强连通分量

吉林大学计算机学院
谷方明
fmgu2002@sina.com

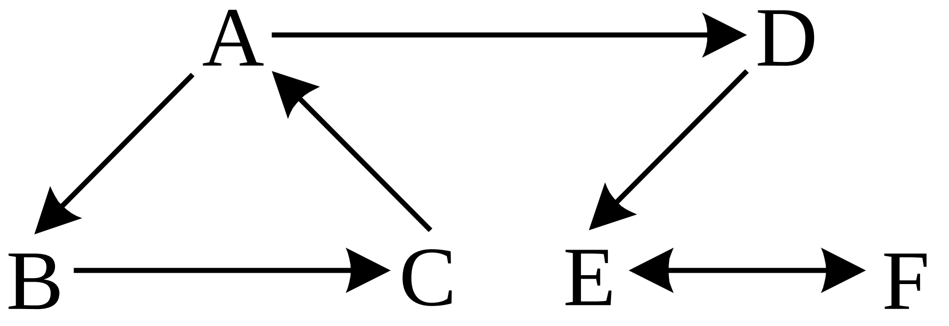
学习目标

- 理解求强连通分量的All_Componet算法
- 掌握求强连通分量的求Tarjan算法

定义回顾

- 有向图中如果两个点能互相可及（或连通）则称这两个点强连通。
- 有向图G中任意两点互相可及（或连通），则称G是强连通图。
- 有向图G的极大强连通子图，称为强连通分量(SCC,Strongly Connected Componenet)。

例



- 三个强连通分量: **{A,B,C},{D},{E,F}**

教材上的方法

- 利用Warshall算法求图G的可及矩阵；
- 根据强连通分量的定义判断两个顶点是否属于同一连通分量；

- 容器：记录当前SCC中的点
  - ✓ 栈、队列、线性表均可

- visited[ ]：visited[i]表示结点i是否已处理过

算法All_Componet(E.)

/*输入图的边集E，输出图中所有的强连通分量*/

All_Componet1[初始化]

FOR k = 1 TO n DO visited[k] ← 0.

Warshall(E.WSM). //计算图的可及矩阵

t ← 0. //记录连通分量的个数

CREATESTACK(scc).

```
All_Componet2 [计算图的全部连通分量]
 FOR v=1 TO n DO(
  IF (visited[v]=0)  THEN (  /*处理新的连通分量*/
   t ←t+1. visited[v]  ← 1.
   FOR i=1 TO n DO(
     IF( i≠v AND WSM[v][i]=1 AND WSM[i][v]=1) THEN(
        visited [i] ← 1.
        scc.push(i).))
   PRINT (“第”  t “个连通分量：”).
   WHILE(!scc.empty()) DO ( PRINT(scc.pop()).
  ) )  ▌
```

- 算法All_Componet的时间复杂度$O(n^3)$，空间复杂度$O(n^2)$

强连通分量的实用算法

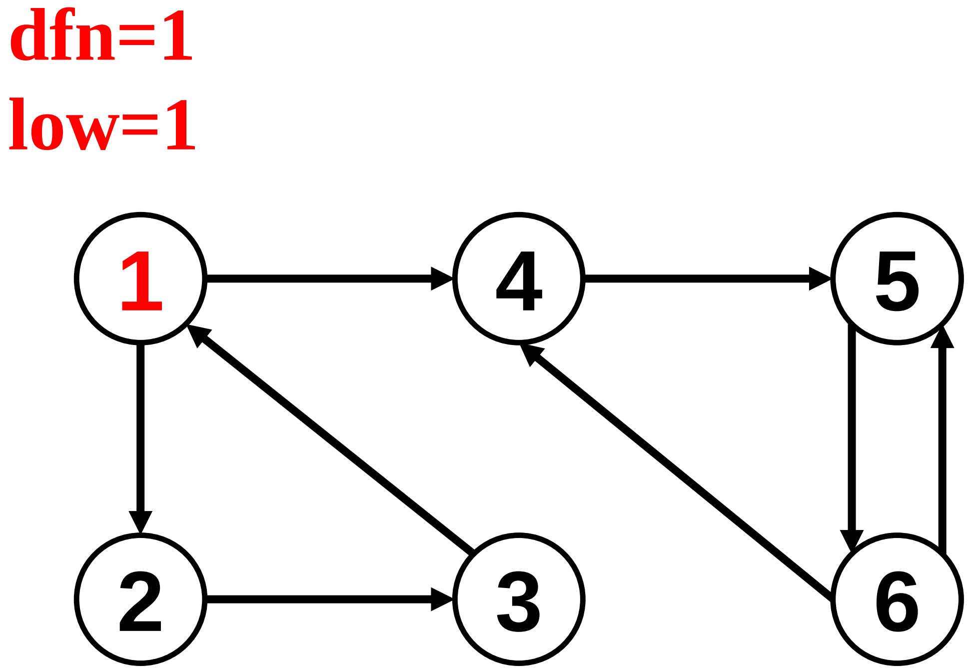
- **Kosaraju-Sharir算法**
  - ✓ 两次DFS，一次原图，一次逆图
  - ✓ 参考算法导论/网上教程，直观，但实现略复杂
- **Tarjan算法**
  - ✓ 利用DFS的dfn和搜索树的根编号low
- **Gabow算法**
  - ✓ Tarjan算法的提升版；利用DFS 和 两个栈；

