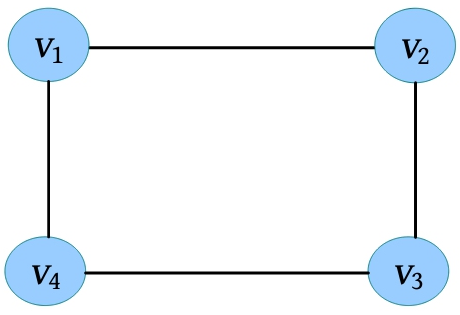
**图的连通性**

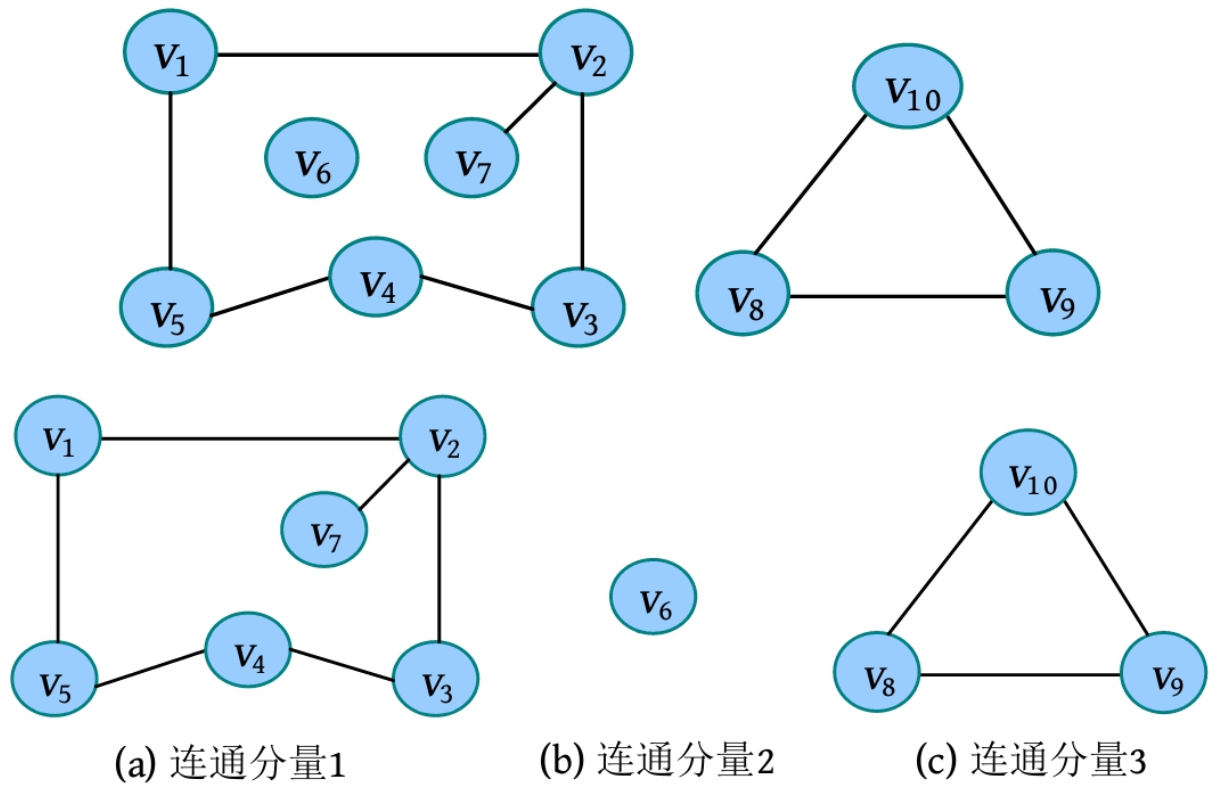
1. **无向图的连通分量**

在无向图中，如果从节点vi到节点vj有路径，则称节点vi和节点vj是连通的。如果图中任意两个节点都是连通的，则称图G为**连通图**。如下图所示就是一个连通图。



无向图G的极大连通子图被称为图G的连通分量。极大连通子图是图G连通子图，如果再向其中加入一个节点，则该子图不连通。连通图的连通分量就是它本身；非连通图则有两个以上的连通分量。

