Tarjan求强连通分量

主讲人：邓哲也

万门中学 zx.wanmen.org

强连通分量的求解

求解有向图的强连通分量主要有两种算法：

Tarjan 算法：DFS 一次原图

Kosaraju算法：DFS 一次原图，DFS一次逆图

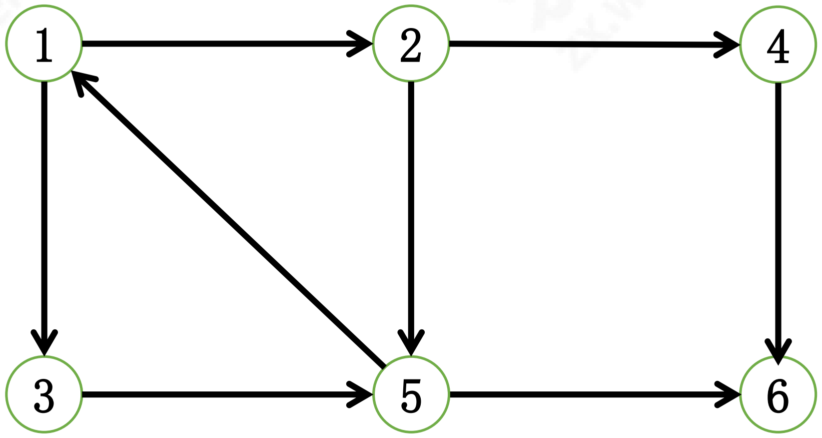
Tarjan算法

这里我们介绍一种只需从某个顶点出发进行一次遍历，就可以求出图中所有强连通分量的 Tarjan 算法。

时间复杂度：$O(n + m)$

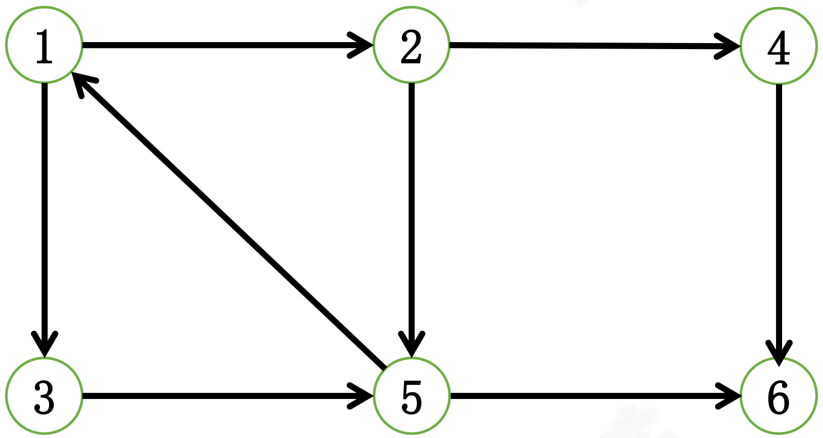
Tarjan算法

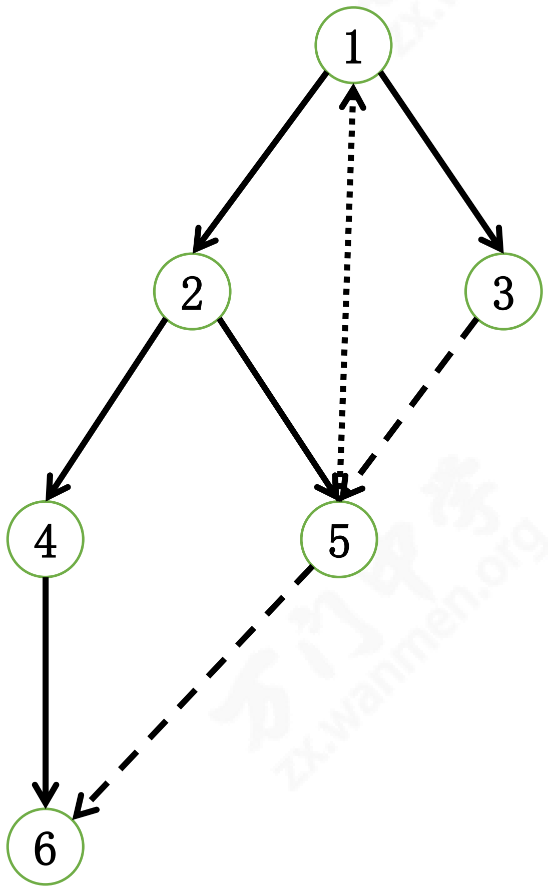
以这张图为例，我们来看看 Tarjan 算法是如何运行的。



Tarjan算法

首先从某个顶点出发进行深度优先搜索后，得到深度优先搜索生成树。

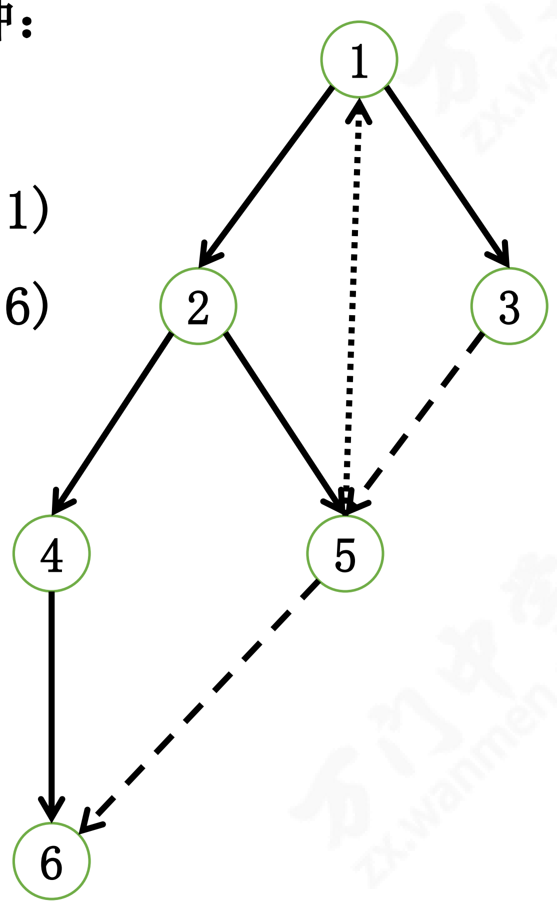




深度优先搜索生成树

深度优先搜索生成树中的边可以分为三种：

1. 生成树的边。例如：（1，2），（4，6）
2. 返向边。例如图中点线表示的边(5，1)
3. 交叉边。例如图中虚线表示的边(5，6)



判断边的种类

```
对顶点 u 的邻接顶点 v
    如果 v 还没有访问          // <u, v> 是生成树的边
        从 v 出发进行 DFS
    否则
