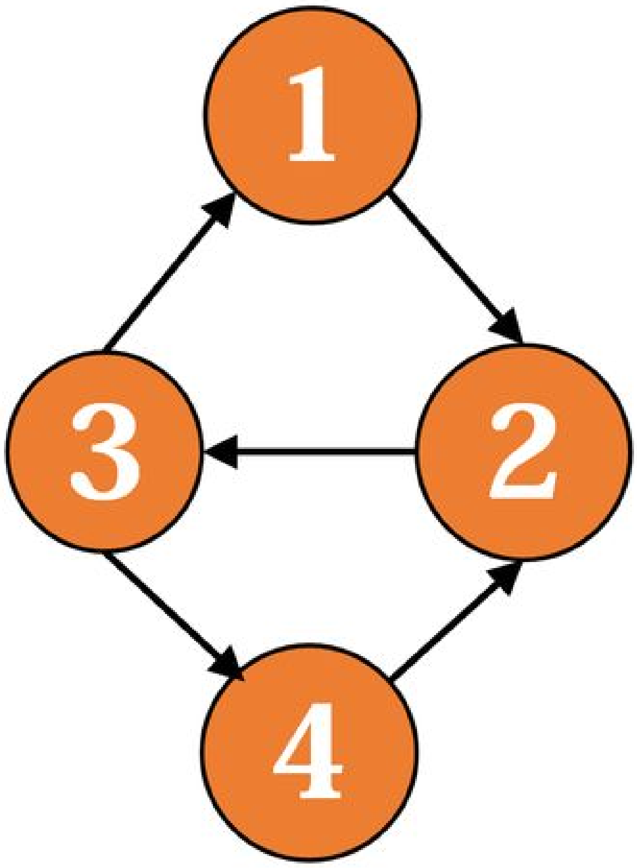
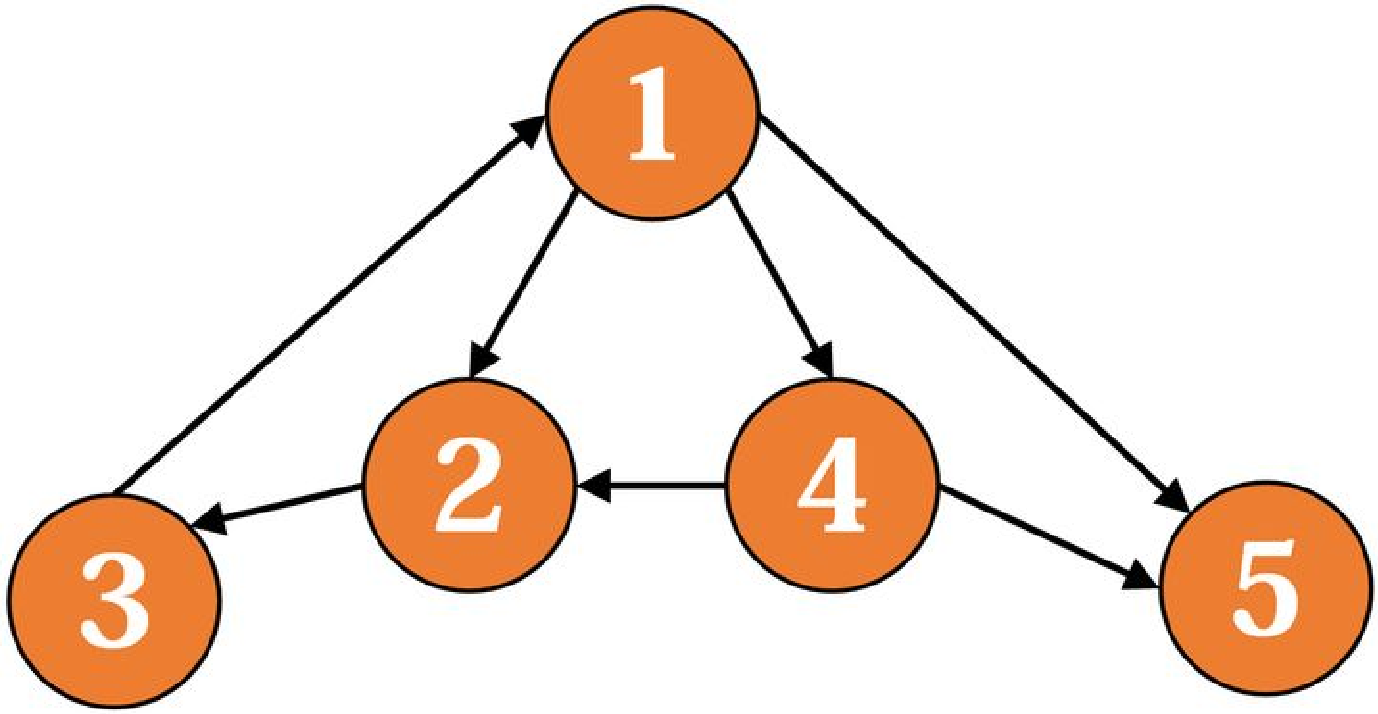
强连通分量