例如在下图中有3个连通分量。

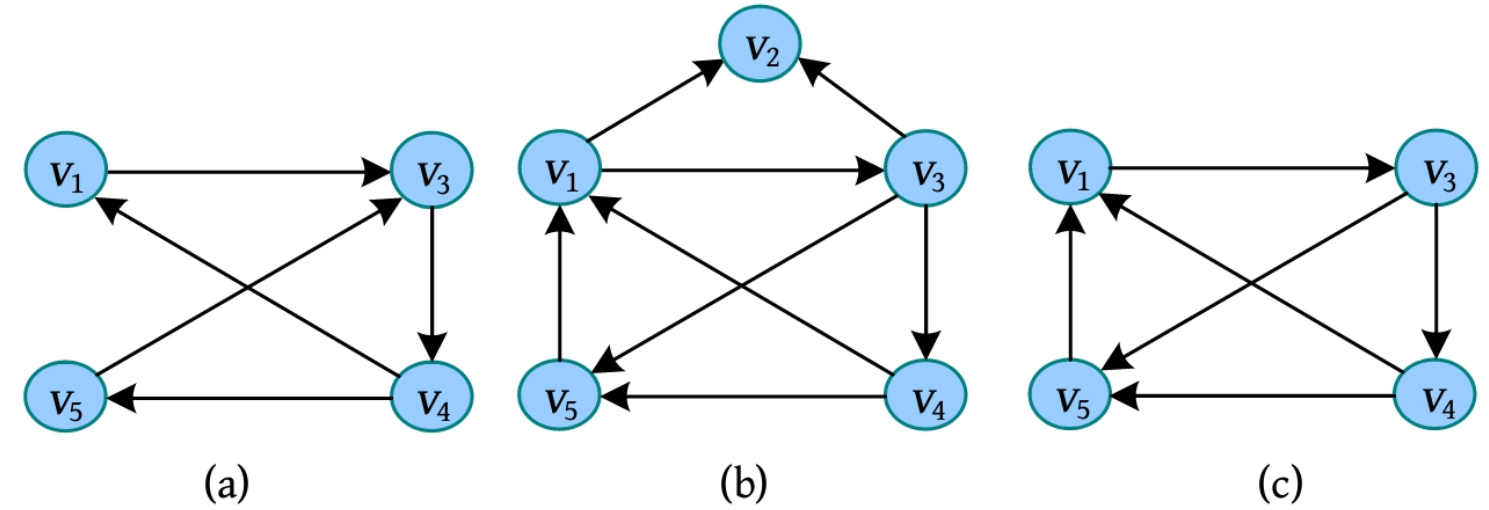


**2. 有向图的强连通分量**

在有向图中，如果图中的任意两个节点从vi到vj都有路径，且从vj到vi也有路径，则称图G为**强连通图**。

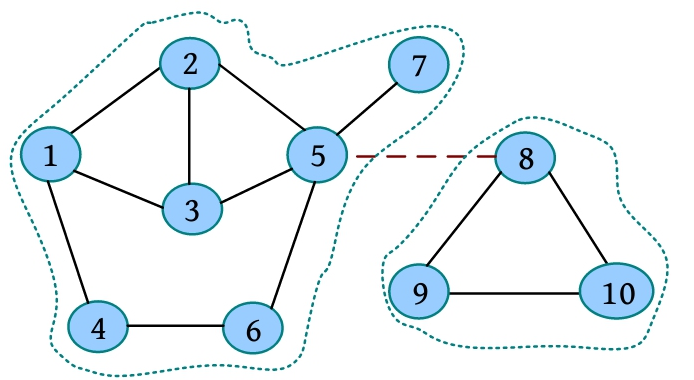
有向图G的极大强连通子图被称为图G的强连通分量。极大强连通子图是图G的强连通子图，如果再向其中加入一个节点，则该子图不再是强连通的。

例如在下图中，(a)是强连通图，(b)不是强连通图，(c)是(b)的强连通分量。



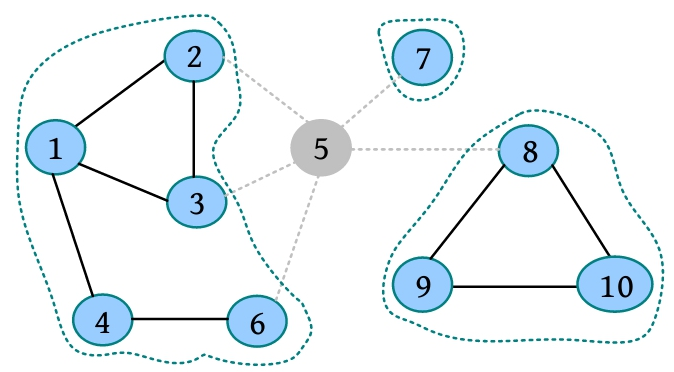
**3. 无向图的桥与割点**

在生活中，桥是连接河两岸的交通要道，桥断了，则河两岸不再连通。在图论中，桥有同样的含义，如下图所示，去掉边5-8后，图分裂成两个互不连通的子图，边5-8为图G的桥。同样，边5-7也为图G的桥。



如果在去掉无向连通图G中的一条边e后，图G分裂为两个不相连的子图，那么e为图G的**桥或割边**。

在日常网络中有很多路由器使网络连通，有的路由器坏掉也无伤大雅，网络仍然连通，但若非常关键节点的路由器坏了，则网络将不再连通。如下图所示，如果节点5的路由器坏了，图G将不再连通，会分裂成3个不相连的子图，则节点5为图G的割点。



如果在去掉无向连通图G中的一个点v及与v关联的所有边后，图G分裂为两个或两个以上不相连的子图，那么v为图G的**割点**。

**注意：删除边时，只把该边删除即可，不要删除与边关联的点；而删除点时，要删除该点及其关联的所有边。**

**割点与桥的关系：①有割点不一定有桥，有桥一定有割点**； ②**桥一定是割点依附的边**。

**4. 无向图的双连通分量**

如果在无向图中不存在桥，则称它为**边双连通图**。在边双连通图中，在任意两个点之间都存在**两条及以上路径**，且路径上的**边互不重复**。

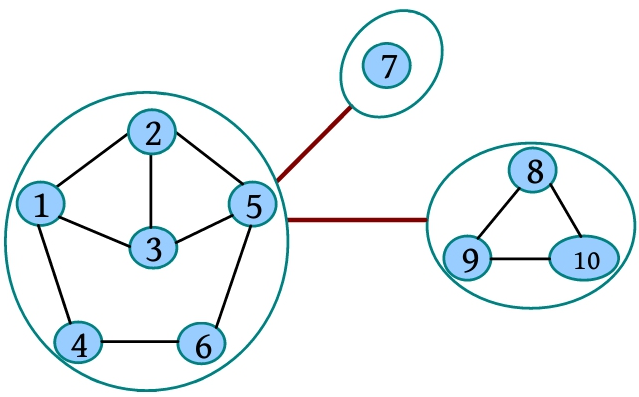
如果在无向图中不存在割点，则称它为**点双连通图**。在点双连通图中，如果**节点数大于2**，则在任意两个点间都存在**两条或以上路径**，且路径上的**点互不重复**。

