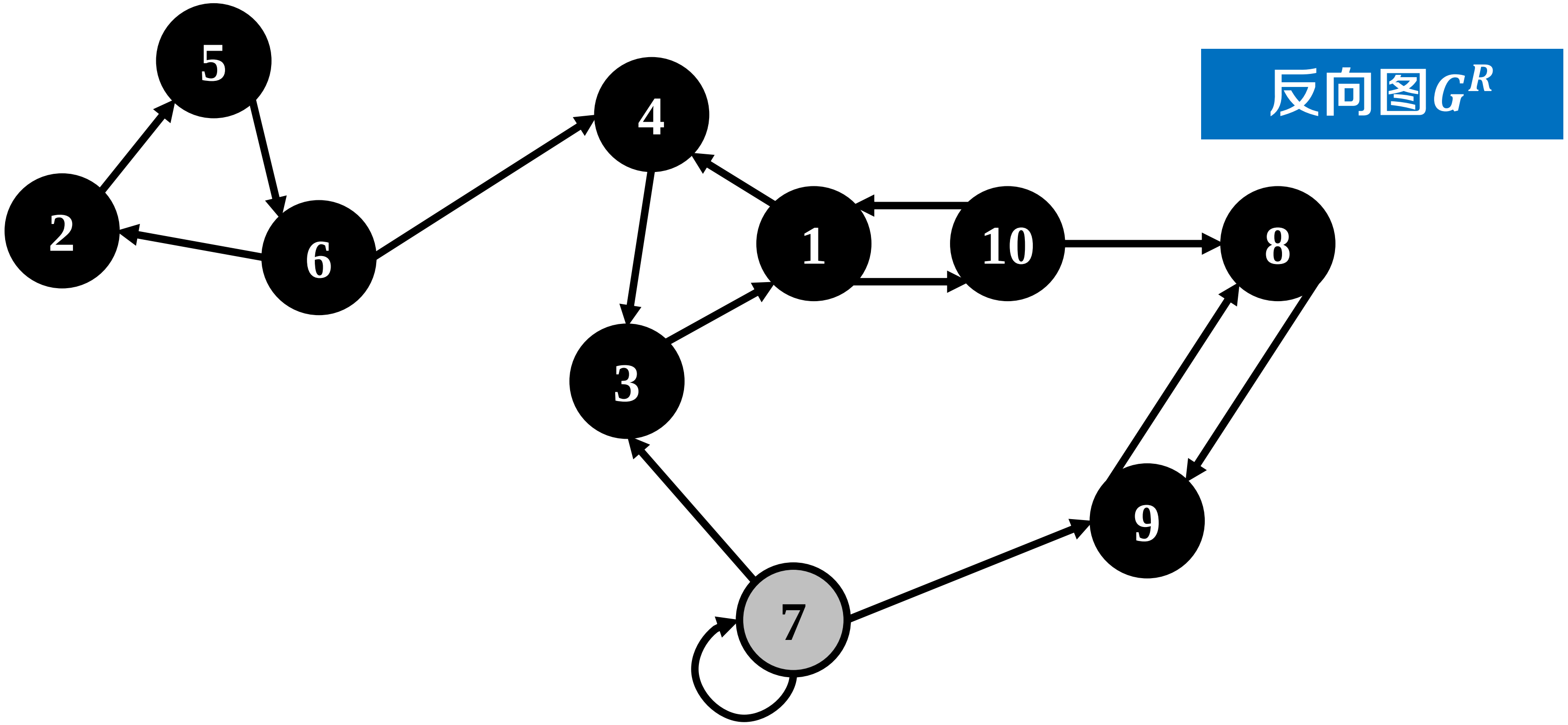


反向图G^R

L	3	4	9	8	10	1	6	5	2	

算法框架与实例

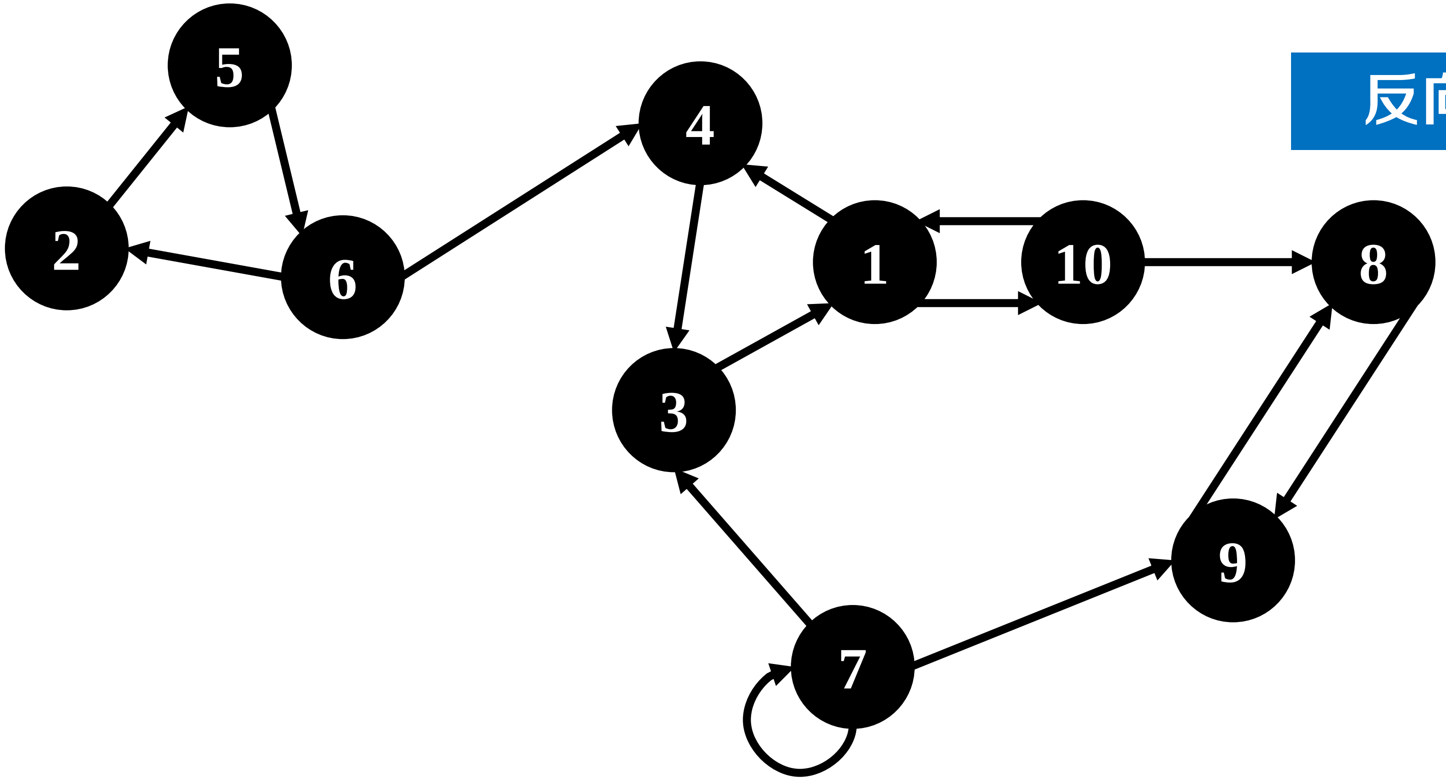
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L