- **三个算法都是基于DFS的，都是O(n+e)的。**

Tarjan算法思想

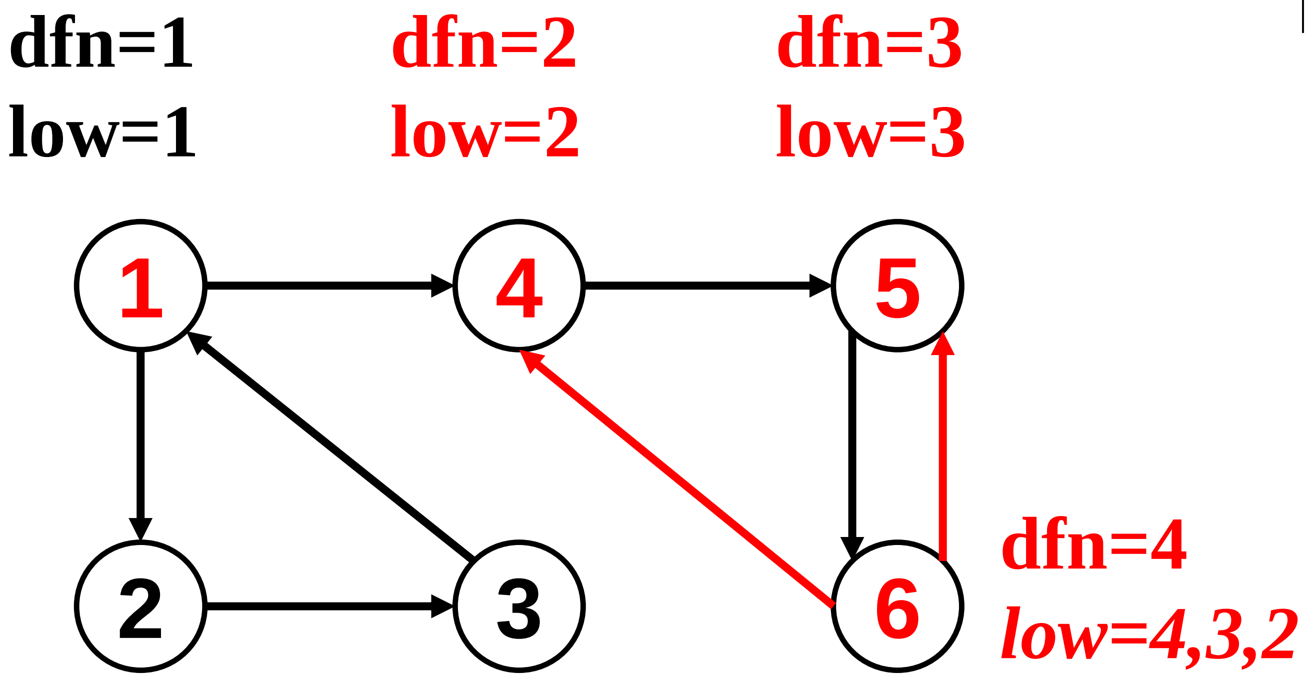
- SCC性质：存在一条**回路**能从初始点经过其中所有点又回到初始点。
- 处于同一个SCC中的结点必然构成DFS树的一棵子树。
- 要找SCC，找到它在DFS树上的根。
- **辅助结构**
  - ✓ dfn[u]： **dfs时达到顶点u的次序号（时间戳）**；
  - ✓ low[u]： **u所在DFS子树中次序号最小顶点的次序号**；
  - ✓ Stack：存储搜索路径上的点；