无向图的极大边双连通子图被称为**边双连通分量，记为e-DCC**。无向图的极大点双连通子图被称为**点双连通分量，记为v-DCC**。二者被统称为双**连通分量DCC**。

**5. 双连通分量的缩点**

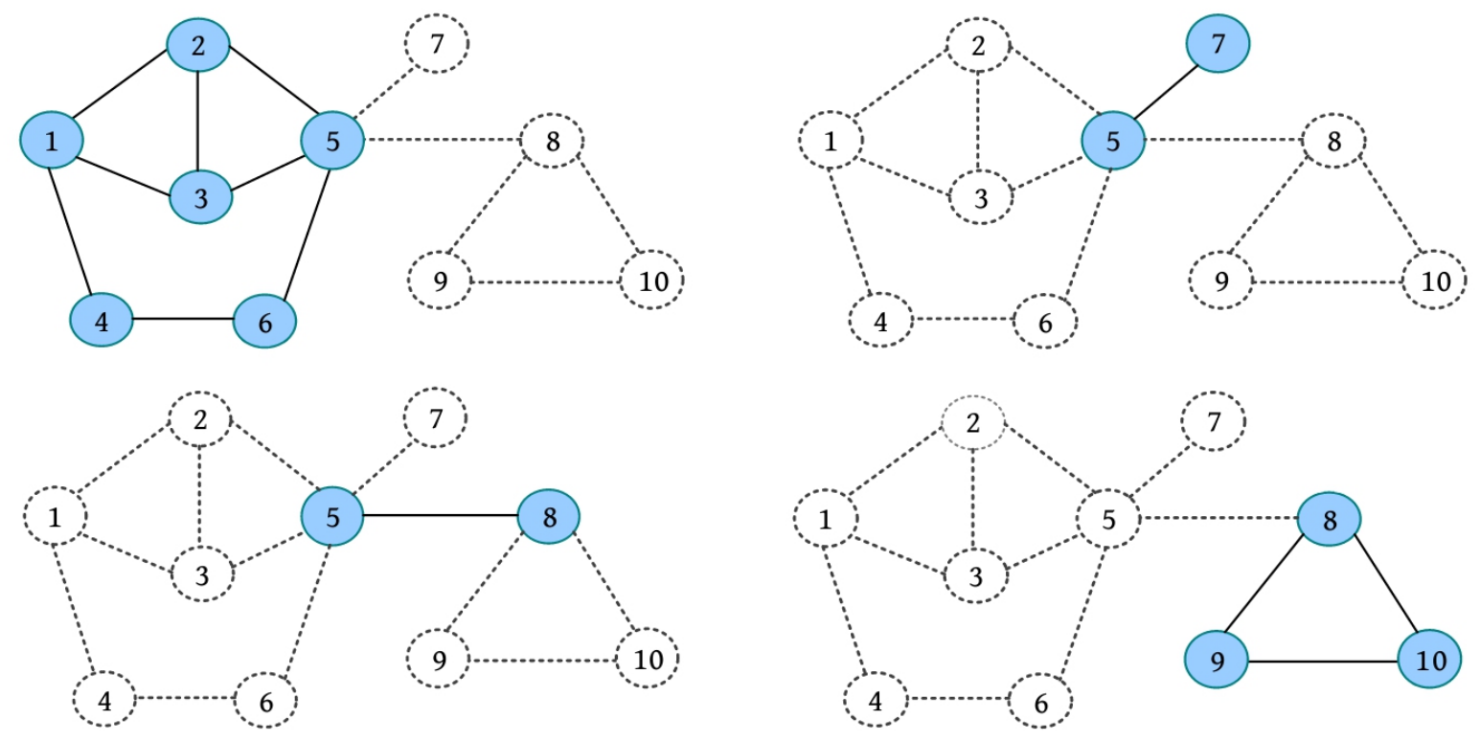
把每一个**边双连通分量e-DCC都看作一个点**，**把桥看作连接两个缩点的无向边**，可**得到一棵树**，这种方法被称为**e-DCC缩点**。