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

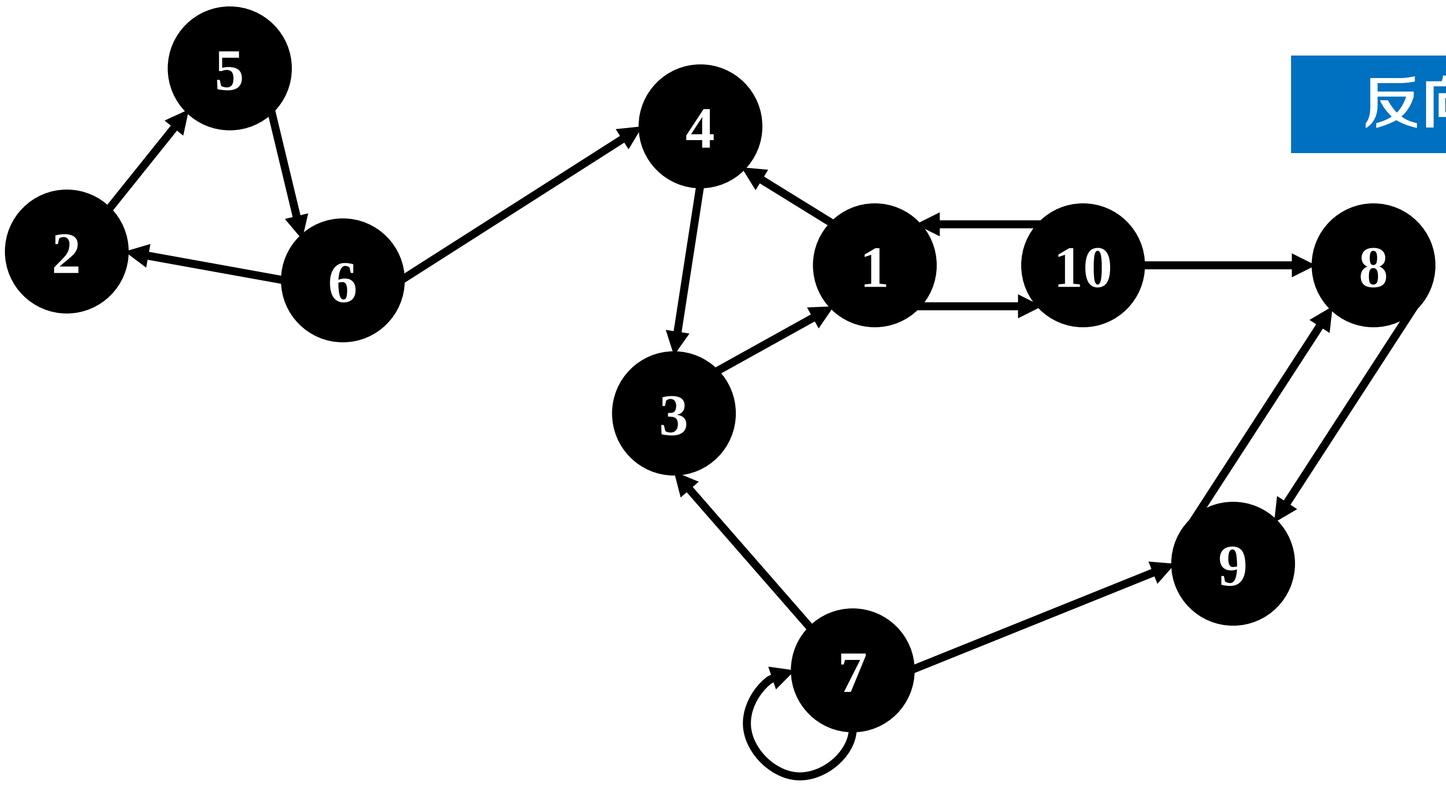


反向图G^R

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

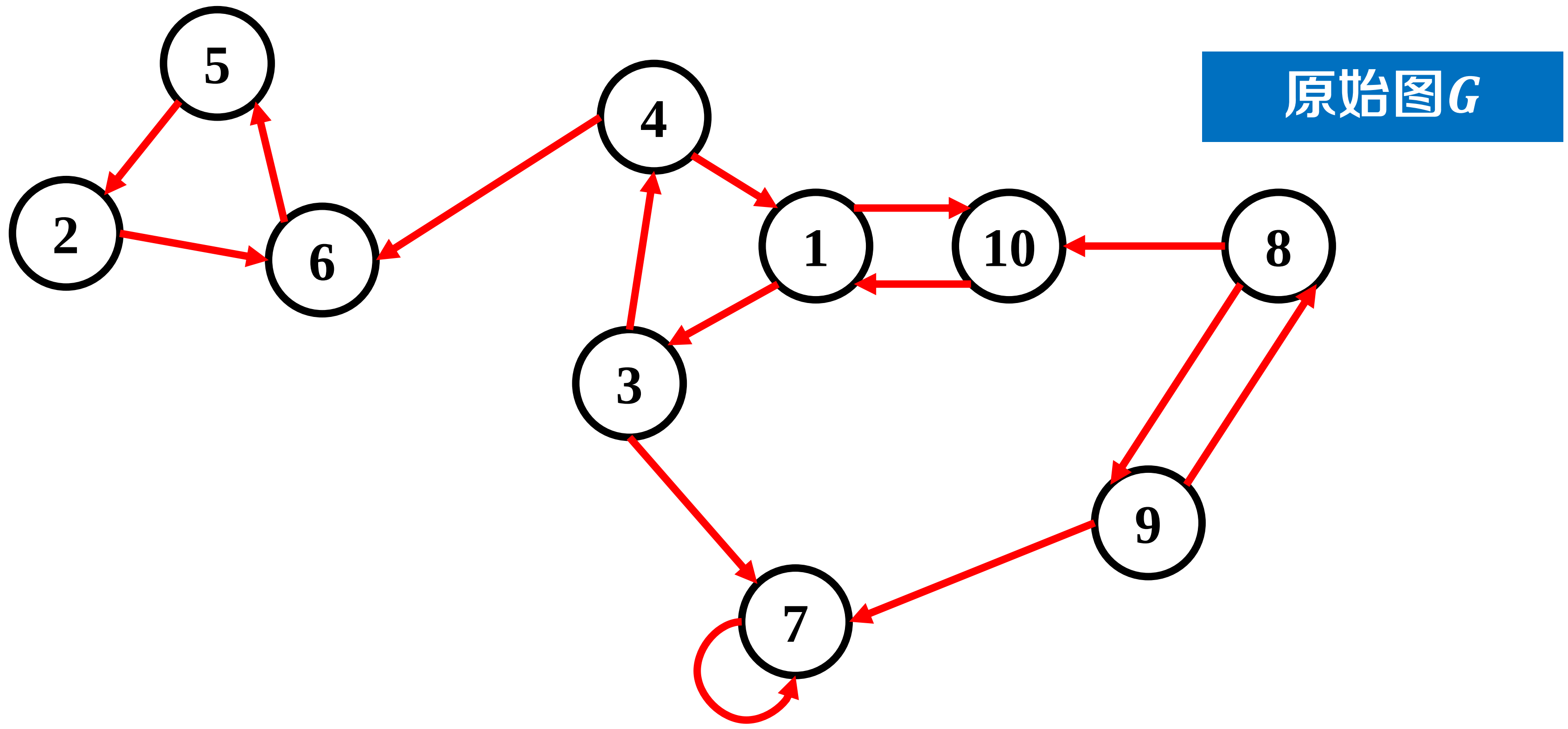


反向图G^R

L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

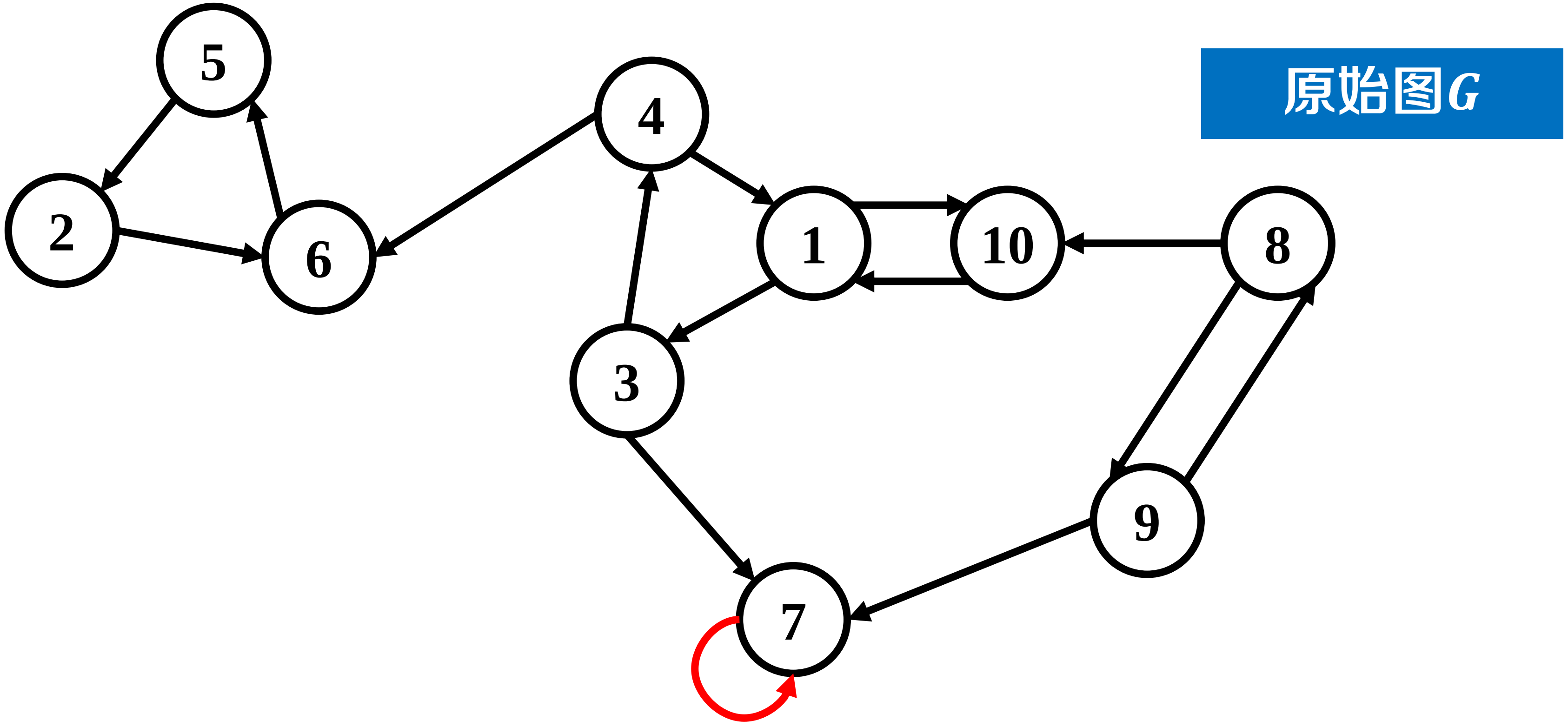
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

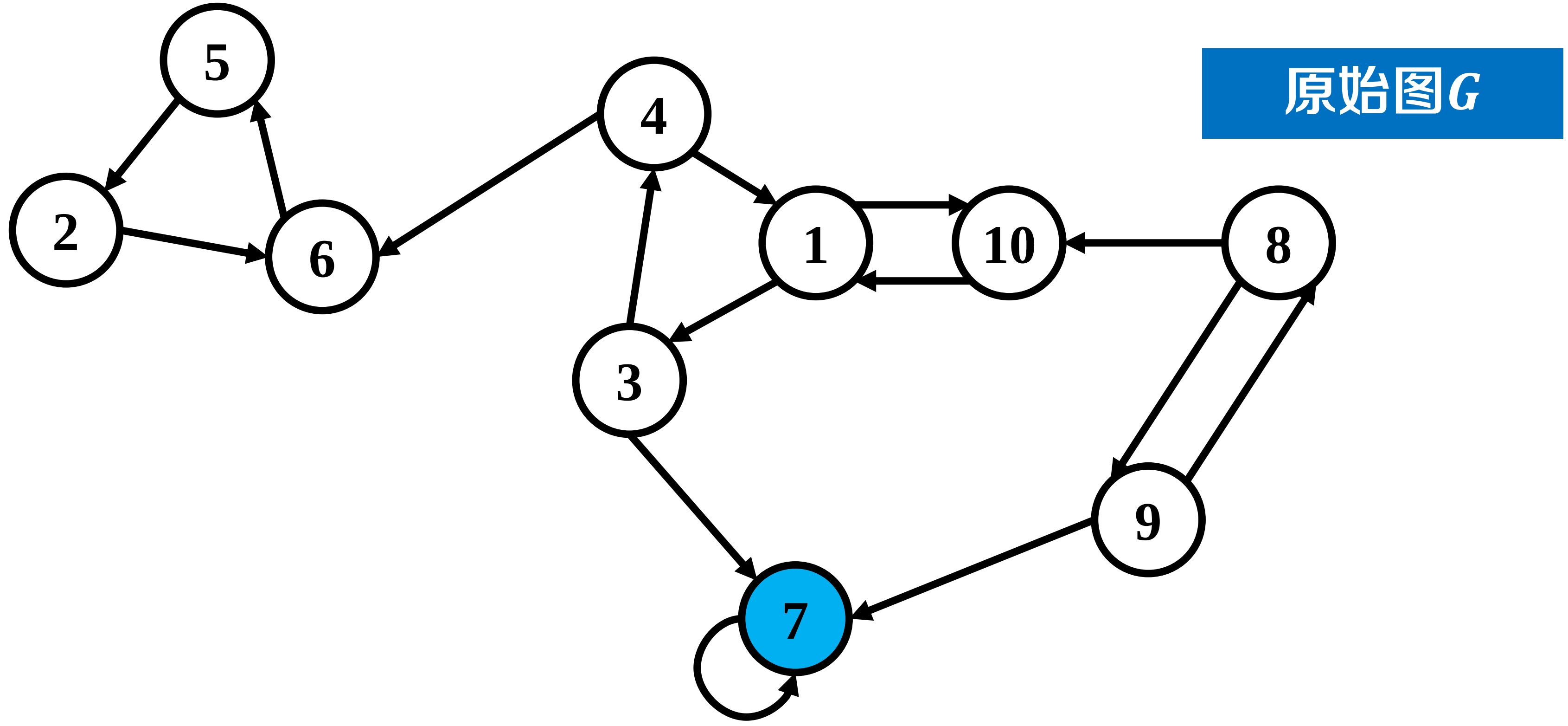
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

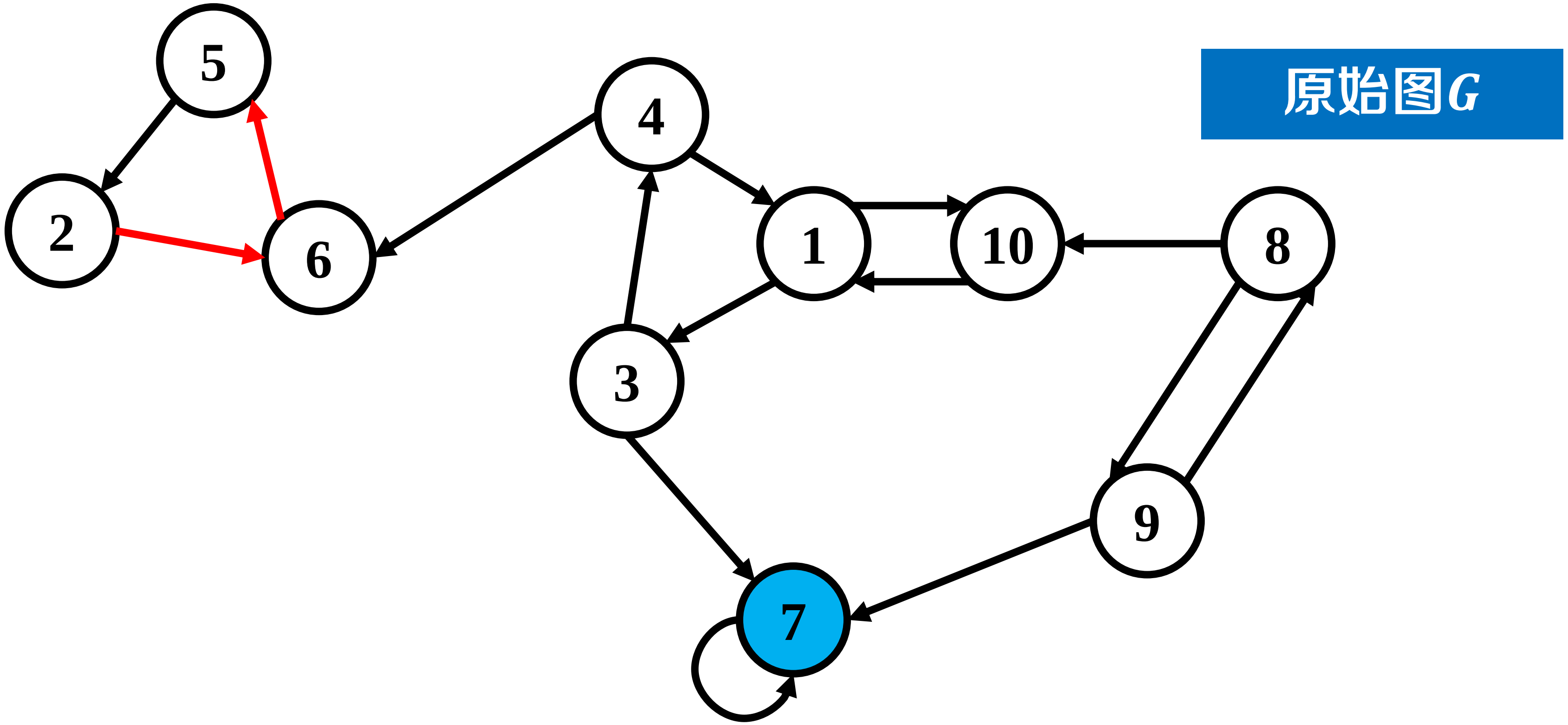
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

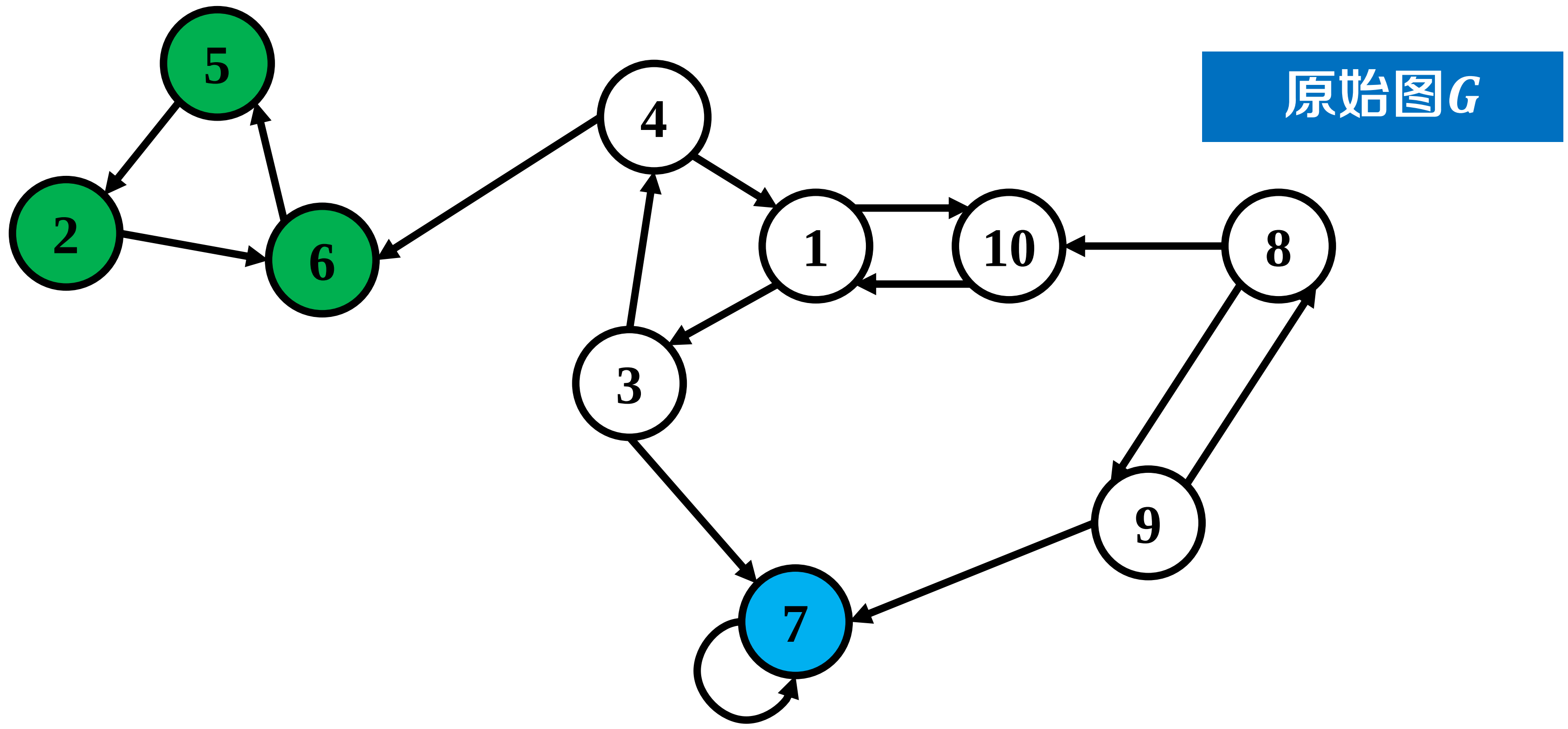
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