对于一张**有向图**而言，它是**强连通**的当且仅当其上每两个顶点都相互可达。强连通图类似于嵌套的**环**，强连通图一定有环，但 n 个节点的强连通图不一定有 n 元环：

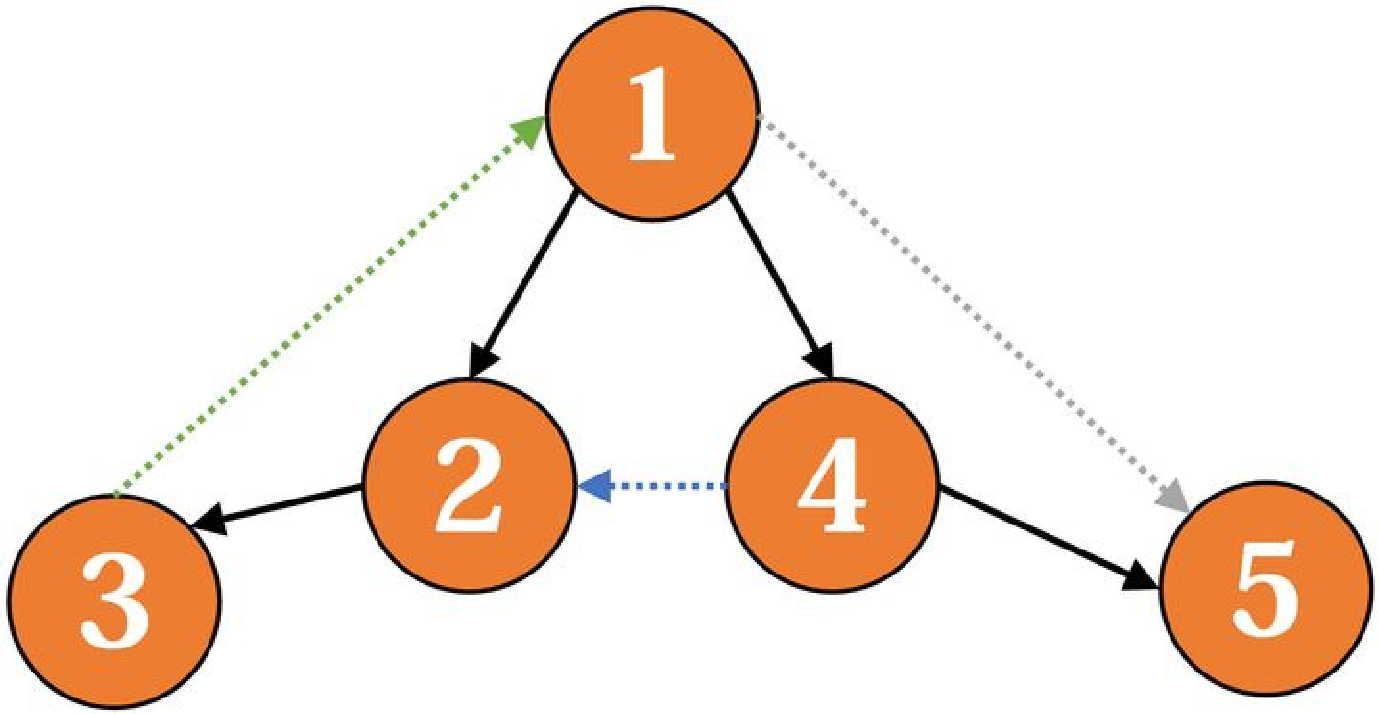


强连通分量是有向图的极大的强连通子图，所谓“极大”意味着，把图划分为若干个强连通分量后，不存在两个强连通分量相互可达。

处理强连通分量的一个有力的工具是**dfs生成树**：在dfs时，每当通过某条边 e 访问到一个新节点 v ，就加入这个点和这条边，最后得到的便是dfs生成树。例如对于下面这张有向图：