        如果 v 还在栈中        // v 是 u 的祖先
            <u, v> 是返向边
        否则
            <u, v> 是交叉边
```

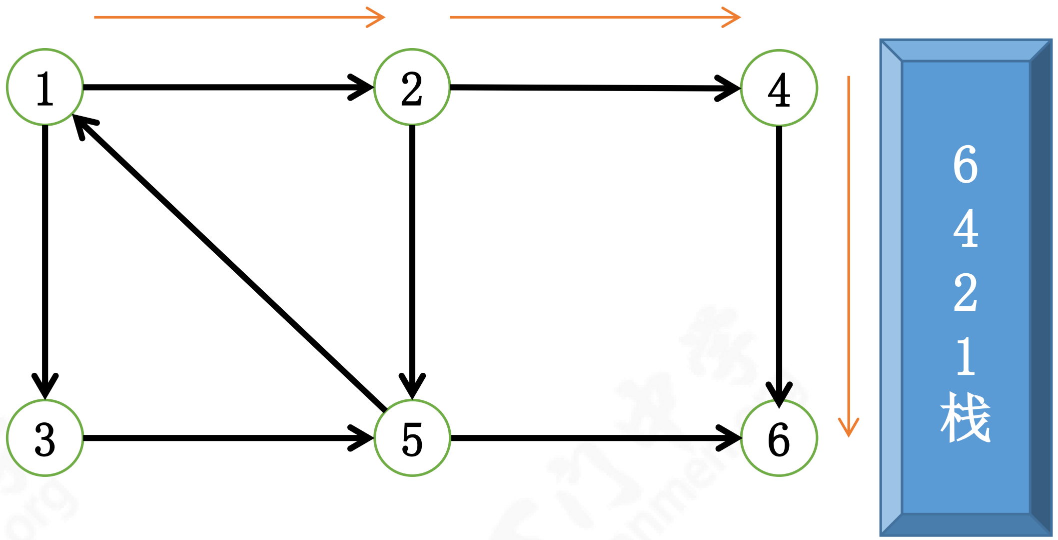
Tarjan算法伪代码

```
tarjan(u) {
        dfn[u] = low[u] = ++ tot;
        stack.push(u);
        for (u, v) in E:
                if (!vis[v])
                        tarjan(v)
                        low[u] = min(low[u], low[v])
                else if (v in stack)                    // 这是返向边
                        low[u] = min(low[u], dfn[v])
        if (dfn[u] == low[u])                           // 找到一个强连通分量
                repeat
                        v = stack.pop()
                        print(v)
                until u == v
}
```