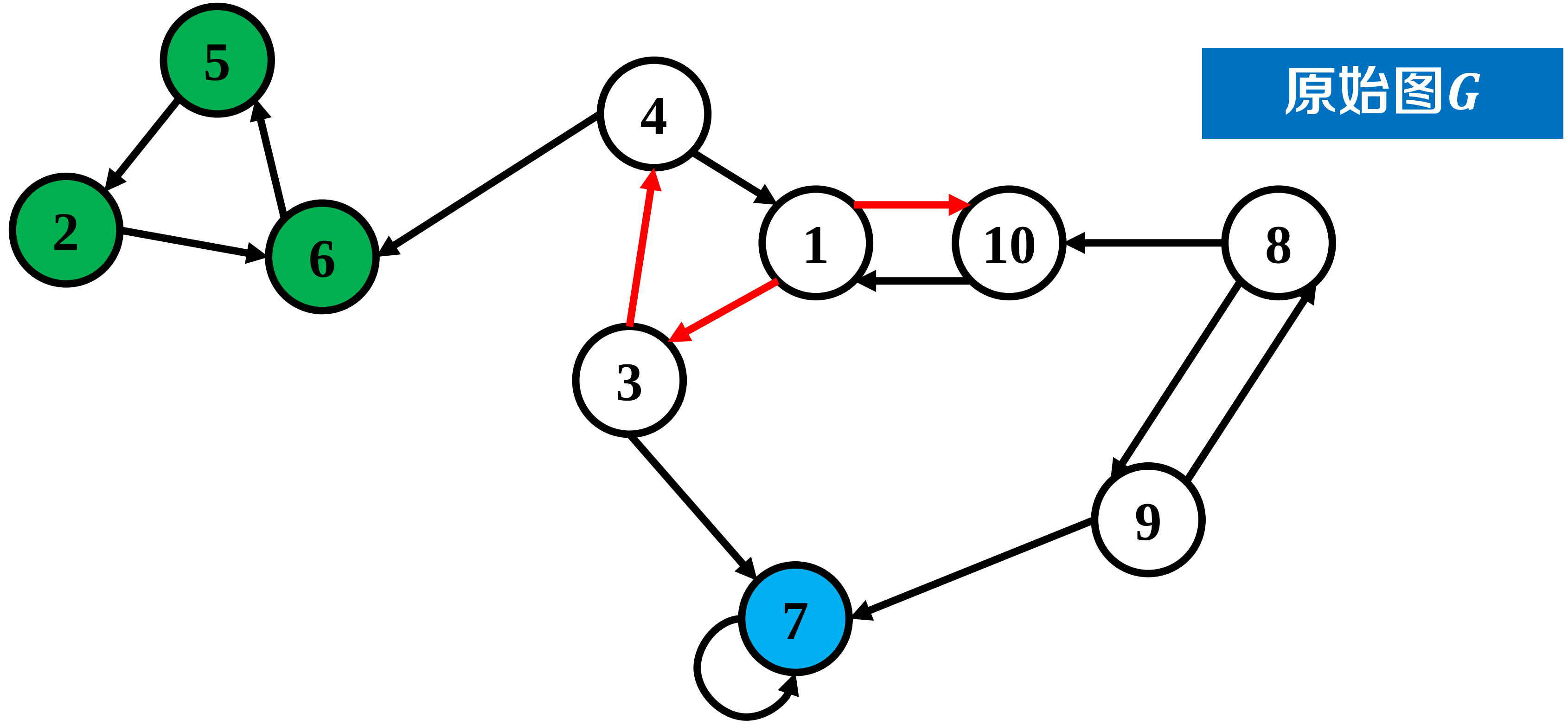
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

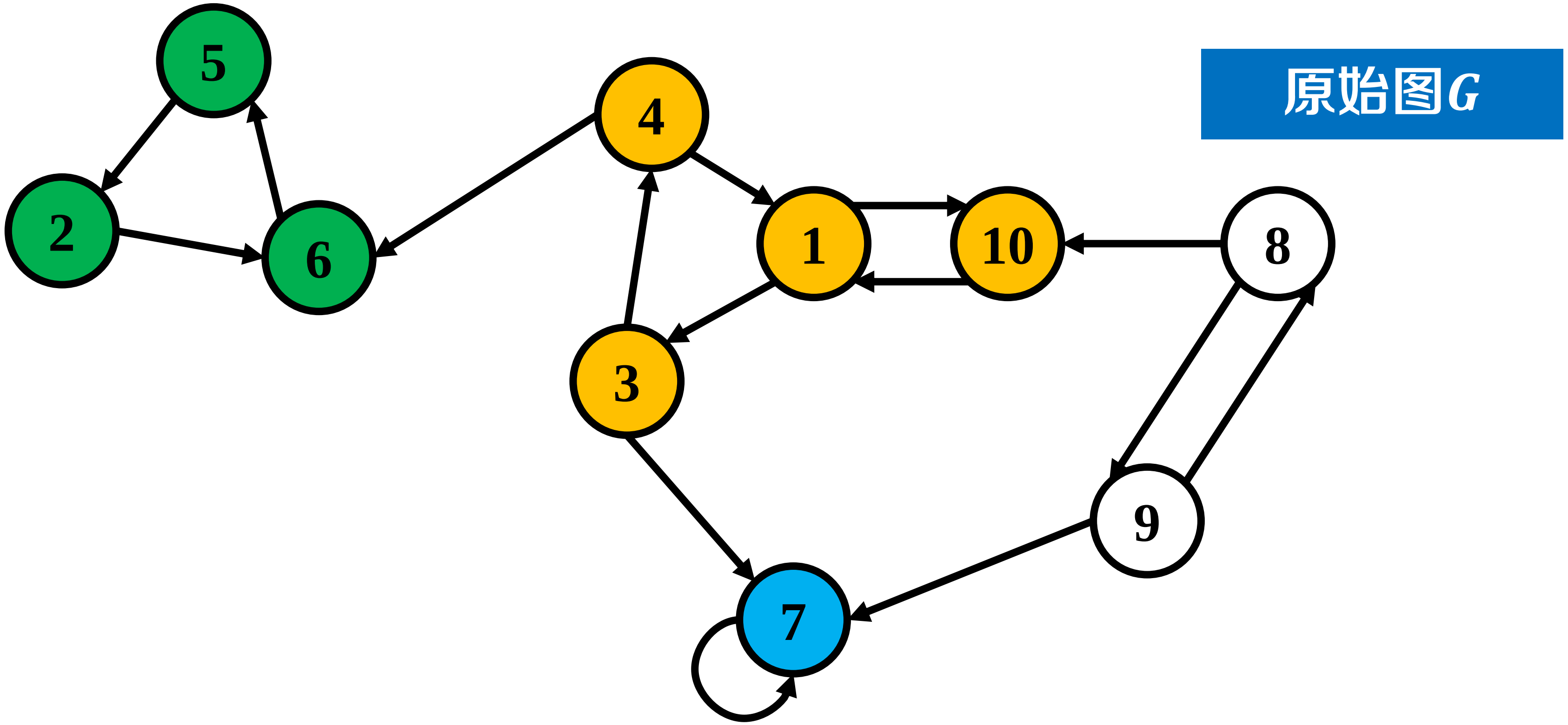
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



L	3	4	9	8	10	1	6	5	2	7

算法框架与实例

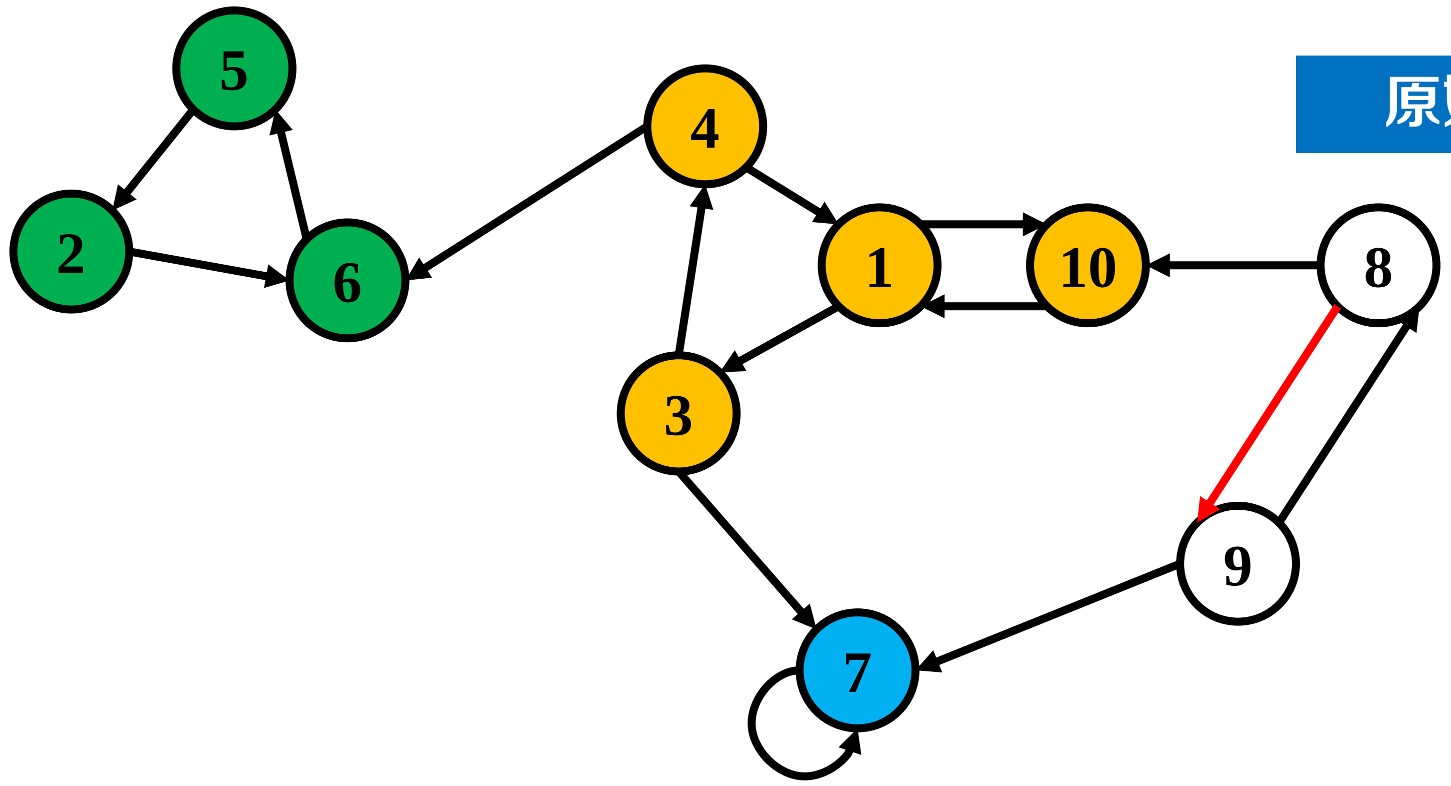
- 步骤1：把边反向，得到反向图G^R
- 步骤2：在G^R上执行DFS，得到顶点完成时刻顺序L
- 步骤3：在G上按L逆序执行DFS，得到强连通分量