Tarjan算法示例



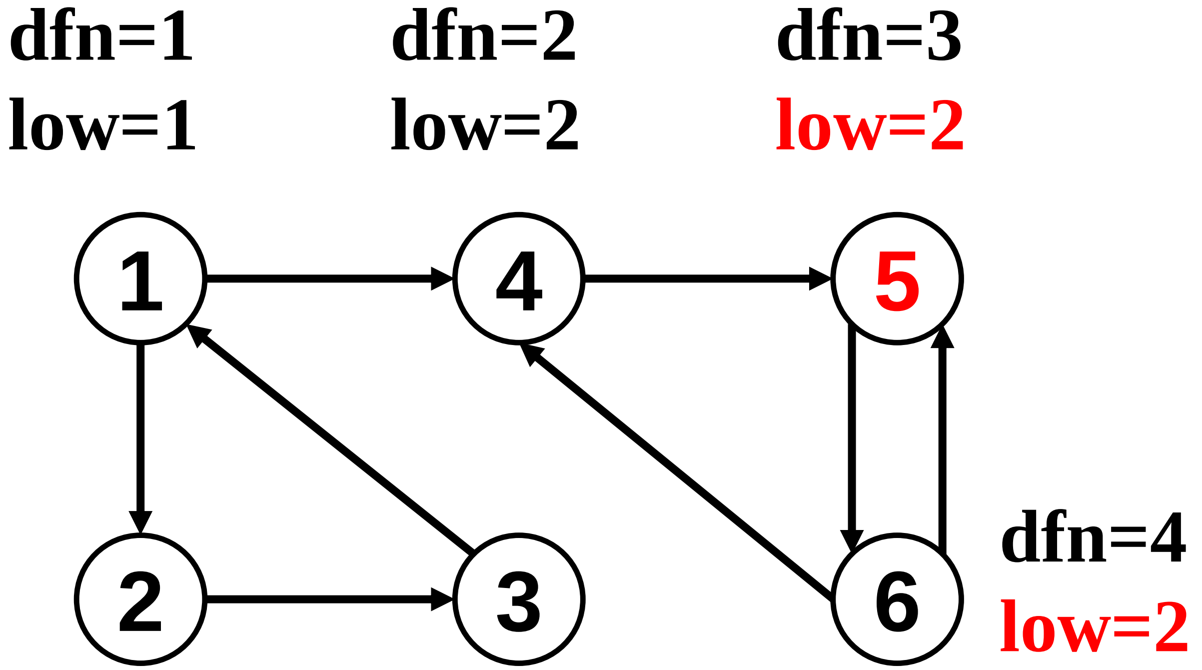
从结点1开始搜索，初始化dfn和low，将结点入栈

Stack：1



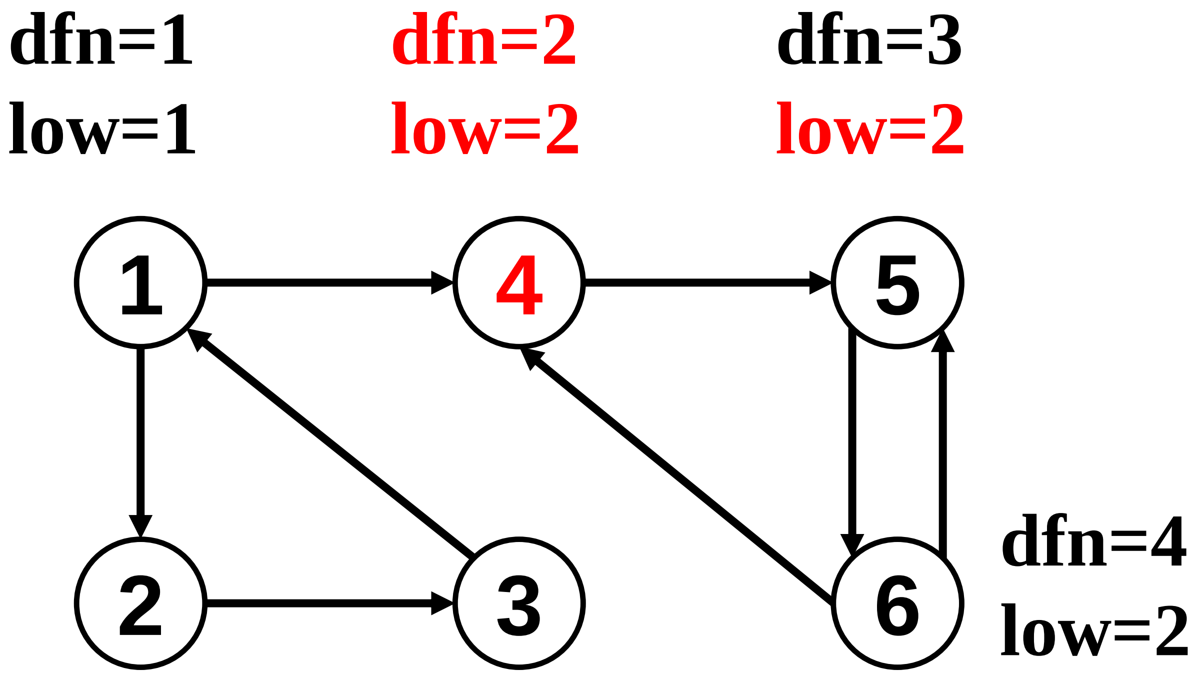