它的dfs生成树可能是这样（黑色实线）：

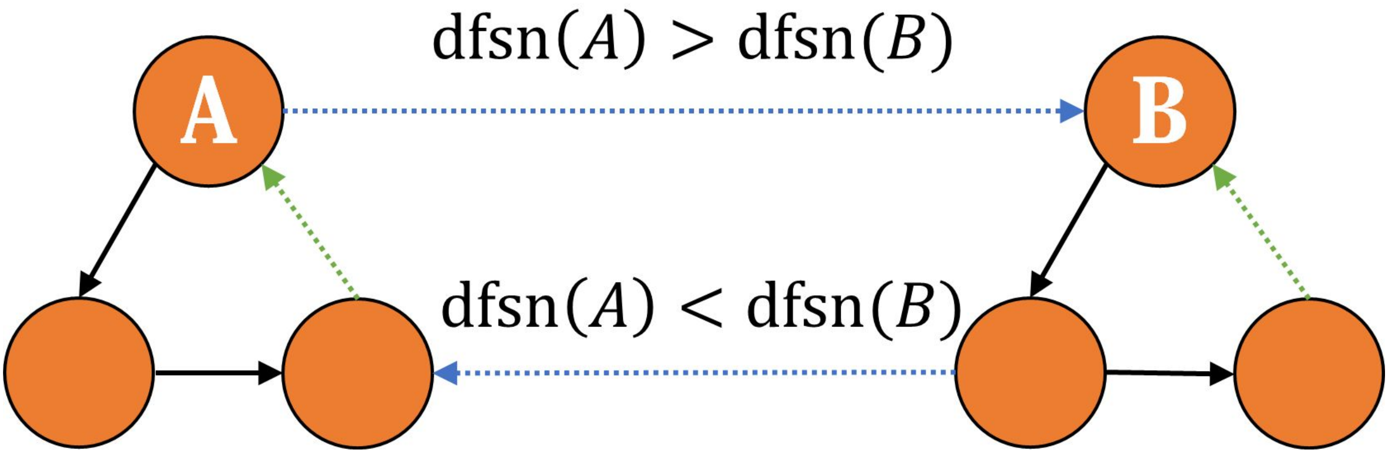


除了生成树的**树边**外，原图剩下的边可以分为三种：

前向边（上图灰色虚线）：从某个点到它的某个子孙节点的边。这种边相当于提供某种“捷径”，在这个问题里不太重要，即使把它们全部删去，对于连通性也没什么影响。
反向边（上图绿色虚线）：从某个点到它的某个祖先节点的边。这种边就是产生环的原因，如果删去所有反向边，那么原图会成为有向无环图。
横叉边（上图蓝色虚线）：从某个点到一个既非它子孙节点、也非它祖先节点的边。这种边本身不产生环，但是它可能把两个强连通子图“连接”起来，形成一个更大的强连通子图。

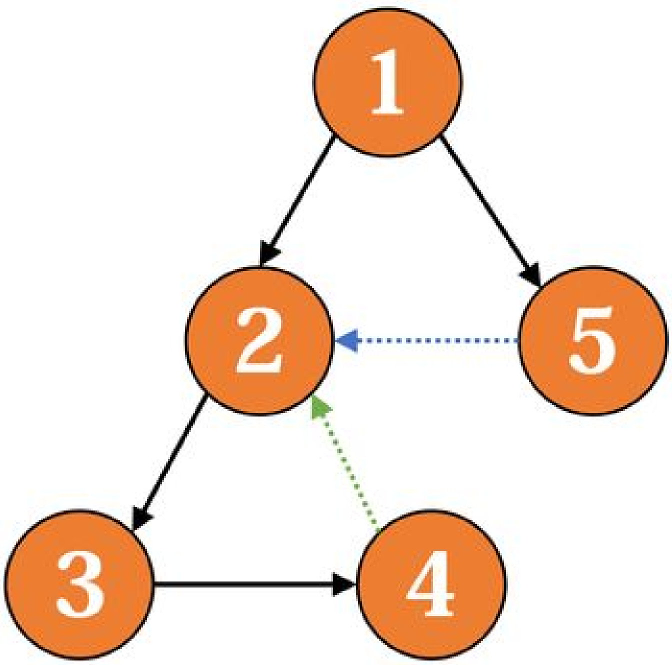
反向边和**横叉边**都有一个特点：**起点的dfs序必然大于终点的dfs序。**[1]

这可以导出一个有用的结论：**对于每个强连通分量，存在一个点是其他所有点的祖先。**若不然，则可以把强连通分量划成 n 个分支，使各分支的祖先节点互相不为彼此的祖先。这些分支间不能通过树边相连，只能通过至少 n 条横叉边相连，但这必然会违背上一段讲的性质。



我们把这个唯一的祖先节点称为强连通分量的**根**。显然，根是强连通分量中dfs序最小的节点。

为了求强连通分量，我们常常使用**Tarjan算法**。首先，我们把 $\mathrm{low}(p)$ 定义为 p **所在子树的节点经过最多一条非树边 $u \to v$ （其中 v 可达 u ）能到达的节点中最小的dfs序**。根据这样的定义，某个点 p 是强连通分量的根，等价于 $\mathrm{dfsn}(p) = \mathrm{low}(p)$ [2][3]。我们这里必须强调 v 可达 u ，否则在下图中，会使 $\mathrm{low}(5) = 2$ ，但它实际应是一个强连通分量的根。