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

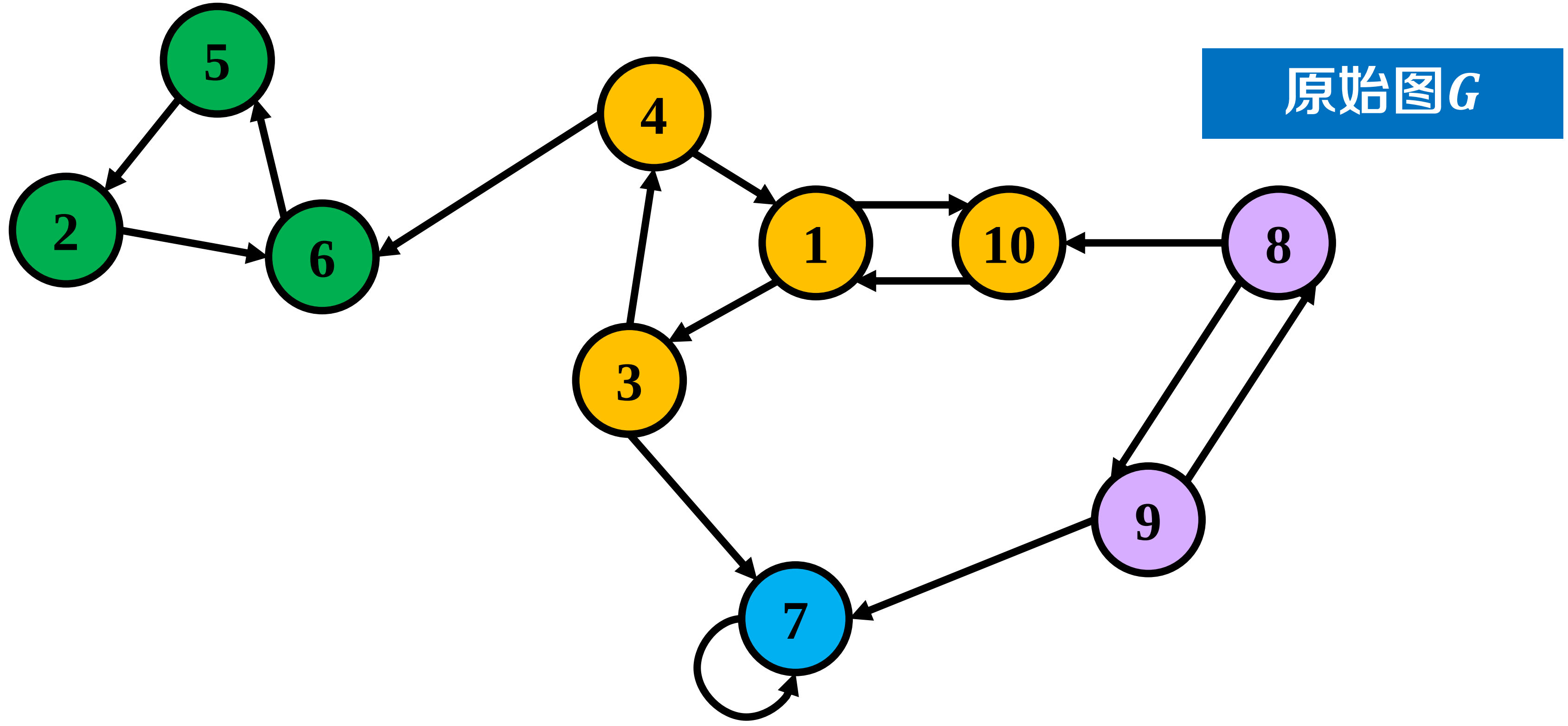


原始图G

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

提纲

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

伪代码

- **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

伪代码

- **Strongly-Connected-Component(*G*)**

输入: 图 G
输出: 强连通分量
$R \leftarrow \{\}$
$G^R \leftarrow G.reverse()$
$L \leftarrow \mathrm{DFS}(G^R)$
$color[1..V] \leftarrow WHITE$
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

伪代码

- **Strongly-Connected-Component(*G*)**

输入: 图 G
输出: 强连通分量
$R \leftarrow \{\}$
$G^R \leftarrow G.reverse()$
$L \leftarrow \mathrm{DFS}(G^R)$
$color[1..V] \leftarrow WHITE$
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

伪代码

- **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?

伪代码

- **DFS(G)**

```
输入: 图 G
新建数组 color[1..V], L[1..V]
for v ∈ V do
|   color[v] ← WHITE
end
for v ∈ V do
|   if color[v] = WHITE then
|   |   L′ ← DFS-Visit(G, v)
|   |   向L结尾追加L′
|   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] ← GRAY
for w ∈ G.Adj[v] do
|   if color[w] = WHITE then
|   |   L ← DFS-Visit(G, w)
|   end
end
color[v] ← BLACK
向L结尾追加顶点v
return L
```

顶点按完成时刻排列

复杂度分析

- **Strongly-Connected-Component(G)**

输入: 图 G
输出: 强连通分量

$R \leftarrow \{\}$
$G^R \leftarrow G.reverse()$ $O(|V|+|E|)$
$L \leftarrow \mathrm{DFS}(G^R)$ $O(|V|+|E|)$ 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**
    $L_{scc} \leftarrow \mathrm{DFS\text{-}Visit}(G, u)$
    $R \leftarrow R \cup set(L_{scc})$
  end
end
return R