未遇到SCC之前，搜索正常进行，一直到6

Stack：1，4，5，6



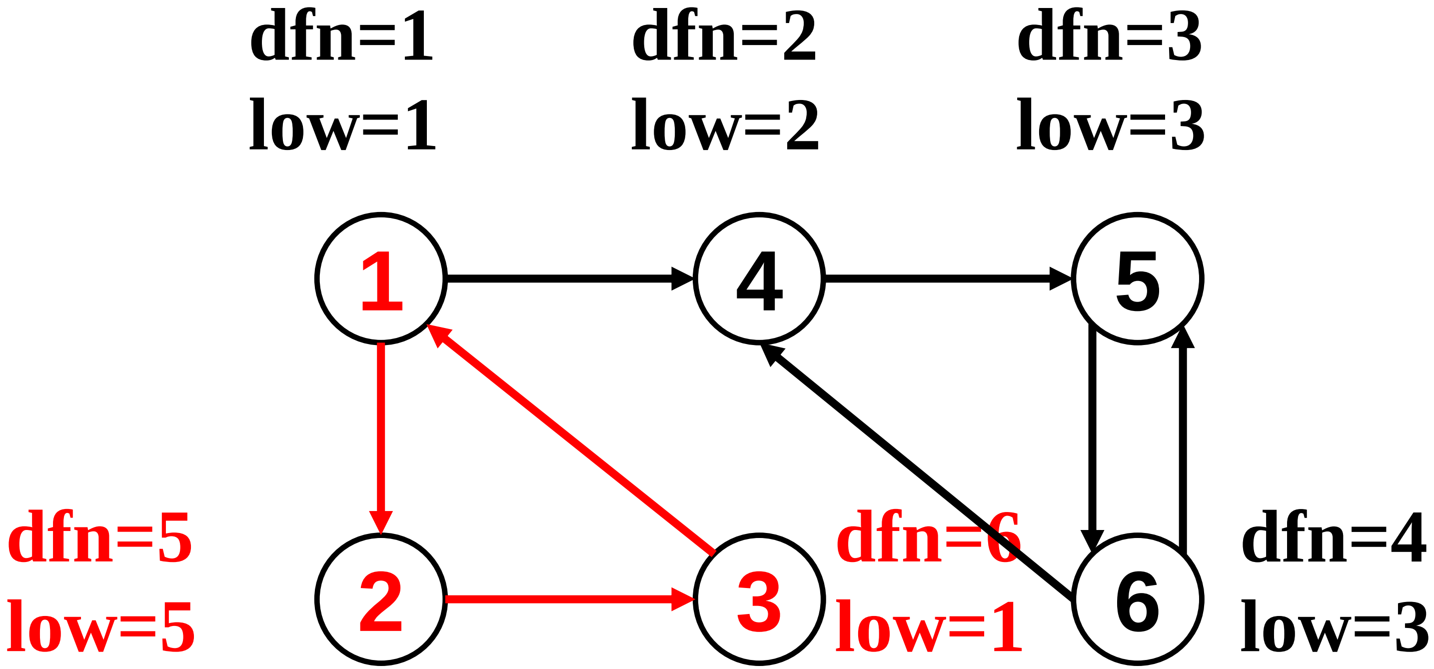
6不能继续搜索；回溯到5，更新low[5]=2

