# 双联通分量

无向连通图或有向连通图中，如果删除某点后，图变成不连通，则称该点为割点。

无向连通图或有向连通图中，如果删除某边后，图变成不连通，则称该边为桥。

双连通分量又分点双连通分量和边双连通分量两种。双联通分量是对无向图说的。

若一个无向图中的去掉任意一个节点都不会改变此图的连通性，即不存在割点，则称作点双连通图。

若一个无向图中的去掉任意一个边都不会改变此图的连通性，即不存在桥，则称作边双连通图。

一个无向图中的每一个极大 点或边双连通子图，称作此无向图的点（边）双连通分量。

求双连通分量可用[Tarjan算法](https://baike.baidu.com/item/Tarjan%E7%AE%97%E6%B3%95)。

## 求割点的Tarjan算法：

**(1)、dfn[ ]，和强联通分量一样，表示这个点在dfs时是第几个被搜到的。**

**(2)、low[u]** 是u或者u的子树中能够通过非父子边(父子边就是搜索树上的边）追溯到的最早的节点的

dfn值

无向联通图中，各个点互相可达，我们从任意一个起点开始进行深度优先搜索，可以得到一棵树，并且这棵树中所有结点的子树之间不存在边，即没有跨越两棵子树的边（考虑一下，如果存在，那么与深度优先搜索树的定义互相矛盾）。于是有如下定理：

* 对于某个点u是根结点，u为割点，当且仅当它有两个或者多个子结点，原因是根节点若只有1个子树那去掉它也无所谓，有2个以上，去掉它会使它的子树不连通
* 对于某个点u是非根结点，u为割点，当且仅当树中u的所有子孙节点中，不存在点v,能连到u的祖先节点的点(连到u本身没事)，代码表现为Low[v]>=DFN[u]，因为这样就保证了u子孙中每个点最多只会到达u本身，而不会到达u的祖先节点。

## 编码细节：

Vis数组标记有没有访问，now代表当前走的点，next是下一个节点

* 对于一个访问过的也就是vis[now]==1的点，, 只有一下情况才更新low[now]=min(low[now],dfn[next]); 记录dfs的父节点p，下一个节点next不是p,且下一个节点的dfn比当前的小， 即dfn[next] < dfn[now]&&next!=p时候
* 对于重边，自环要考虑，需要提前清除
* 时间复杂度总体等同于 强联通分量的tarjan算法

## 求桥的Tarjan算法：

桥割点关系是：两个相邻割点之间的边是桥，桥也叫割边(cutEdge)

* 那么对于树上某一个点u来说，以它为起点边是桥，当且仅当树中u的所有子孙节点中，不存在点v,能连到u以及u的祖先节点的点(连到u本身也不行),代码表现为Low[v]>DFN[u],和割点类似

## 求割点所分成的点双联通分量，以及割边分成的边双联通分量：

点双联通分量：比较麻烦，用一个栈记录 边。一定记录边，而不是点，因为割点会被重复访问，没法确定这个点被几个双联通分量包含。每次更新low的时候入栈，之后每次找到一个割点时候出栈，直到等于当前当前dfs访问的连接now和next的这条边为止。这里要注意对于出栈只有一条边的情况，不是点双联通分量形成的边，因为只有一条边时候，它是两个相邻割点形成的边。求点双联通分量的栈最后一定会被出空，不用考虑是否有剩余元素。

还有这些边不能重复，因此在之前割点代码里写的条件判断要精确，dfn[now]&&next!=p绝不能少

得到点双联通分量的边集，再遍历处理成点即可。

边双联通分量：可以用电双联通分量的办法记录维护一个栈，代码类似，但是需要注意的是，求边双联通分量的栈不会被出空，因为找边双联通分量，不区分当前找的点是不是树根，最后会剩余一组.

再有就是，除了最后剩余的那组，每次最后一个出栈的正好是桥，不用输出，因为桥不算是边双联通分量的边集

求边双联通分量还有一种比较简单的方法，但代码麻烦，求出割边同时，在邻接表里标记是不是割边。

可以再做一次无回溯的dfs，已经走过的边和割边都不走，就能得到所有边双联通分量