(for 循环部分) $O(|V|+|E|)$ 第二次深度优先搜索

时间复杂度：$O(|V|+|E|)$

提纲

问题背景与定义

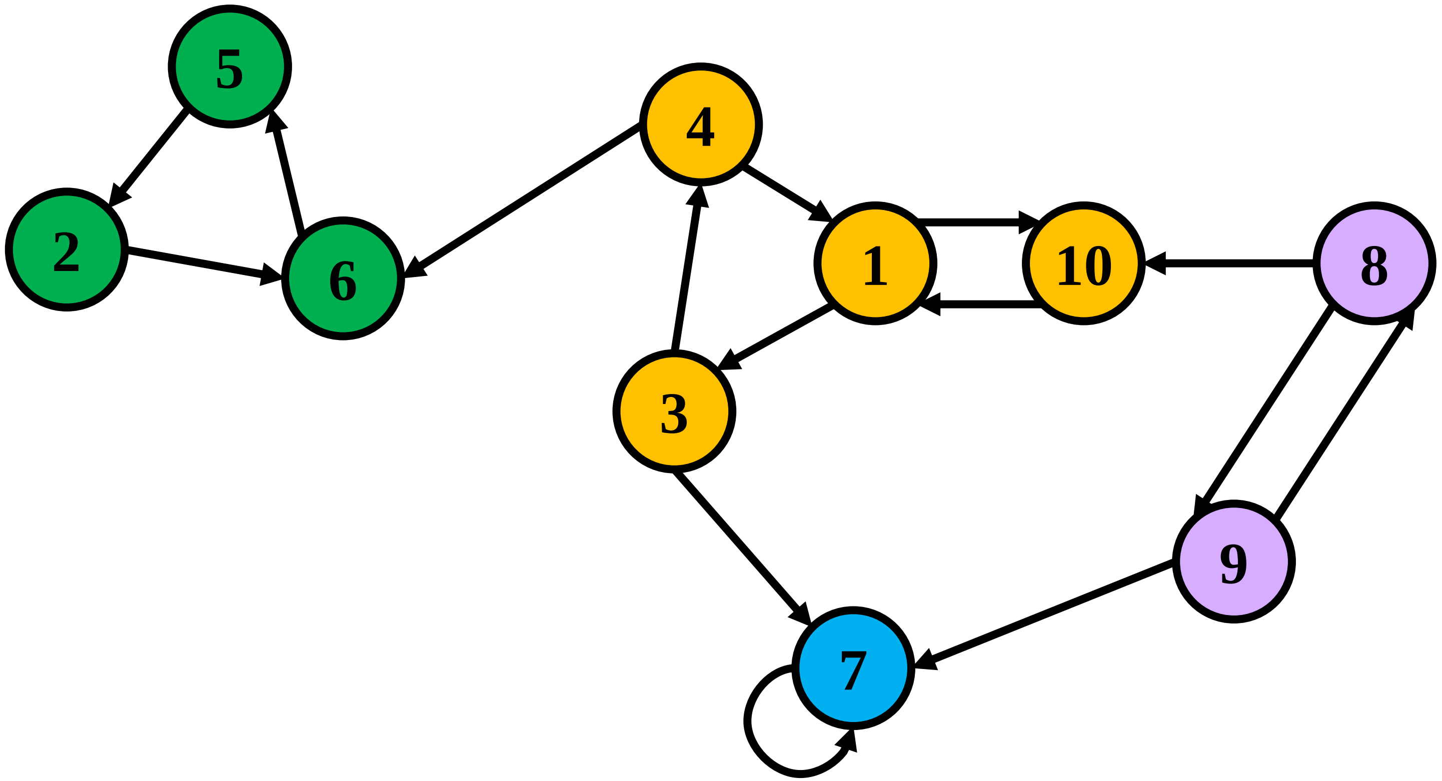
算法框架与实例

伪代码与复杂度

算法正确性证明

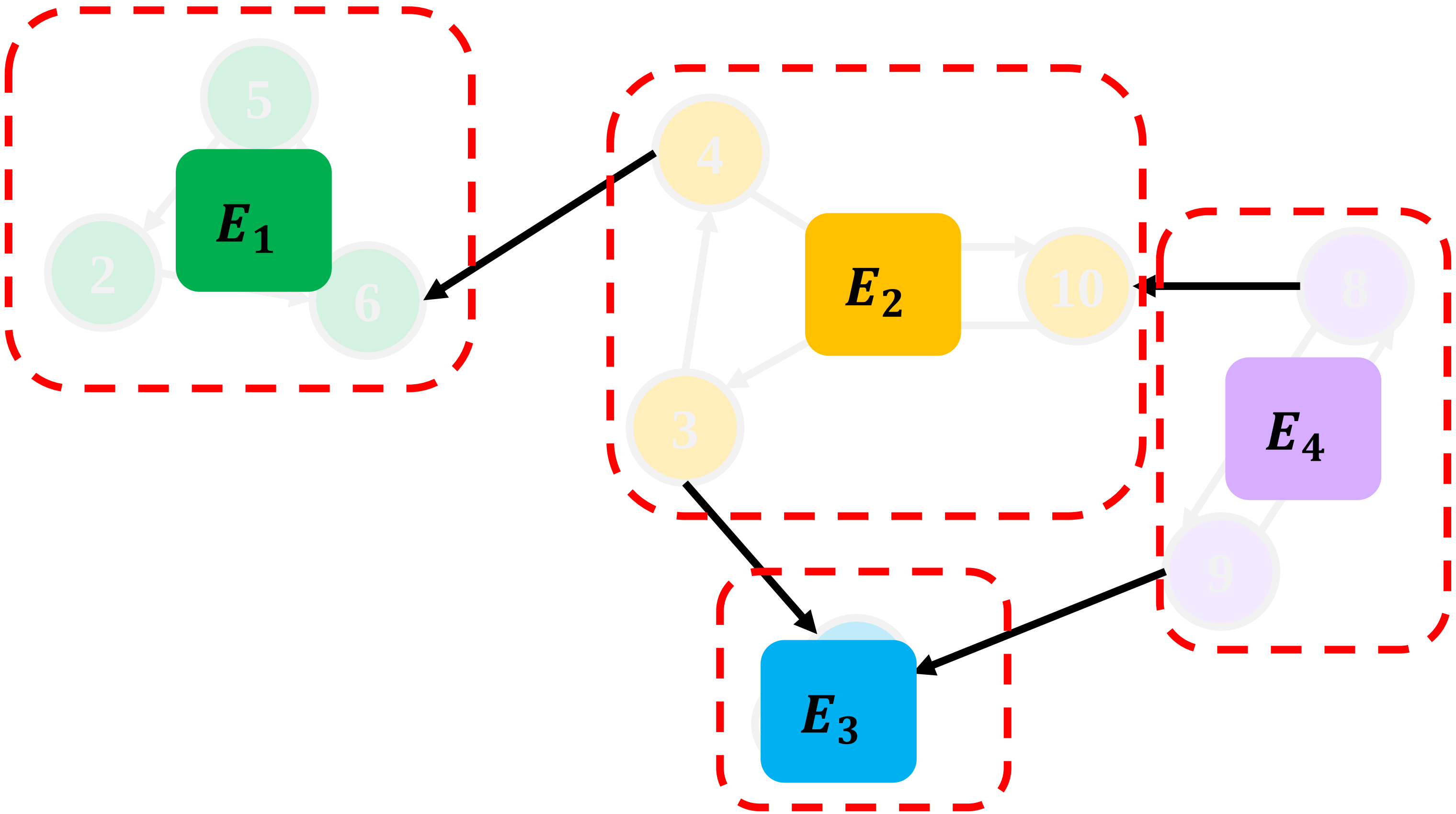
正确性证明

- 强连通分量图G^{SCC}：把强连通分量看作一个点，得到有向图



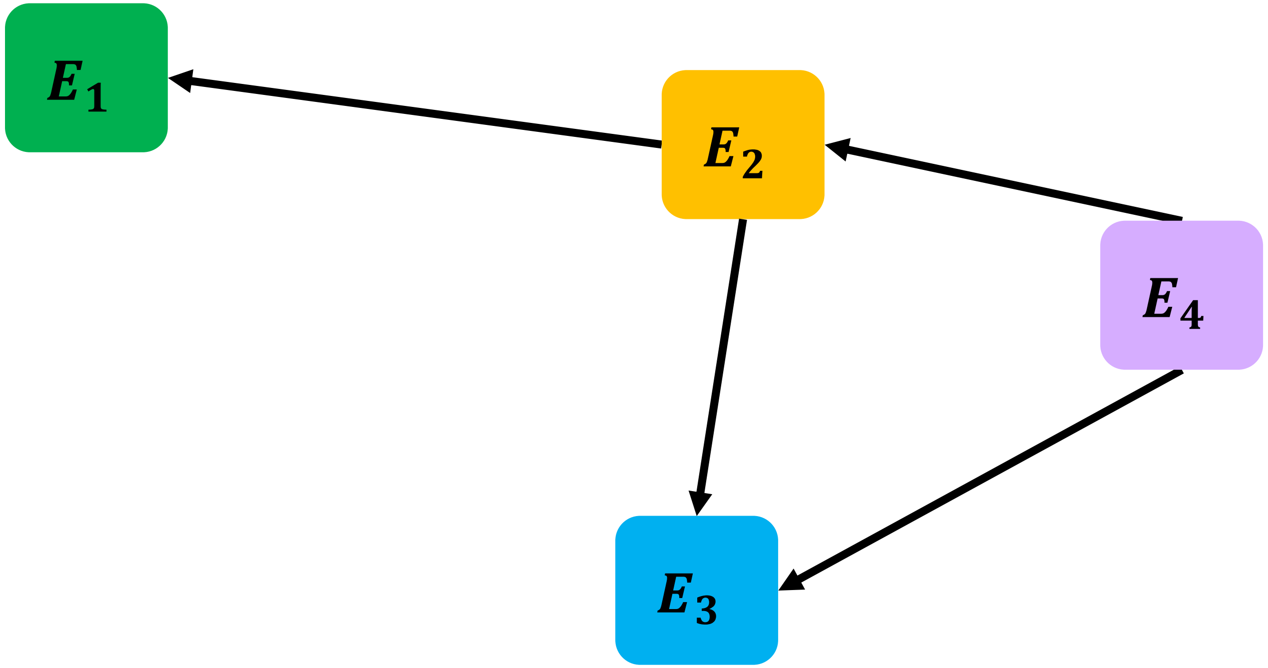
正确性证明

- 强连通分量图G^{SCC}：把强连通分量看作一个点，得到有向图



正确性证明

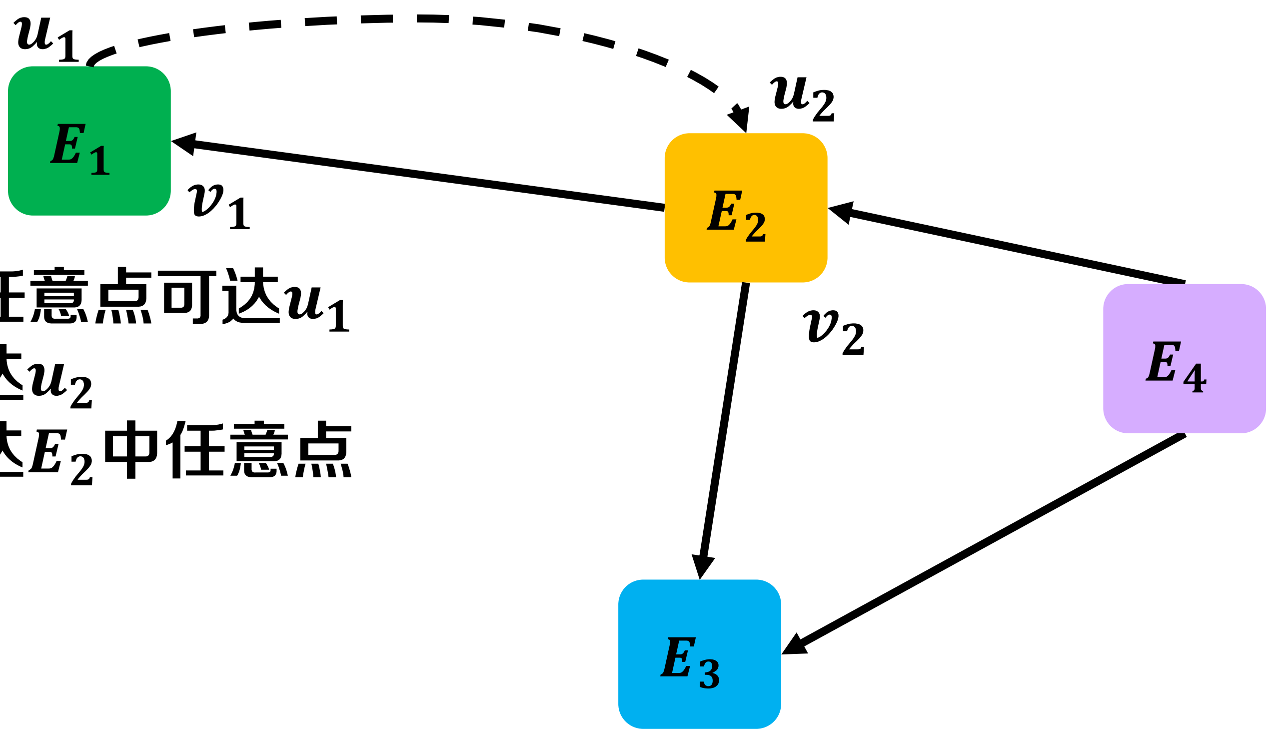
- 强连通分量图G^{SCC}：把强连通分量看作一个点，得到有向图



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
  - 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。

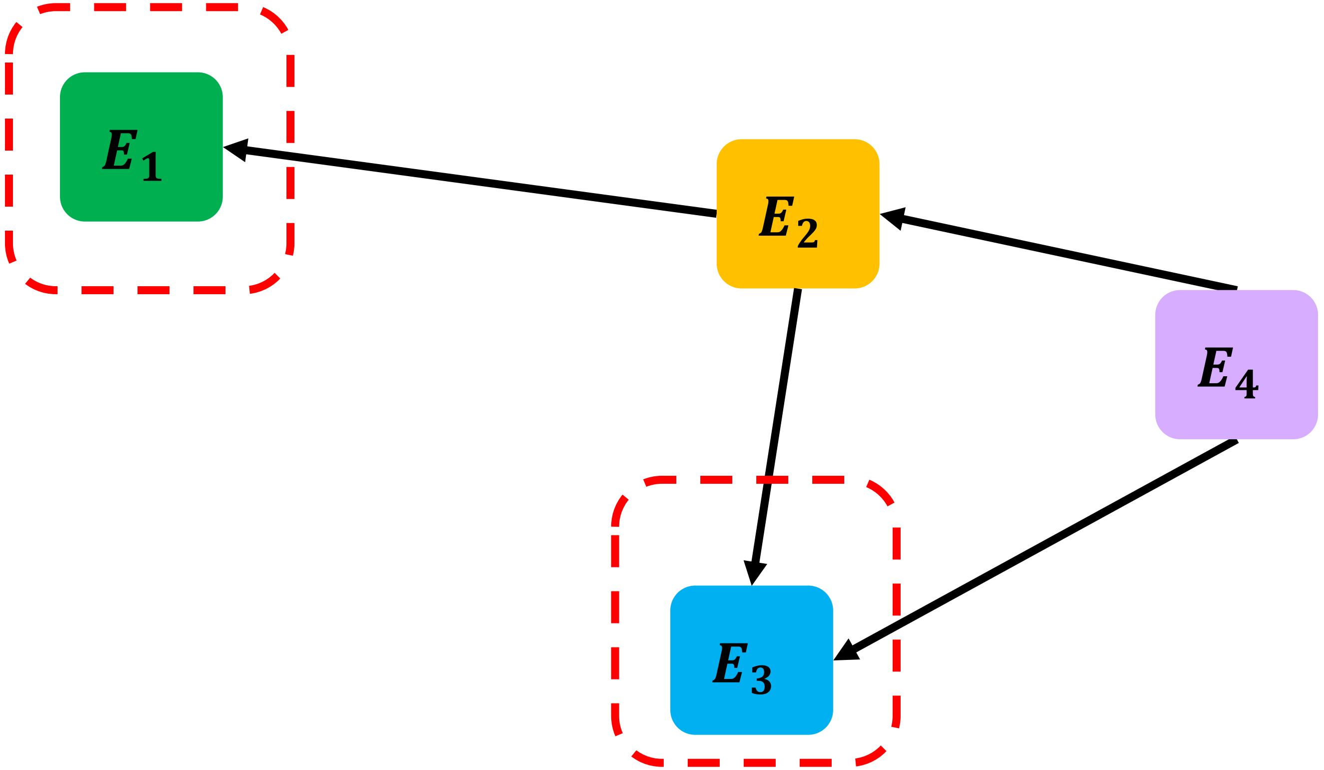
最大性：加入新顶点，不保证相互可达



E_1中任意点可达u_1
u_1可达u_2
u_2可达E_2中任意点

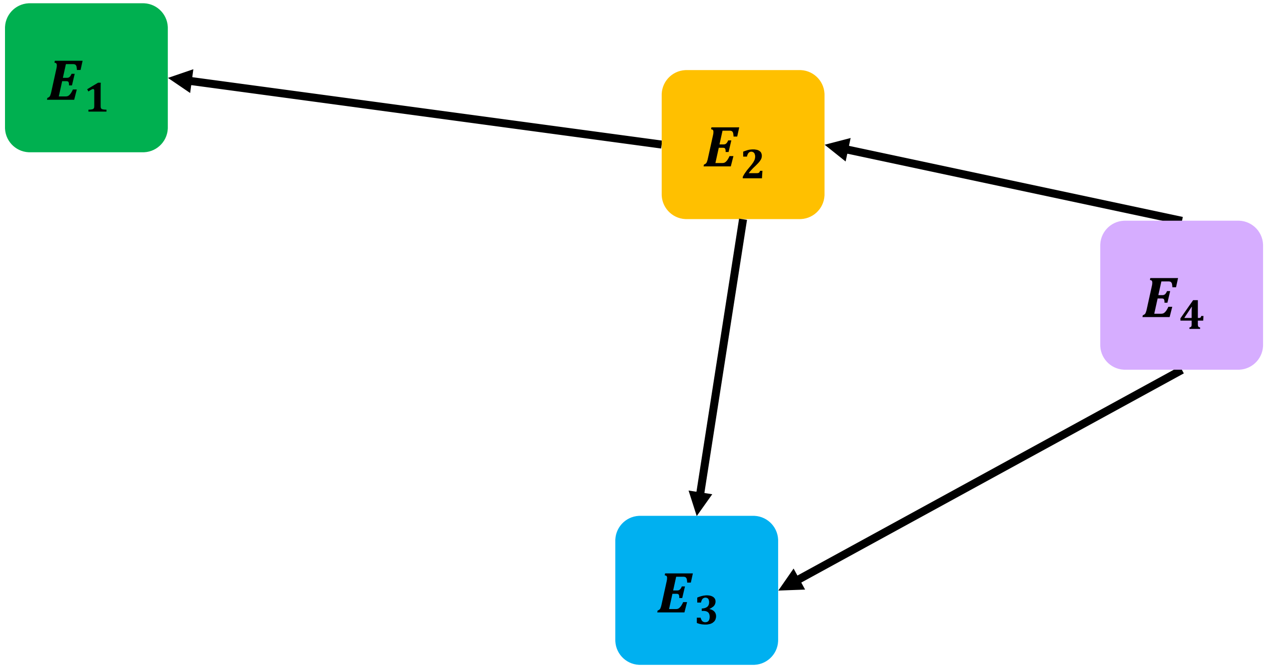
正确性证明

- 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



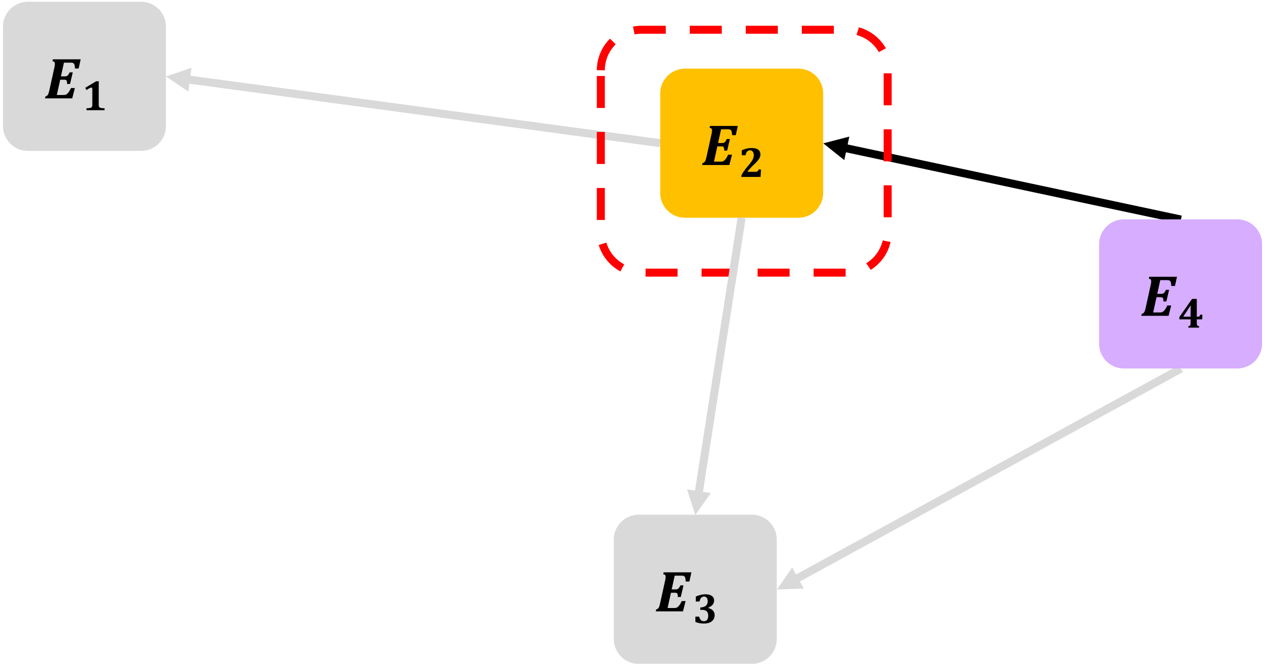
正确性证明

- 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}
  - 反证：若不存在，所有点均有出度，必存在环，矛盾