Stack：1，4，5，6



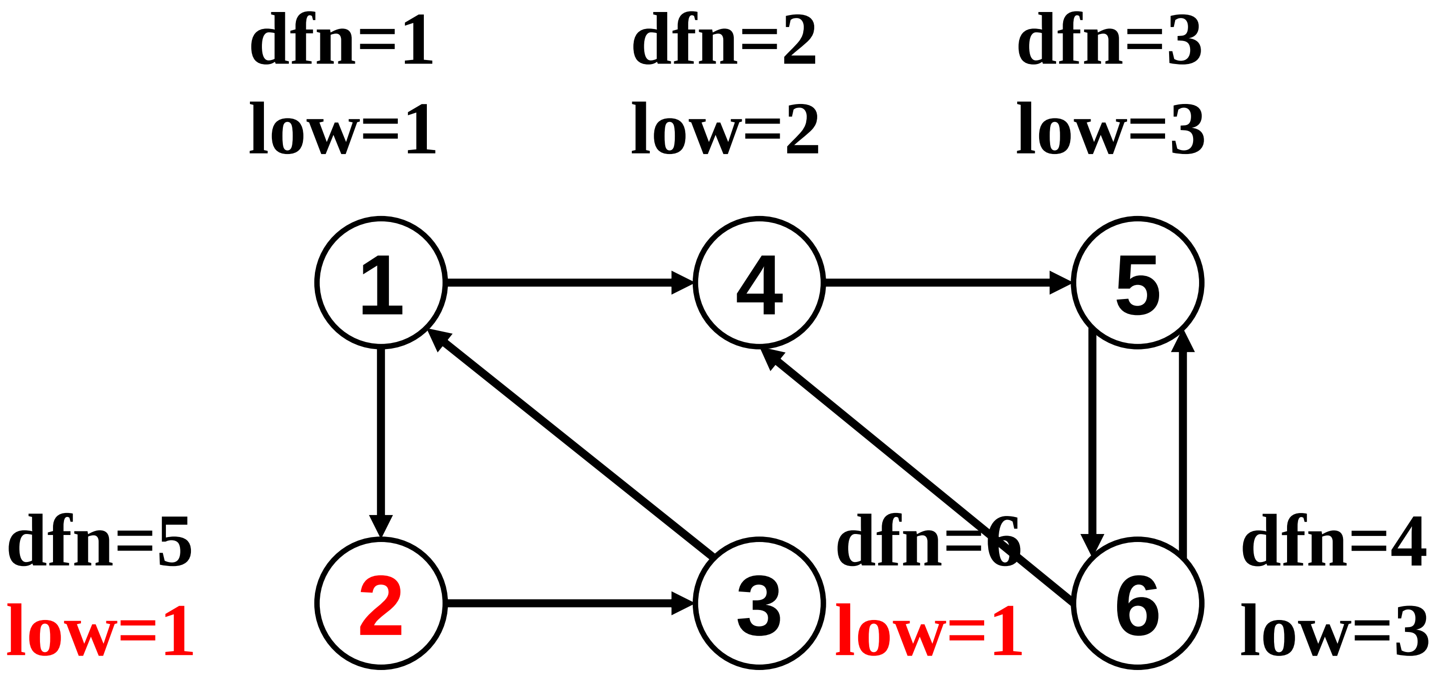
5不能继续搜索；回溯到4，low[4]不变。4不能继续搜索，dfn[4]=low[4]，找到一个SCC={4，5，6}