Tarjan算法模拟

从 1 开始 DFS，沿着箭头方向一直搜索到顶点 6。

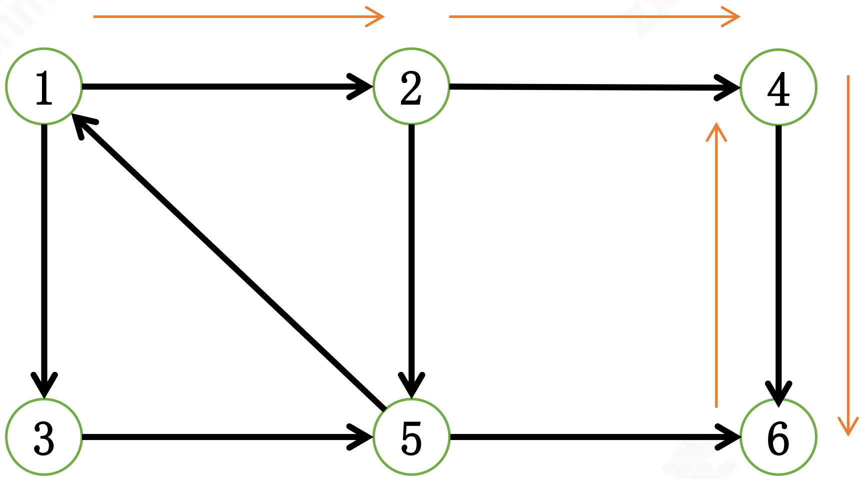
此时无法继续前进，并且由于 low[6] = dfn[6] = 4，所以这是一个强连通分量。

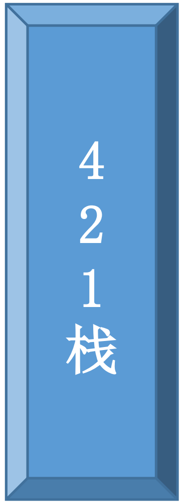


	1	2	3	4	5	6
dfn	1	2		3		4
low						4

Tarjan算法模拟

回退到 4，发现 low[4] = dfn[4] = 3，又是一个强连通分量。





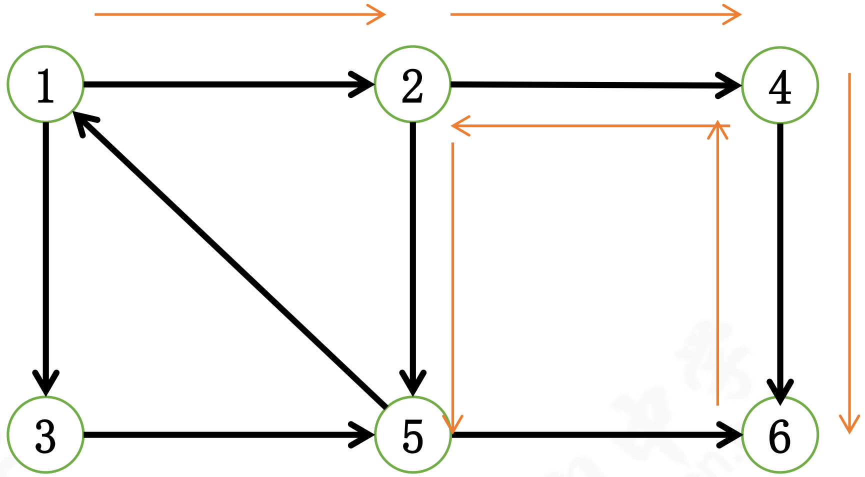
	1	2	3	4	5	6
dfn	1	2		3		4
low				3		4

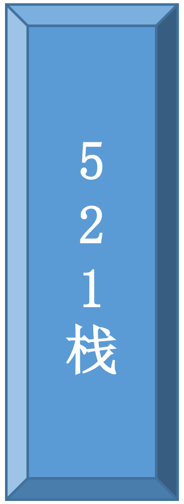
Tarjan算法模拟

回退到 2 并继续搜索 5.