正确性证明

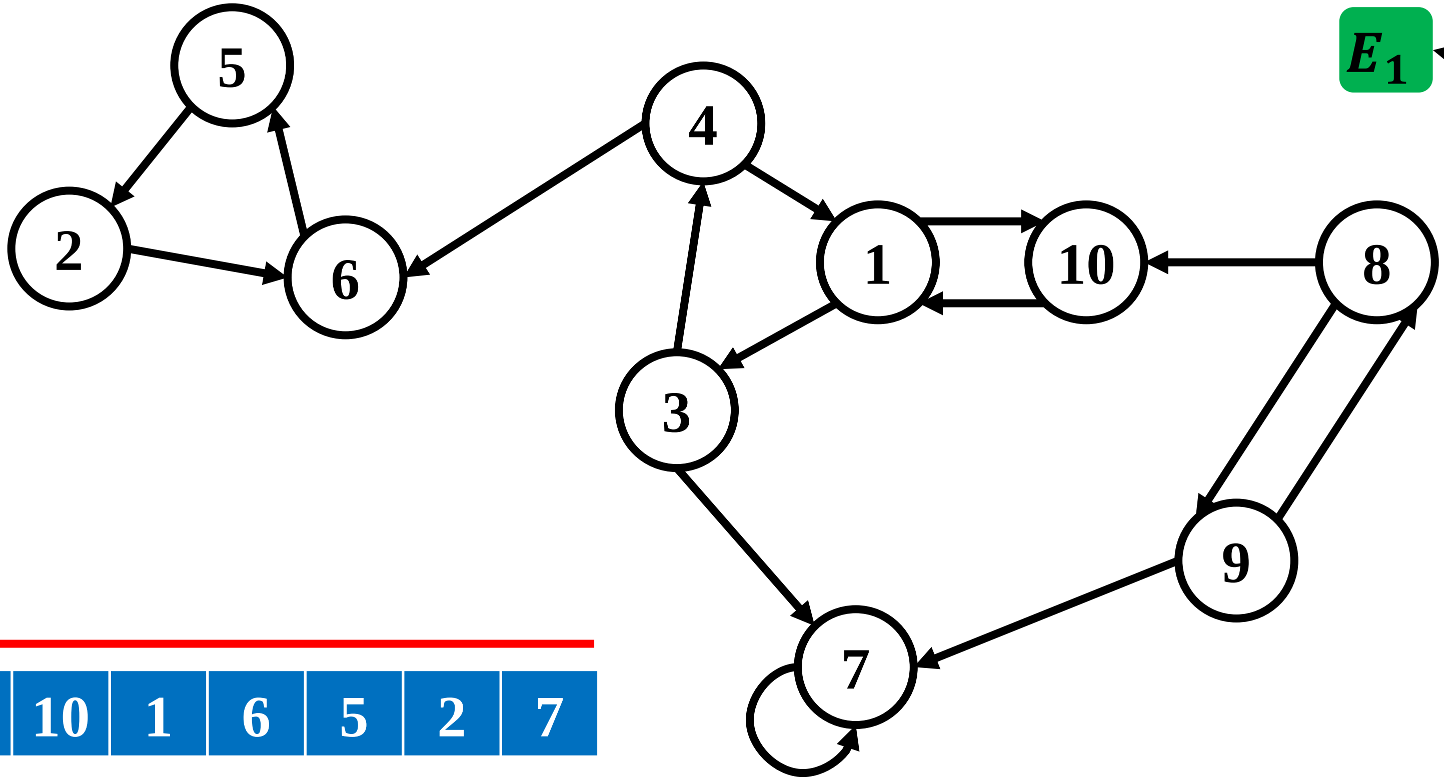
- 强连通分量图G^{SCC}：把强连通分量看作一个点，得到有向图
  - 性质：G^{SCC}一定是有向无环图
- SCC_{Sink}：G^{SCC}中出度为0的点
  - 性质1：G^{SCC}中存在至少一个SCC_{Sink}
  - 性质2：删除SCC_{Sink}，会产生新的SCC_{Sink}
  - 反证：若不存在，所有点均有出度，必存在环；而无环图子图必无环，矛盾