Stack：1



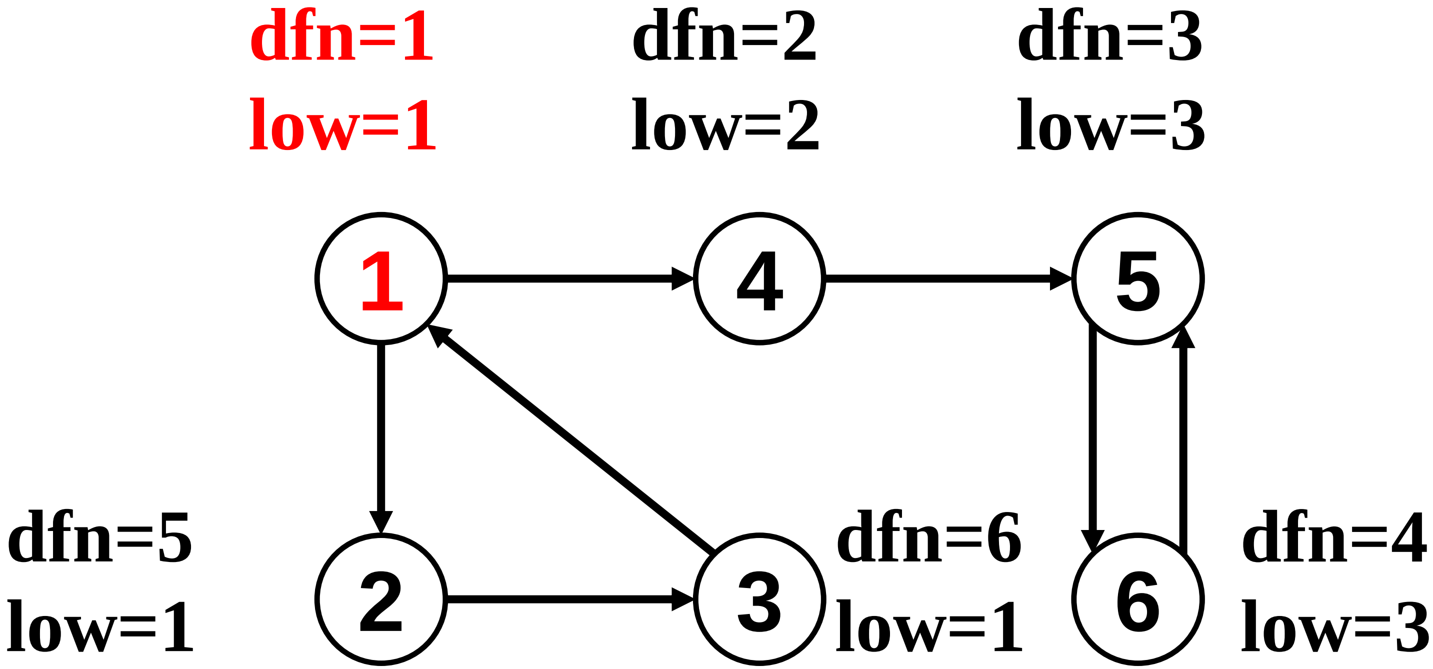
回溯到1，1可以继续搜索，一直到3

Stack：1，2，3



3不能继续搜索；回溯到2，更新low[2]=1

Stack：1，2，3



2不能继续搜索；回溯到1，low[4]不变。 1不能继续搜索，dfn[1]=low[1]，找到一个SCC={1，2，3}

Stack：

Tarjan算法伪代码

```
tarjan(u){
    DFN[u]=Low[u]=++Index    // 为u设定次序编号和Low初值
    Stack.push(u)            // 将u压入栈中
    for each (u, v) in E     // 枚举每一条边
        if (v is not visted) // 如果节点v未被访问过
            tarjan(v)        // 继续向下找
            Low[u] = min(Low[u], Low[v])
        else if (v in S)     // 如果节点v还在栈内
            Low[u] = min(Low[u], DFN[v])
    if (DFN[u] == Low[u])    // 如果节点u是强连通分量的根
        repeat
            v = S.pop  // 将v退栈，为该强连通分量中一个顶点
            print v
        until (u== v)
```

Tarjan算法分析

- **类似DFS，如果求全图的SCC，每次选择一个未被访问的顶点u，做Tarjan算法。**
  - ✓ DFN数组可作访问标志
  - ✓ Instack数组：顶点在栈中标志

- **Tarjan算法中，每个顶点都被访问了一次，且只进出了一次堆栈，每条边也只被访问了一次，所以该算法的时间复杂度为O(n+e)。**

Tarjan算法小结

- **Tarjan算法特点**
  - ✓ 只用对原图进行一次DFS，简洁。
  - ✓ 实测中，Tarjan算法运行效率比Kosaraju算法约高30%

- **Tarjan算法用途**
  - ✓ 缩环
  - ✓ 拓展：割点、桥、双连通分量