例如，在下图中有两个桥：5-7和5-8，将每个桥的边都保留，将桥两端的边双连通分量缩为一个点，生成一棵树。



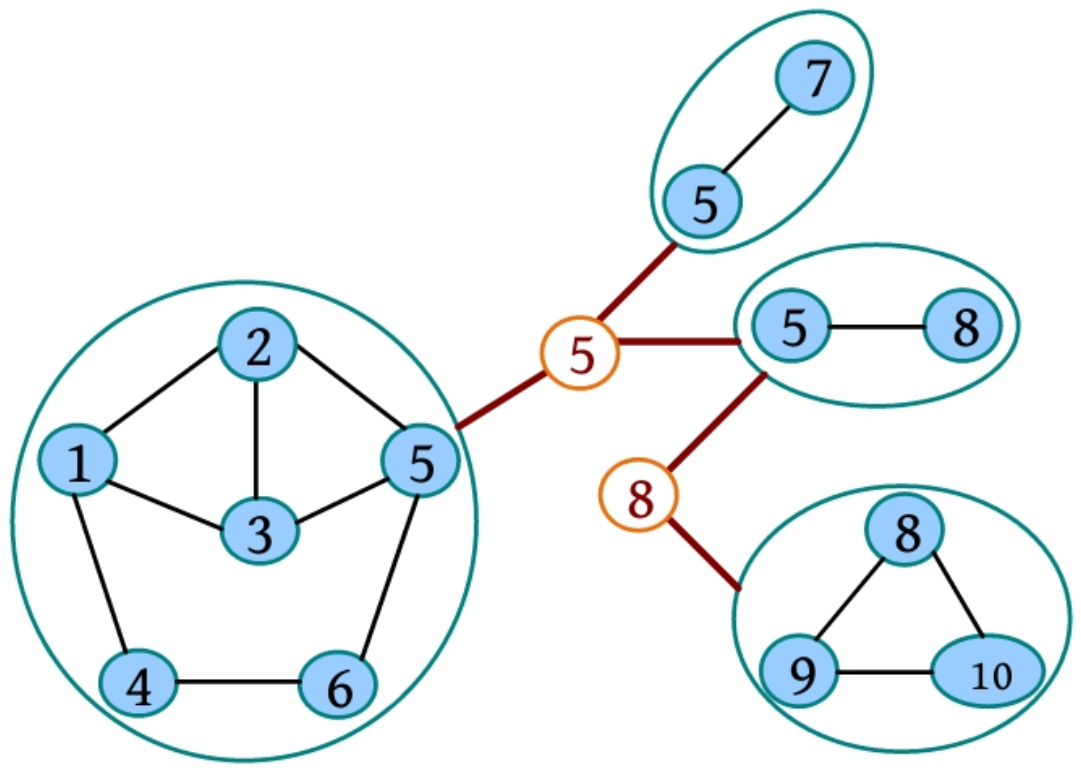
**注意：边双连通分量就是删除桥之后留下的连通块**，**但点双连通分量并不是删除割点后留下的连通块**。

在图G中有两个割点（5和8）及4个点双连通分量，如下图所示。



把**每一个点双连通分量v-DCC都看作一个点**，**把割点看作一个点**，**每个割点都向包含它的v-DCC连接一条边**，**得到一棵树**，这种方法被称为**v-DCC缩点**。

例如，在图G中有两个割点5、8，前3个点双连通分量都包含5，因此从5向它们引一条边，后两个点双连通分量都包含8，因此从8向它们引一条边，如下图所示。



**Tarjan强连通分量算法**



**Robert E.Tarjan （1986年图灵奖）罗伯特·陶尔扬**

Robert Tarjan以在数据结构和图论上的开创性工作而闻名，他的一些著名算法包括**Tarjan最近公共祖先离线算法**、**Tarjan强连通分量算法**及**Link-Cut-Trees算法**等。其中，Hopcroft-Tarjan平面嵌入算法是第1个线性时间平面算法。Robert Tarjan也开创了重要的数据结构，例如斐波纳契堆和**Splay树**，另一项重大贡献是分析了**并查集**。

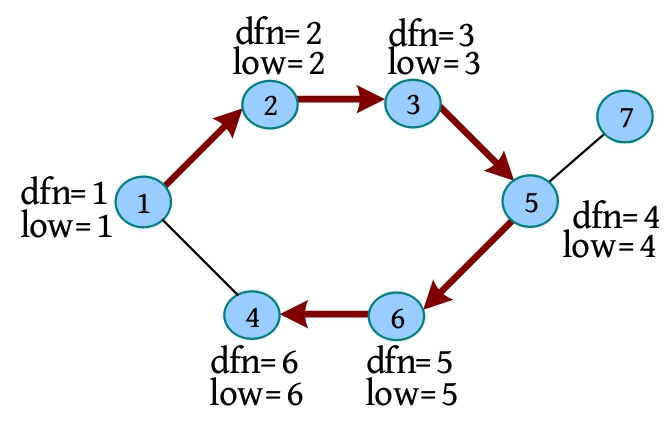
在介绍算法之前，首先引入时间戳和追溯点的概念。

**• 时间戳：**dfn[u]表示节点u深度优先遍历的序号。

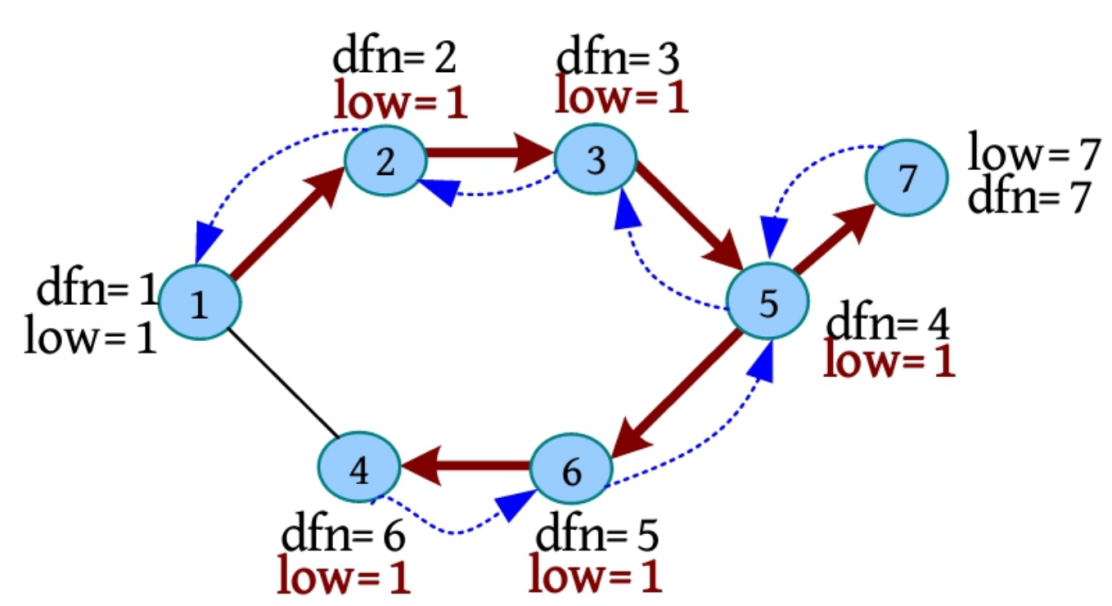
**• 追溯点：**low[u]表示节点u或u的子孙能通过非父子边追溯到的dfn最小的节点序号，即回到最早的过去。