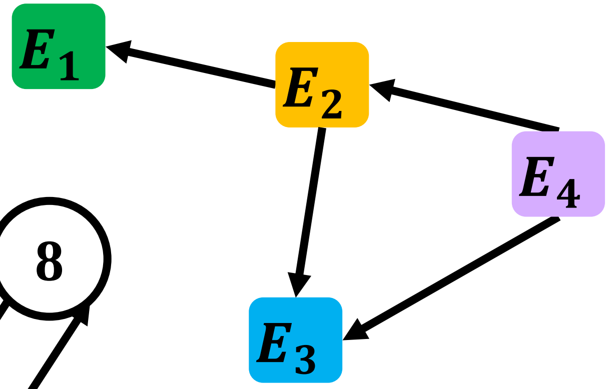


正确性证明

- 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



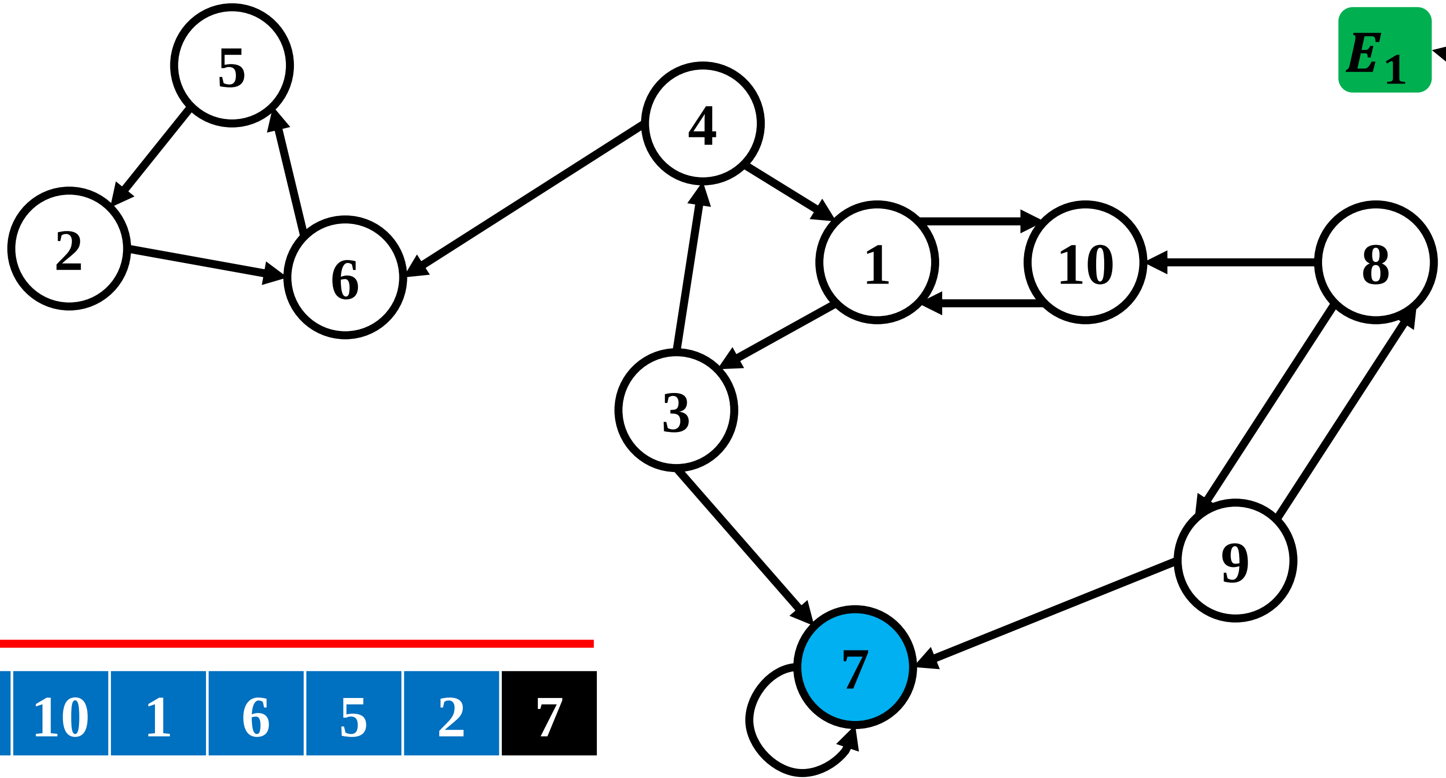


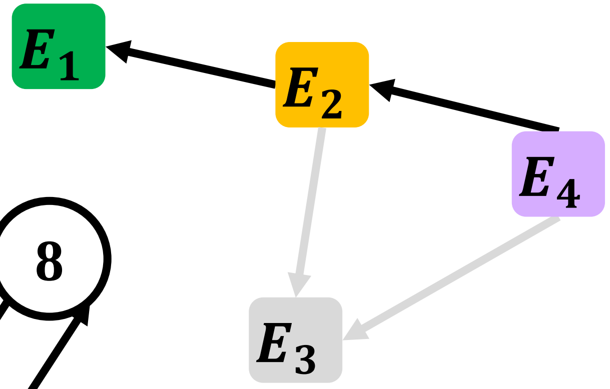
L	3	4	9	8	10	1	6	5	2	7

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



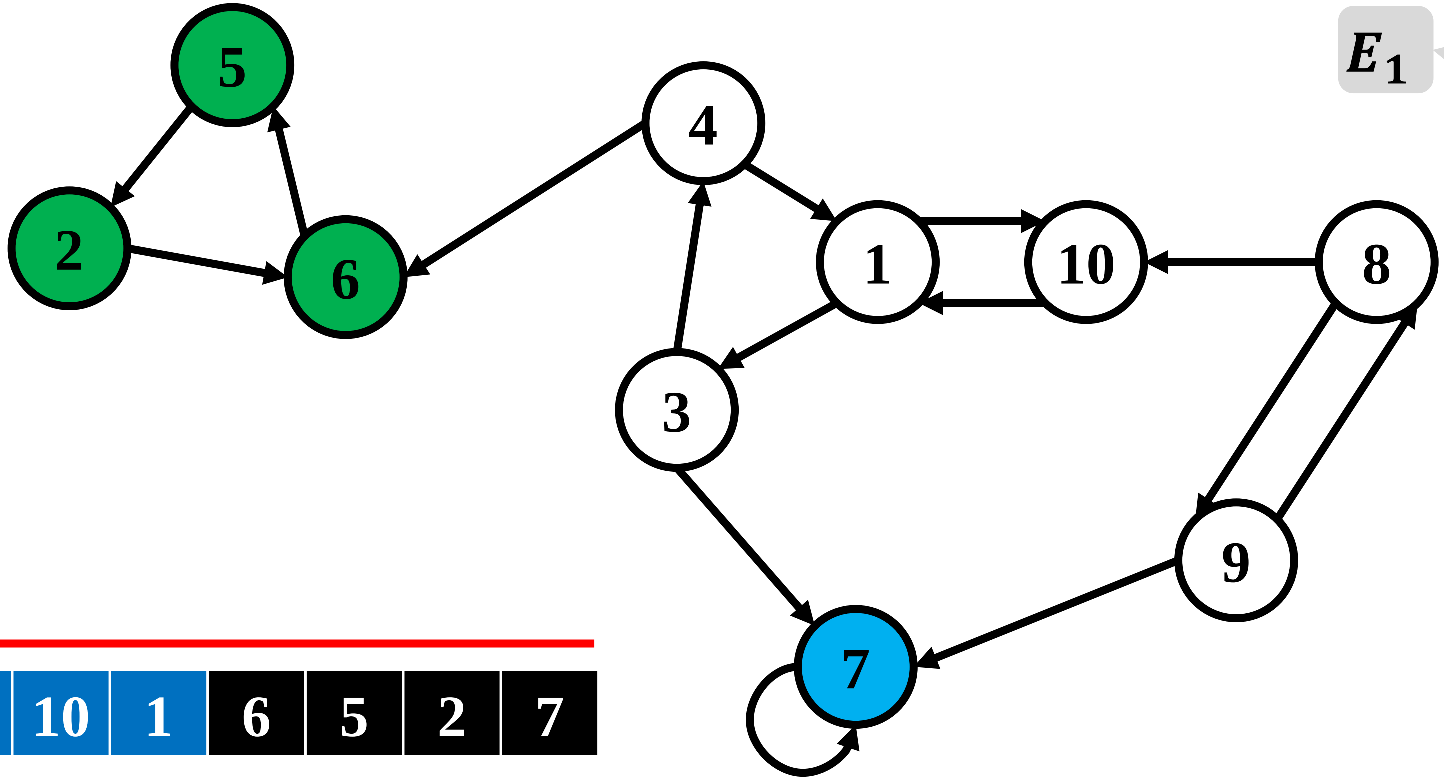


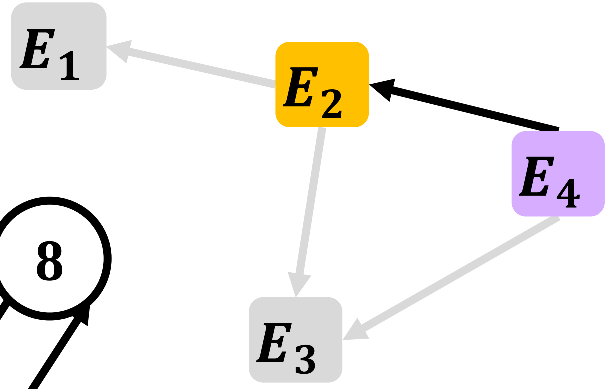
L	3	4	9	8	10	1	6	5	2	7

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



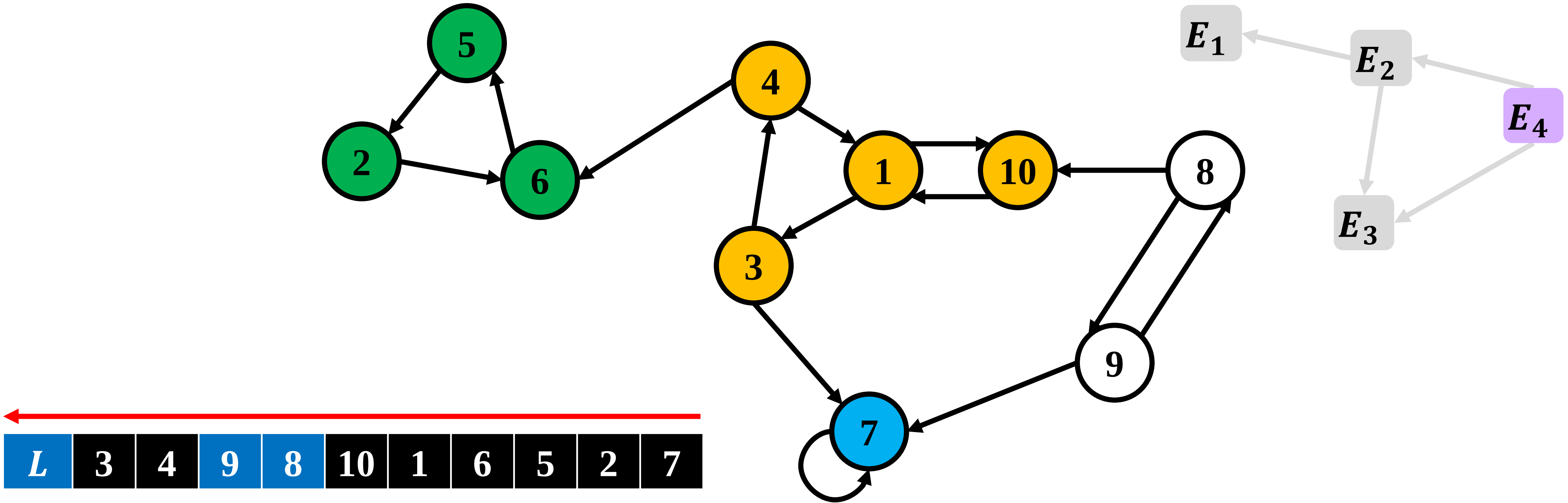


L	3	4	9	8	10	1	6	5	2	7

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}

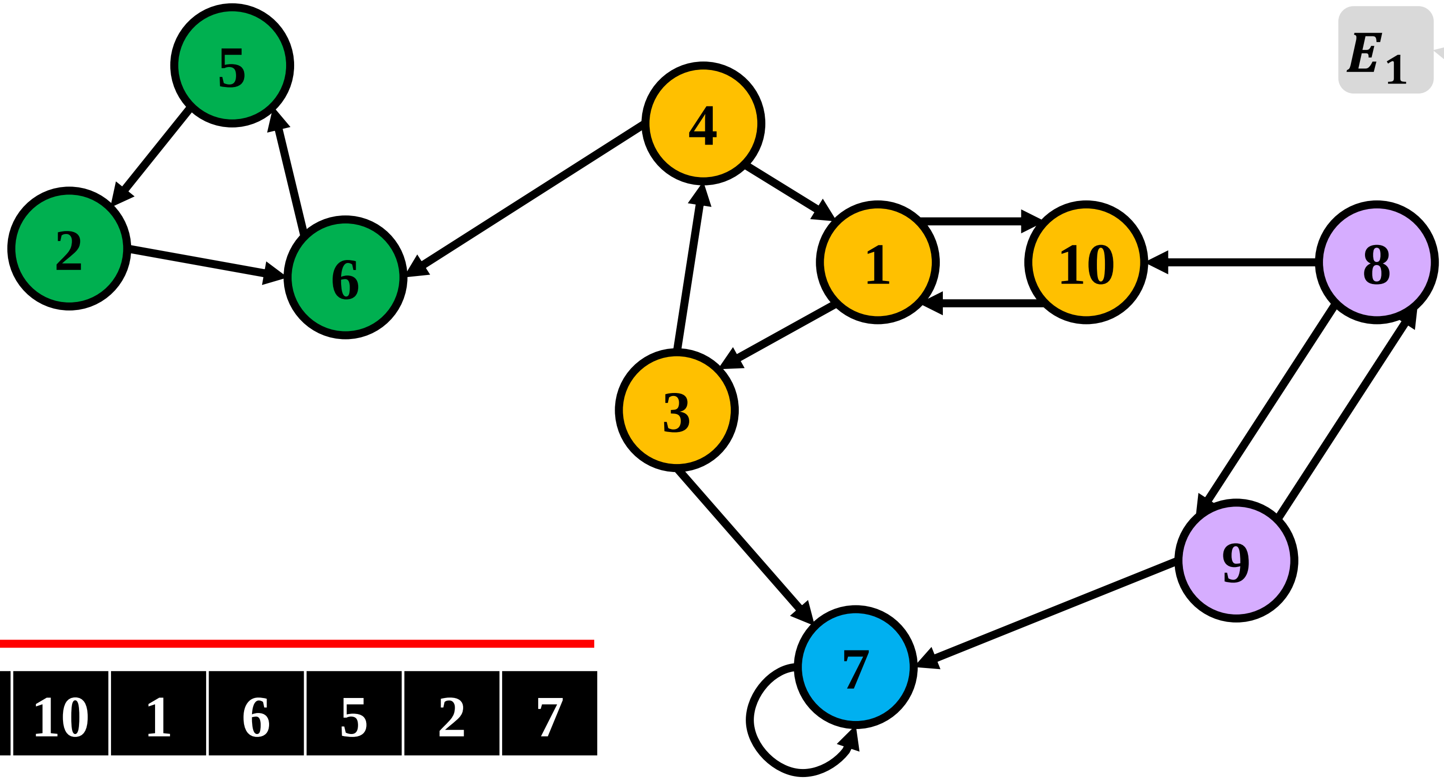
结合性质观察
算法第2次DFS

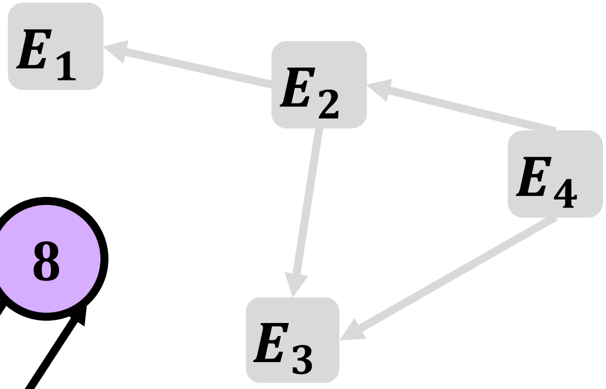


正确性证明

- 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



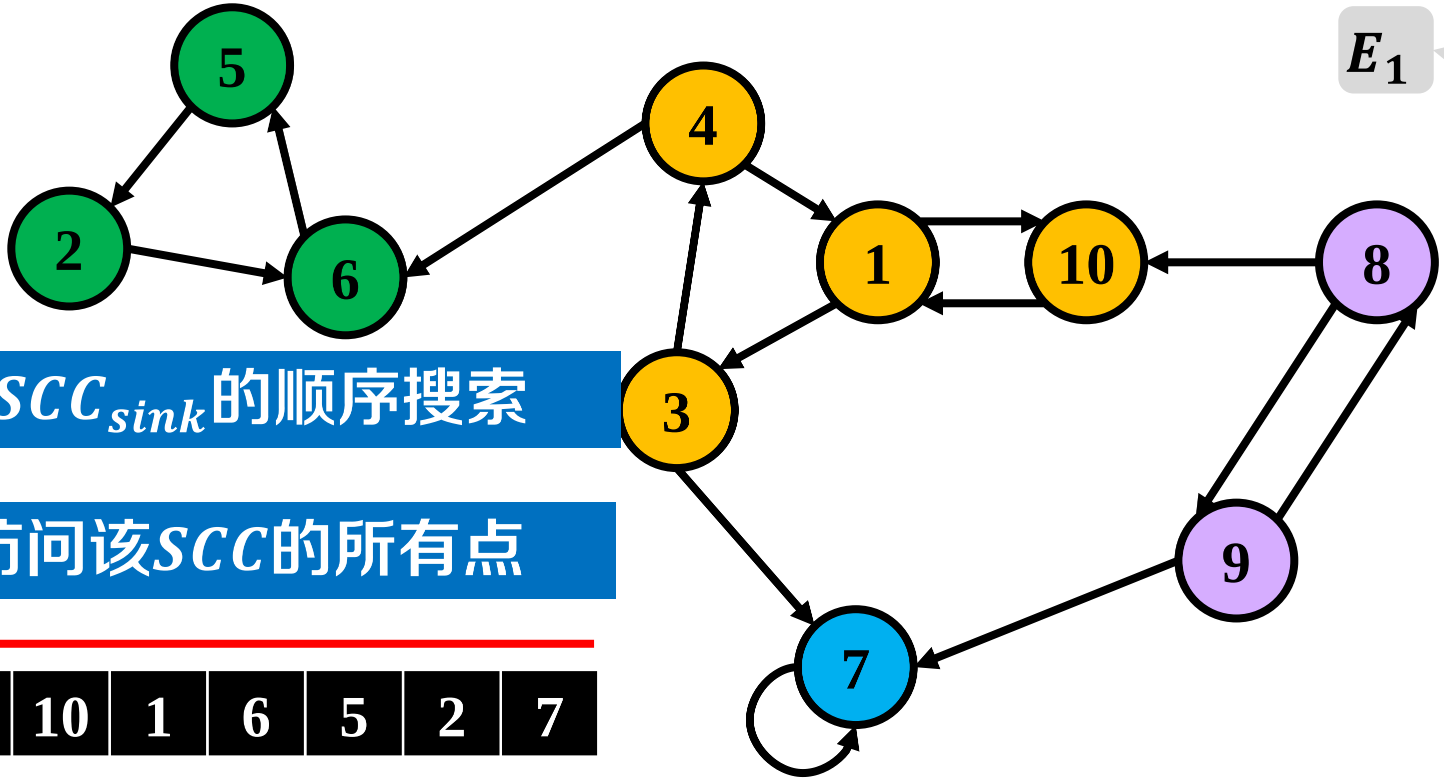


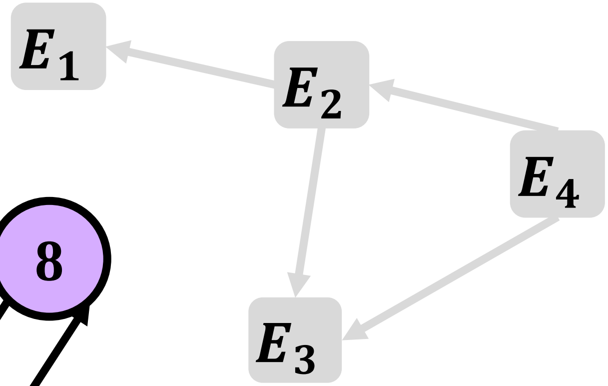
L	3	4	9	8	10	1	6	5	2	7

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





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

L	3	4	9	8	10	1	6	5	2	7

正确性证明

通过按L逆序执行DFS实现

？

第2次DFS按照SCC_{sink}的顺序搜索

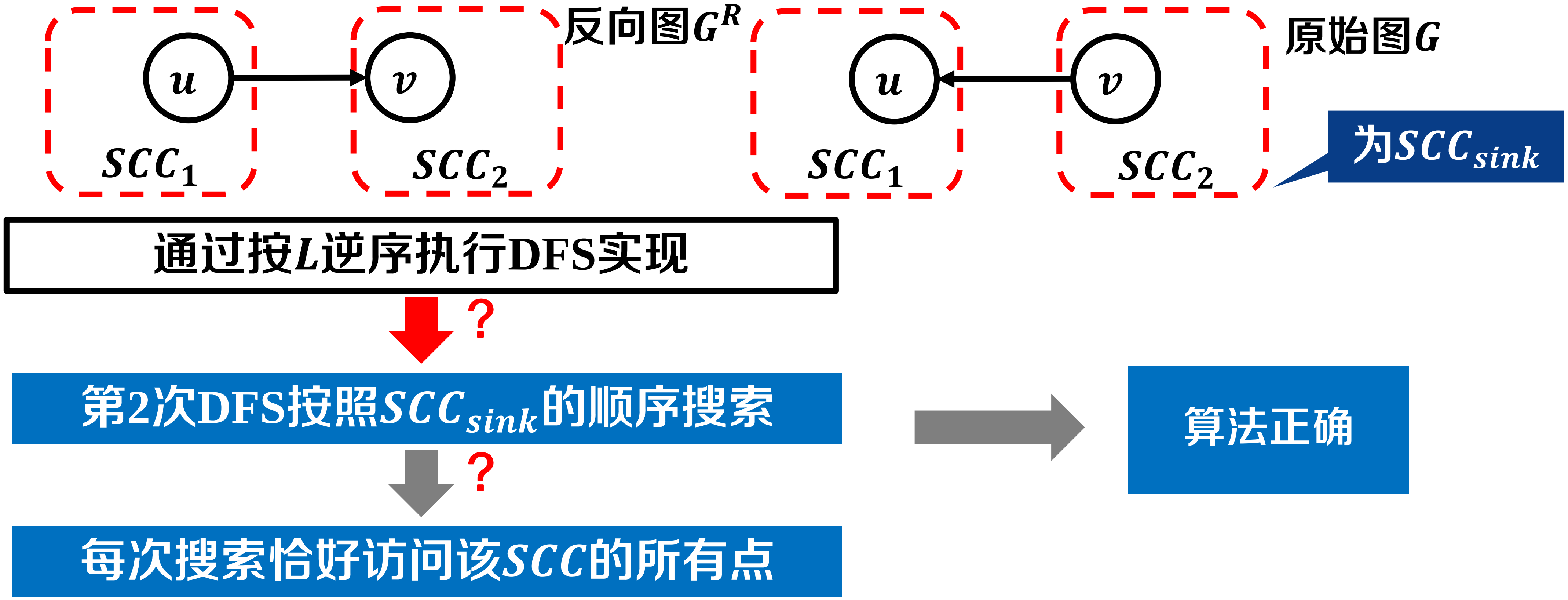
？

每次搜索恰好访问该SCC的所有点

算法正确

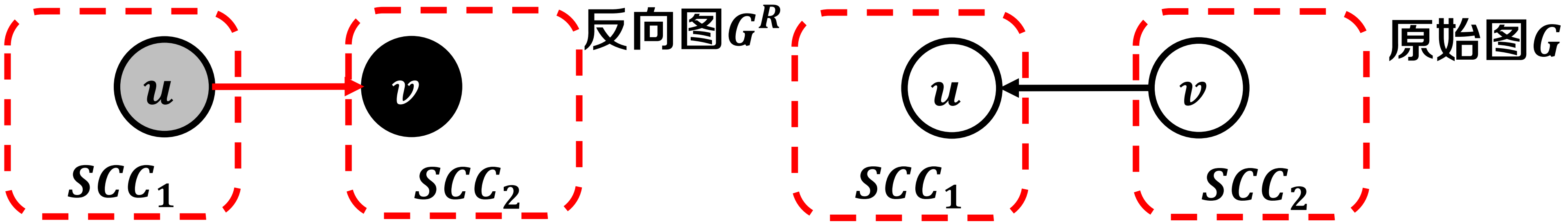
正确性证明

- 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$f(v) < f(u)$



为SCC_{sink}

通过按L逆序执行DFS实现

？

第2次DFS按照SCC_{sink}的顺序搜索

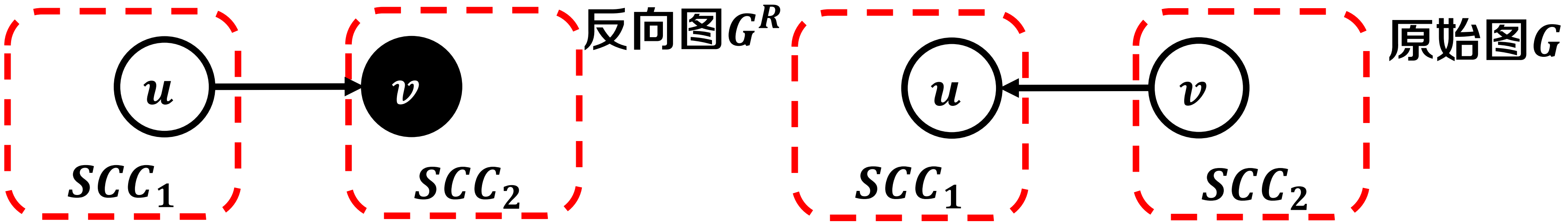
？

每次搜索恰好访问该SCC的所有点

算法正确

正确性证明

- 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$f(v)<f(u)$