5 有到 1 的有向边，且 1 还在栈中，更新 low[5]

5 有到 6 的有向边，但 6 不在栈中，这是交叉边。

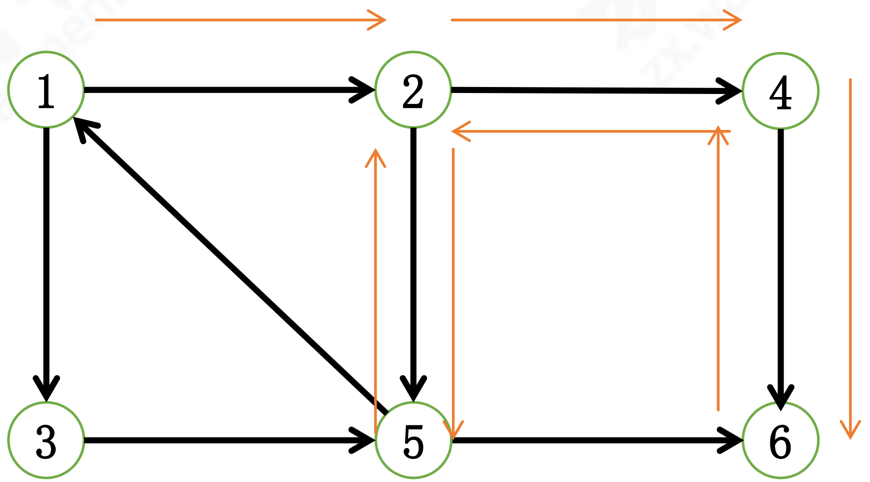


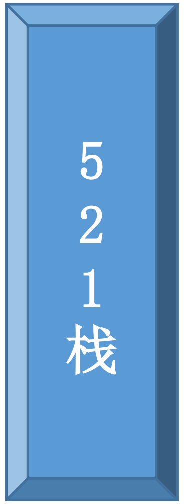


	1	2	3	4	5	6
dfn	1	2		3	5	4
low				3	1	4

Tarjan算法模拟

返回顶点 2，更新 low[2]



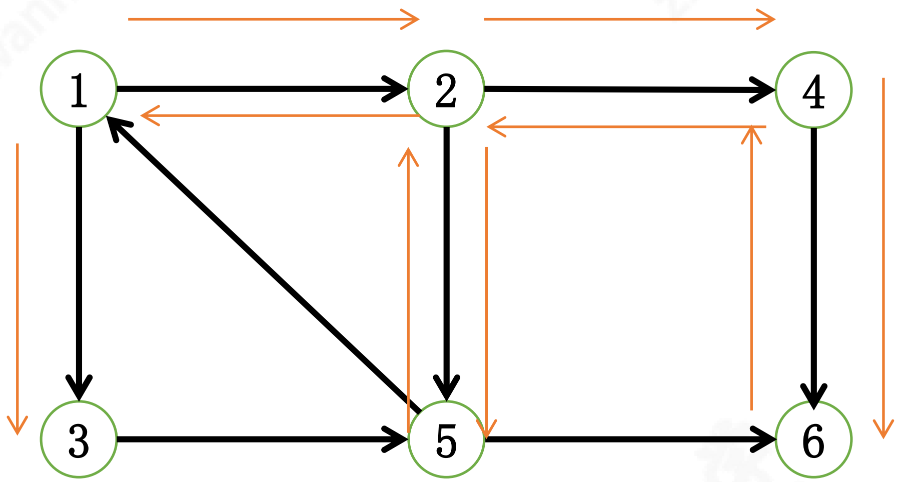


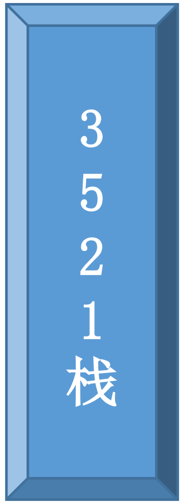
	1	2	3	4	5	6
dfn	1	2		3	5	4
low		1		3	1	4

Tarjan算法模拟

返回顶点 1，然后搜索 3.