例如，在深度优先搜索中，每个点的时间戳和追溯点的求解过程如下。

初始时，dfn[u]=low[u]，如果该节点的邻接点未被访问，则一直进行深度优先遍历，1-2-3-5-6-4，此时4的邻接点1已被访问，且1不是4的父节点，4的父节点是6（深度优先搜索树上的父节点）。



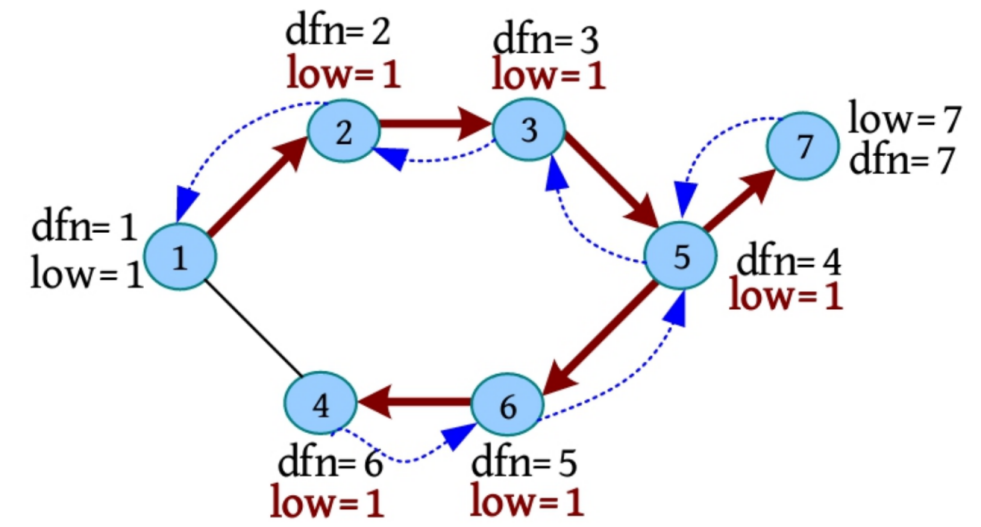
那么节点4能回到最早的节点是节点1（dfn=1），因此low[4]=min(low[4],dfn[1])=1。返回时，更新low[6]=min(low[6],low[4])=1。更新路径上所有祖先节点的low值，因为子孙能回到的追溯点，其祖先也能回到。