为SCC_{sink}

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

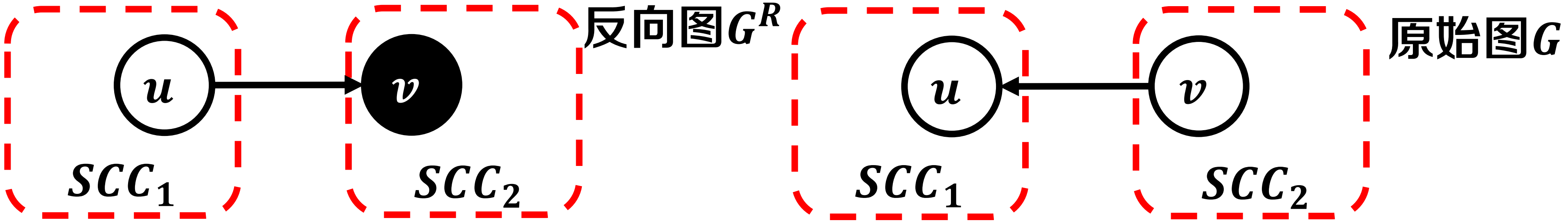
正确性证明

- 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	…	v	…	u	…

L：G^R上DFS完成时刻顺序



为SCC_{sink}

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

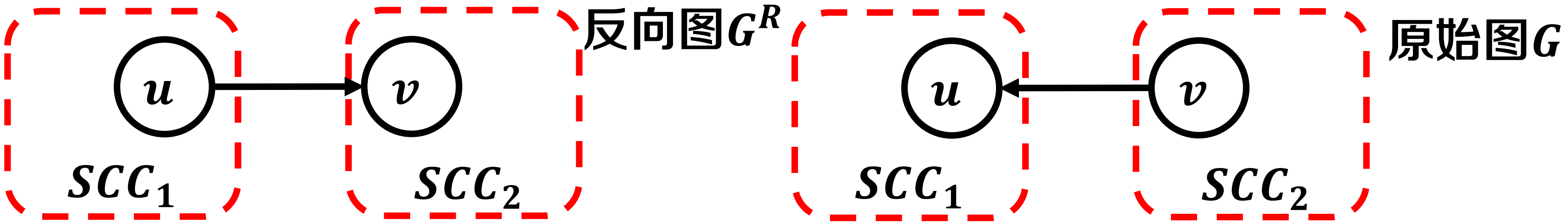
正确性证明

- 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	…	v	…	u	…

L：G^R上DFS完成时刻顺序



为SCC_{sink}

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

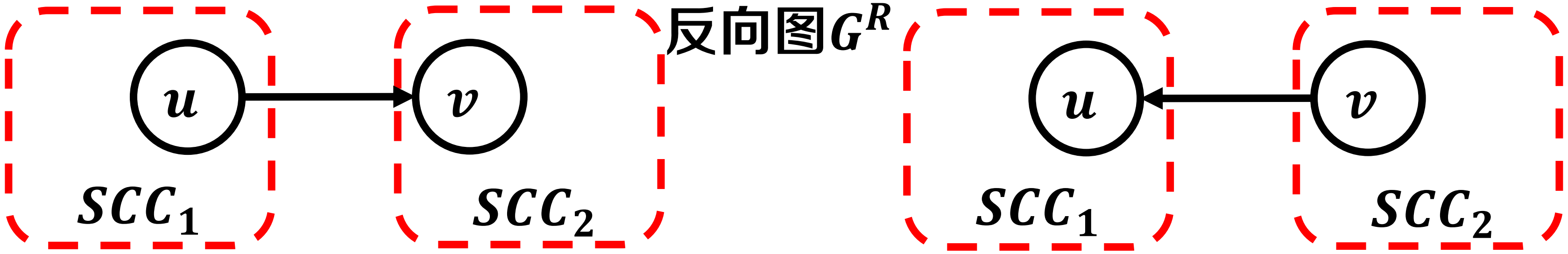
- 给定反向图G^R，存在边(u,v)，$u \in SCC_1$，$v \in SCC_2$
  - 若先搜索u，会从u搜索v，则$f(v) < f(u)$
  - 若先搜索v，v搜索完成才开始搜索u，则$f(v) < f(u)$
  - 所以按L的逆序，总是先搜索u，符合SCC_{sink}的顺序

f越来越小

L	…	v	…	u	…

L：G^R上DFS完成时刻顺序

反向图G^R



原始图G

为SCC_{sink}

通过按L逆序执行DFS实现

第2次DFS按照SCC_{sink}的顺序搜索

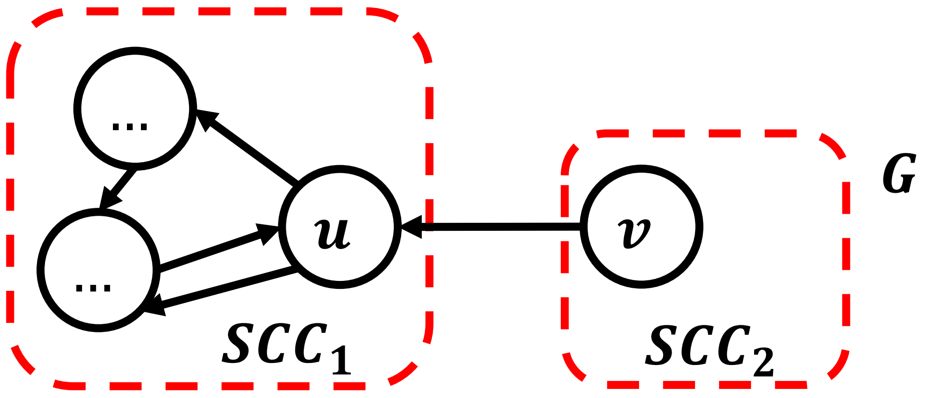
？

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 → DFS可以访问到该SCC所有点
- SCC_{sink}出度为0 → DFS不会访问该SCC以外的点



通过按L逆序执行DFS实现

↓ ✓

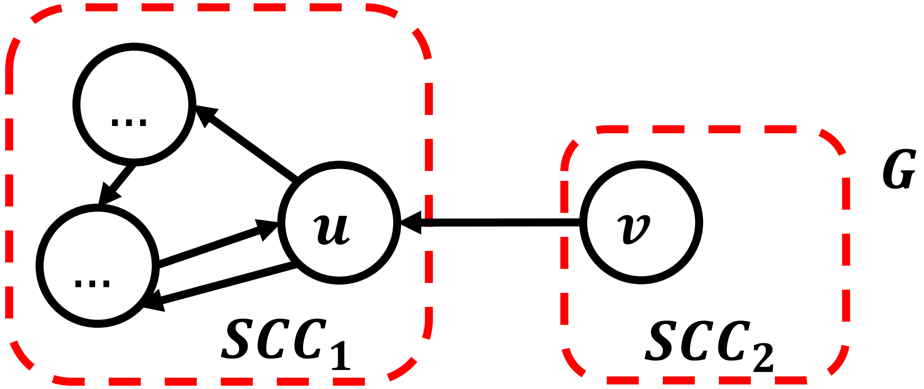
第2次DFS按照SCC_{sink}的顺序搜索 → 算法正确

↓ ?

每次搜索恰好访问该SCC的所有点

正确性证明

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



通过按L逆序执行DFS实现

↓ ✓

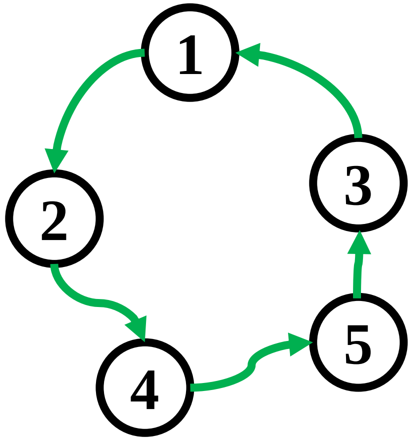
第2次DFS按照SCC_{sink}的顺序搜索 → 算法正确

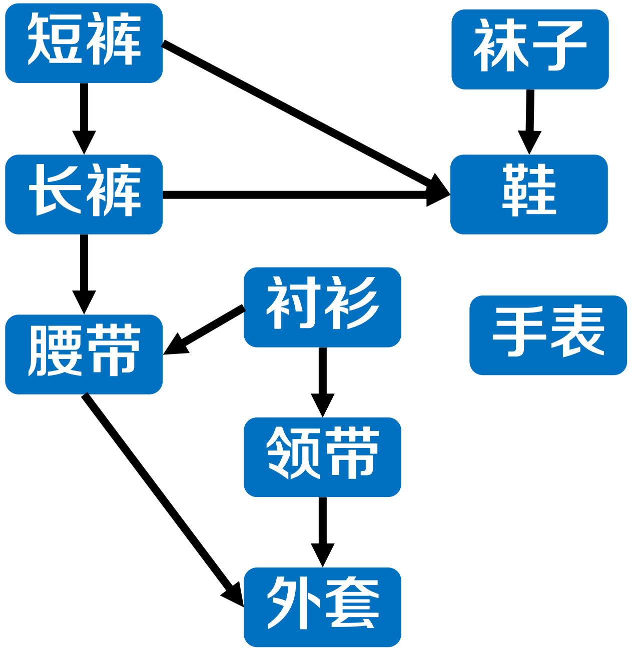
↓ ✓

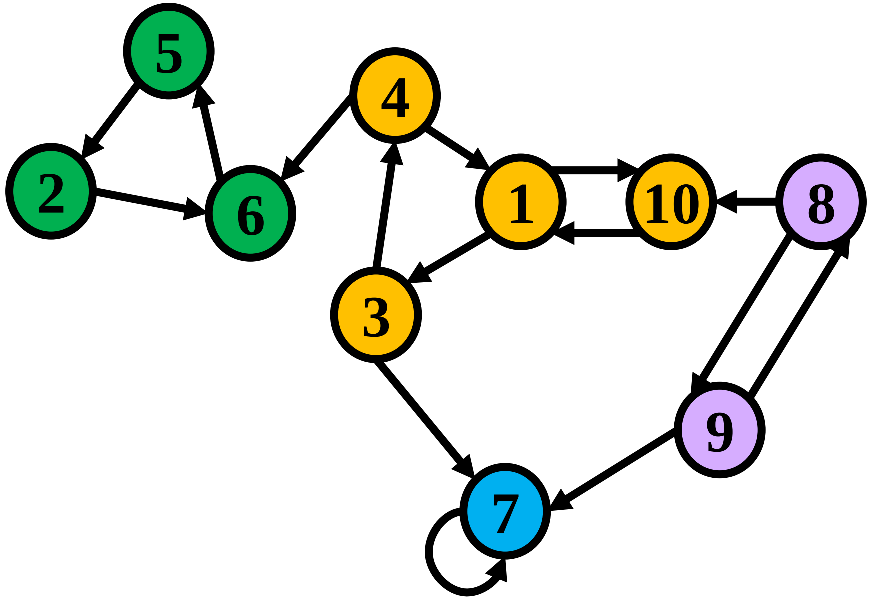
每次搜索恰好访问该SCC的所有点

小结

- 深度优先搜索的应用







环的存在性判定

拓扑排序

强连通分量

利用深度优先搜索边的性质

利用深度优先搜索点的性质（括号化定理）

谢谢

BDA

Beihang University