3 到 5 有一条边，5 还在栈中，更新 low[3]



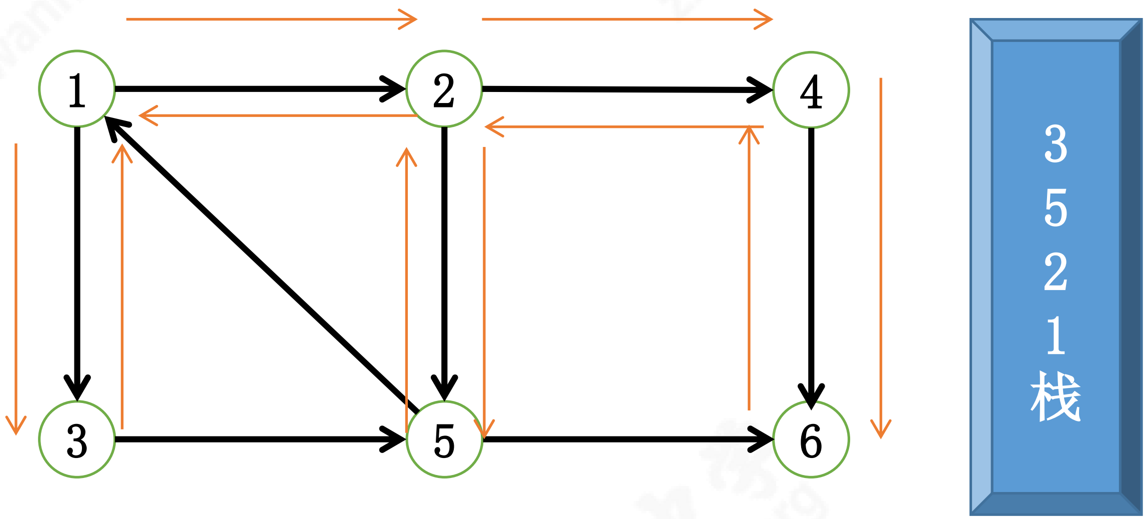


	1	2	3	4	5	6
dfn	1	2	6	3	5	4
low		1	5	3	1	4

Tarjan算法模拟

返回 1，发现 low[1] = dfn[1] = 1

把栈中的顶点全部弹出，组成一个连通分量 {1,2,3,5}

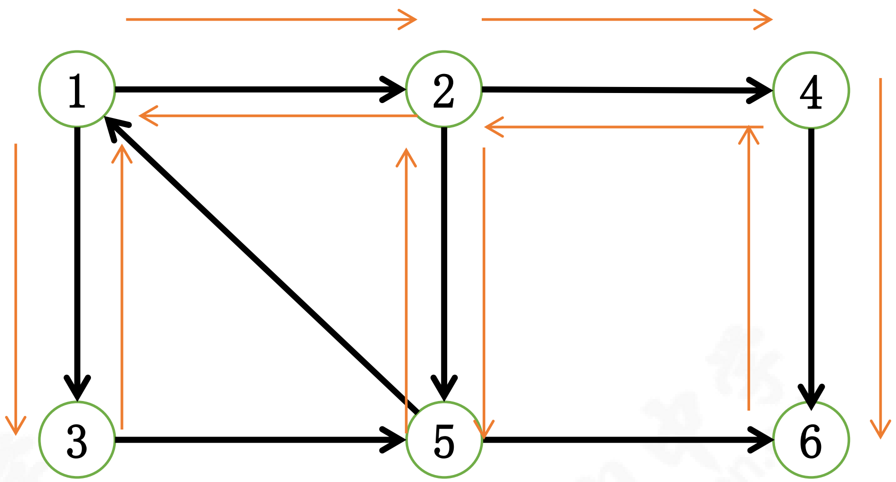


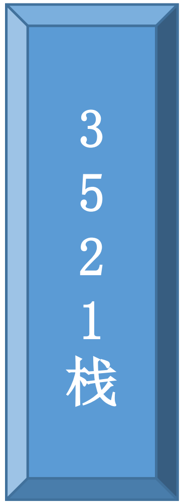
	1	2	3	4	5	6
dfn	1	2	6	3	5	4
low	1	1	5	3	1	4

Tarjan算法时间复杂度

假设用邻接表存图，每个点都被访问一次，每个点都进出栈一次，每条边也只被访问一次。

所以该算法的时间复杂度为 O(n + m)。





	1	2	3	4	5	6
dfn	1	2	6	3	5	4
low	1	1	5	3	1	4

下节课再见

万门中学 zx.wanmen.org