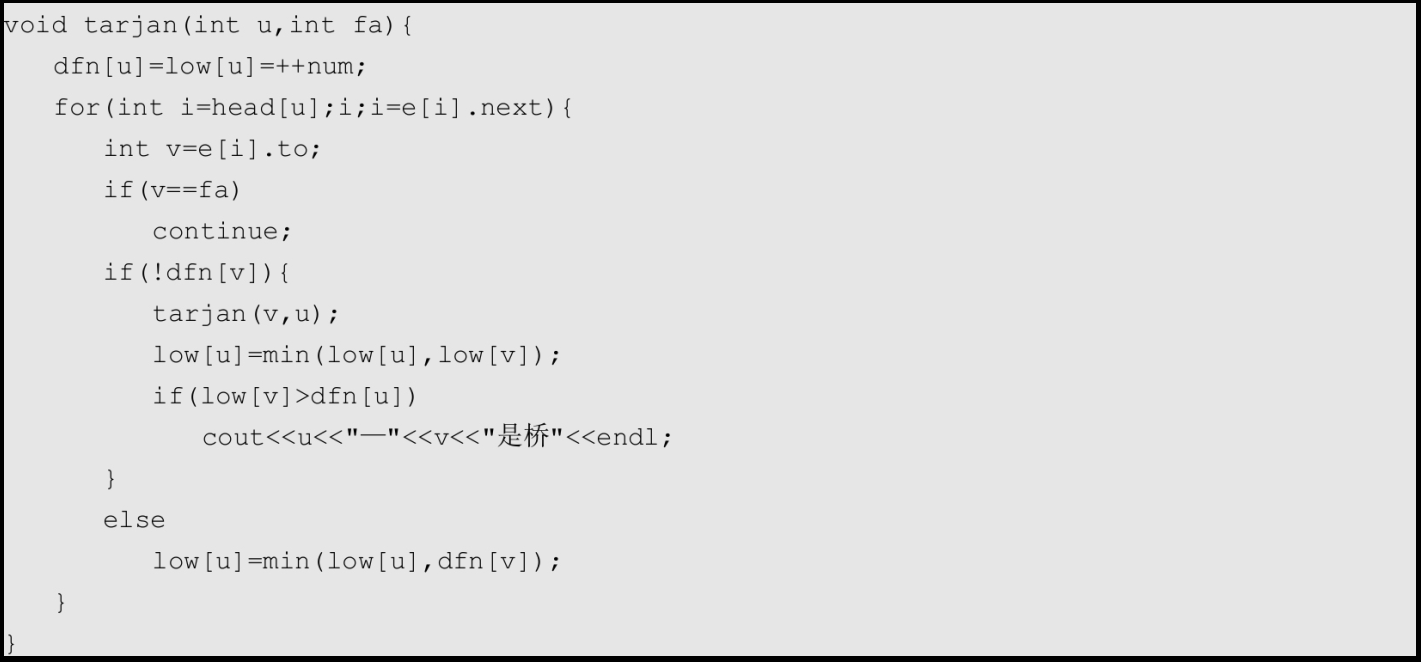
1. **无向图的桥**

**桥判定法则：**无向边x-y是桥，当且仅当在搜索树上**存在x的一个子节点y**时，**满足low[y]>dfn[x]**。

也就是说，若孩子的low值比自己的dfn值大，则从该节点到这个孩子的边为桥。在下图中，边为5-7，5的子节点为7，满足low[7]>dfn[5]，因此边5-7为桥。



**算法代码：**



1. **无向图的割点**

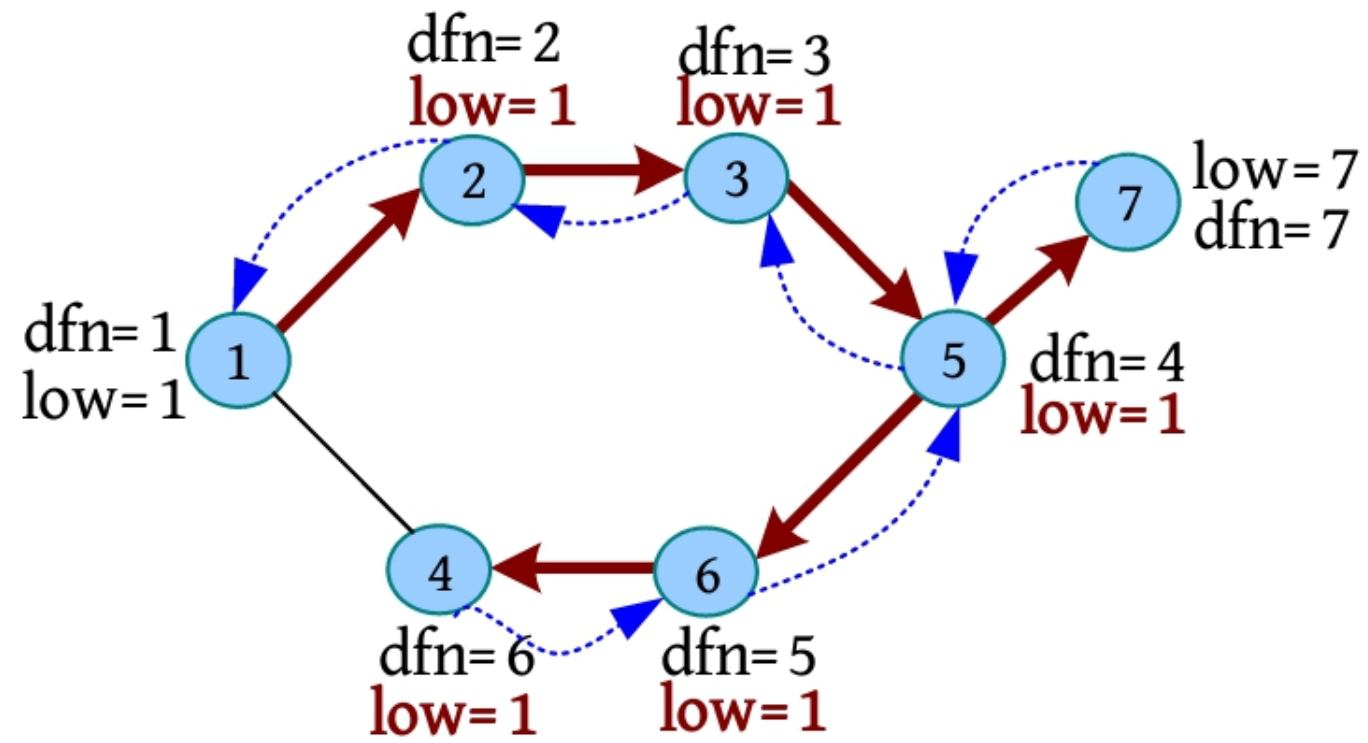
**割点判定法则：**

（1）若**x不是根节点，**则x是割点，当且仅当在搜索树上**存在x的一个子节点y**，**满足low[y]≥dfn[x]**；

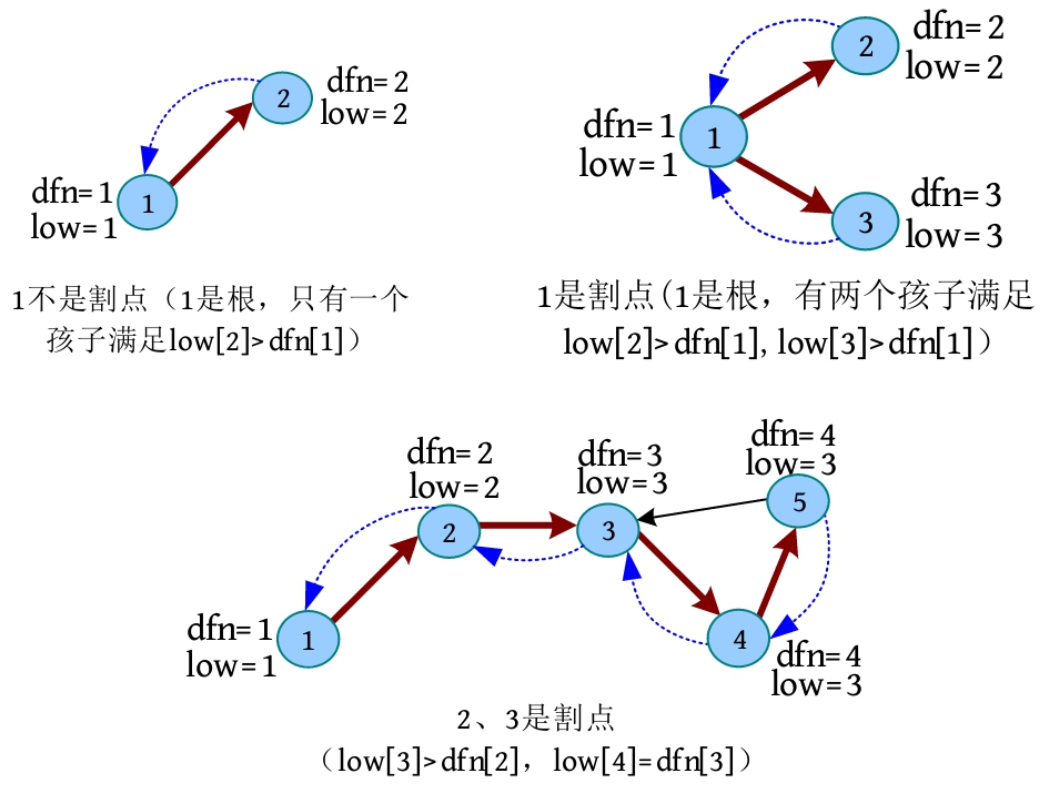
（2）若**x是根节点**，则x是割点，当且仅当在搜索树上**至少存在两个子节点y和z**，满足**low[y]≥dfn[x]和low[z]≥dfn[x]**。

也就是说，如果不是根，且孩子的low值大于或等于自己的dfn值，则该节点就是割点；

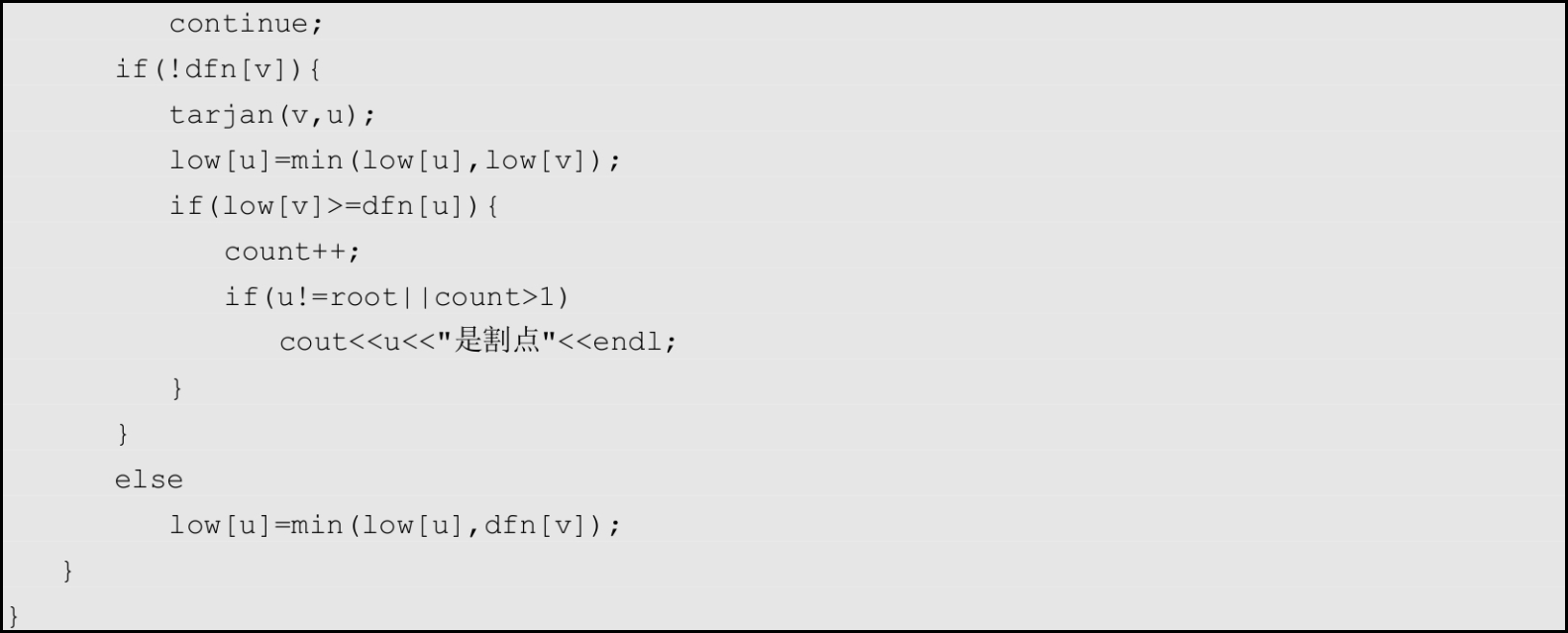
如果是根，则至少需要两个孩子满足条件。在下图中，5的子节点是7，满足low[7]>dfn[5]，因此5是割点。



**所有情况的割点判定，如下图所示。**



**算法代码：**



**3. 有向图的强连通分量算法步骤：**

（1）**深度优先遍历节点，在第1次访问节点x时，将x入栈**，且dfn[x]=low[x]=++num。

（2）**遍历x的所有邻接点y**。

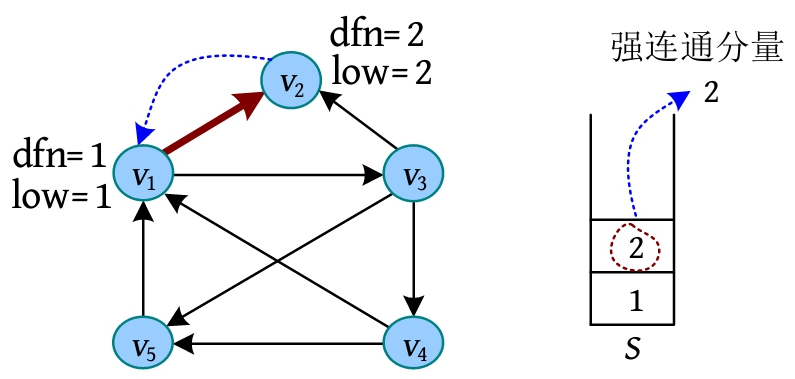
• 若y没被访问，则递归访问y，返回时更新low[x]=min(low[x],low[y])。

• 若y已被访问且在栈中，则令low[x]=min(low[x],dfn[y])。

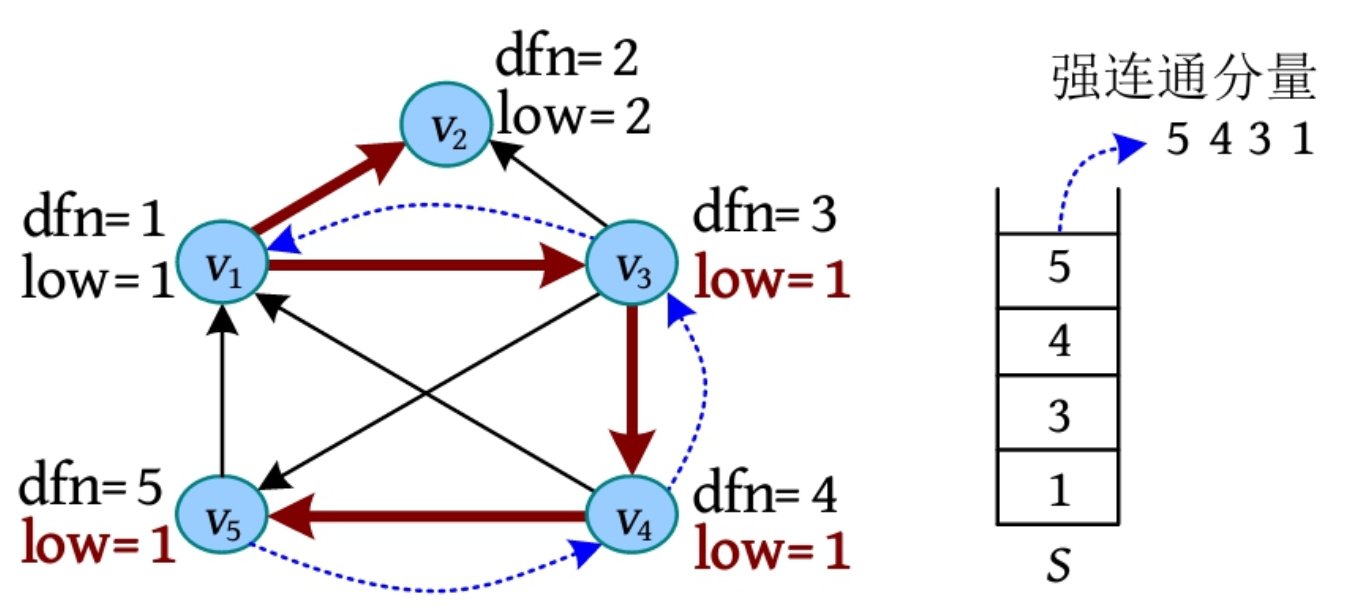
（3）**在x回溯之前，如果判断low[x]=dfn[x]，则从栈中不断弹出节点，直到x出栈时停止。弹出的节点就是一个连通分量**。

例如，求解有向图的强连通分量，过程如下。

① 从节点1出发进行深度优先搜索，dfn[1]=low[1]=1，1入栈；dfn[2]=low[2]=2，2入栈；此时无路可走，回溯。因为dfn[2]=low[2]，所以2出栈，得到强连通分量2。



② 回溯到1后，继续访问节点1的下一个邻接点3。接着访问3-4-5，5的邻接点1的dfn已经有解，且1在栈中，更新low[5]=min(low[5],dfn[1])=1。回溯时更新low[4]=min(low[4],low [5])=1，low[3]=min(low[3],low[4])=1，low[1]=min(low[1],low[3])=1。节点1的所有邻接点都已访问完毕，因为dfn[1]=low[1]，所以开始出栈，直到遇到1，得到强连通分量5 4 3 1。



**算法代码：**