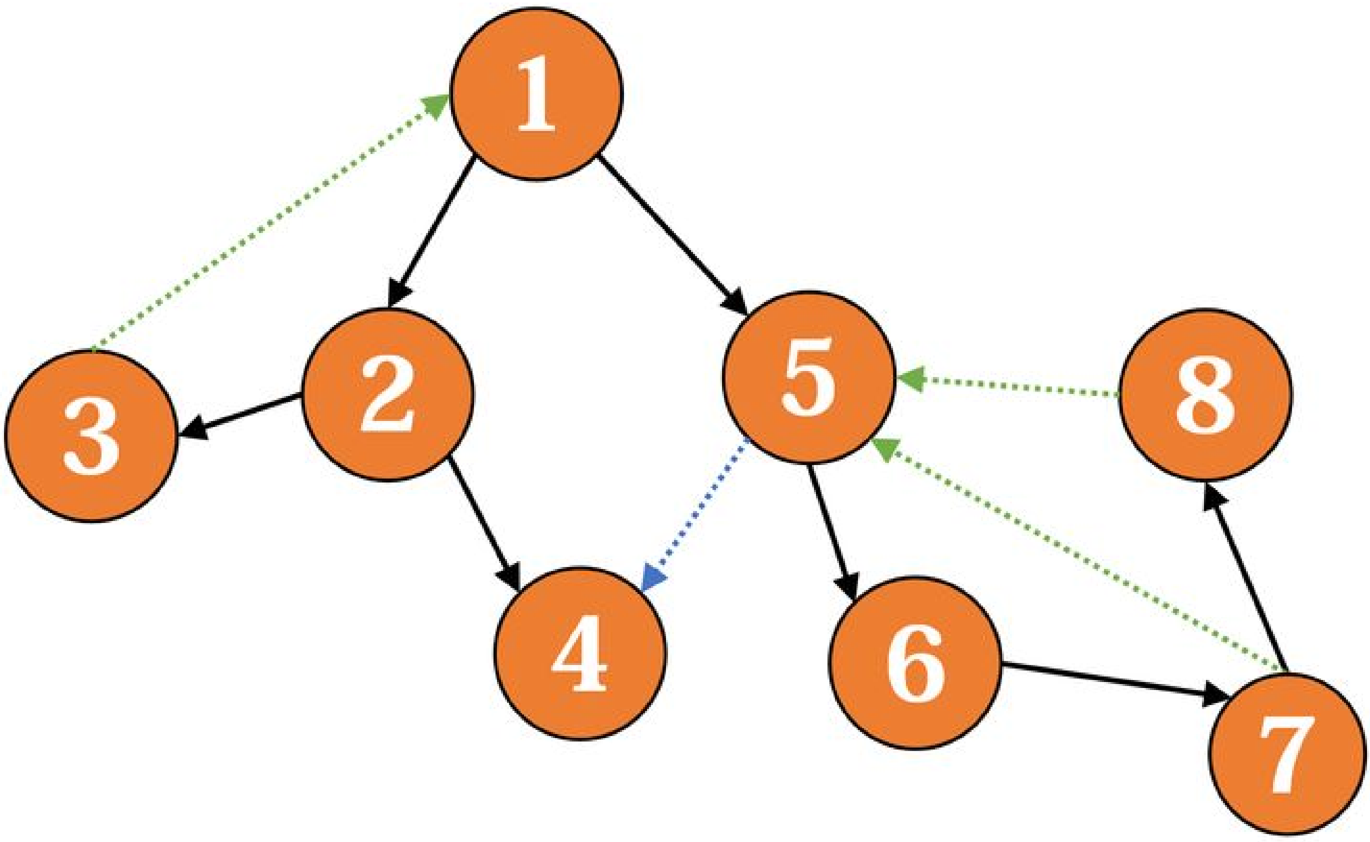
$\mathrm{low}(p)$ 可以通过动态规划得到，对于以某个点 p 为起点的边 $p \to q$ ：

如果 q 未访问过，则 q 在 p 所在的子树上，如果某节点 r 从 q 起可以经过最多一条反向边到达，则从 p 起也可以（先从 p 到 q ，再到 r ），于是先递归处理点 q ，然后令 $\operatorname{low}(p) := \min(\operatorname{low}(p), \operatorname{low}(q))$ 。
如果 q 已访问过，且从 q 可以到达 p ，令 $\operatorname{low}(p) := \min(\operatorname{low}(p), \operatorname{dfsn}(q))$ 。
如果 q 已访问过，且从 q 不能到达 p ，不做处理。（后两种情况都是非树边）

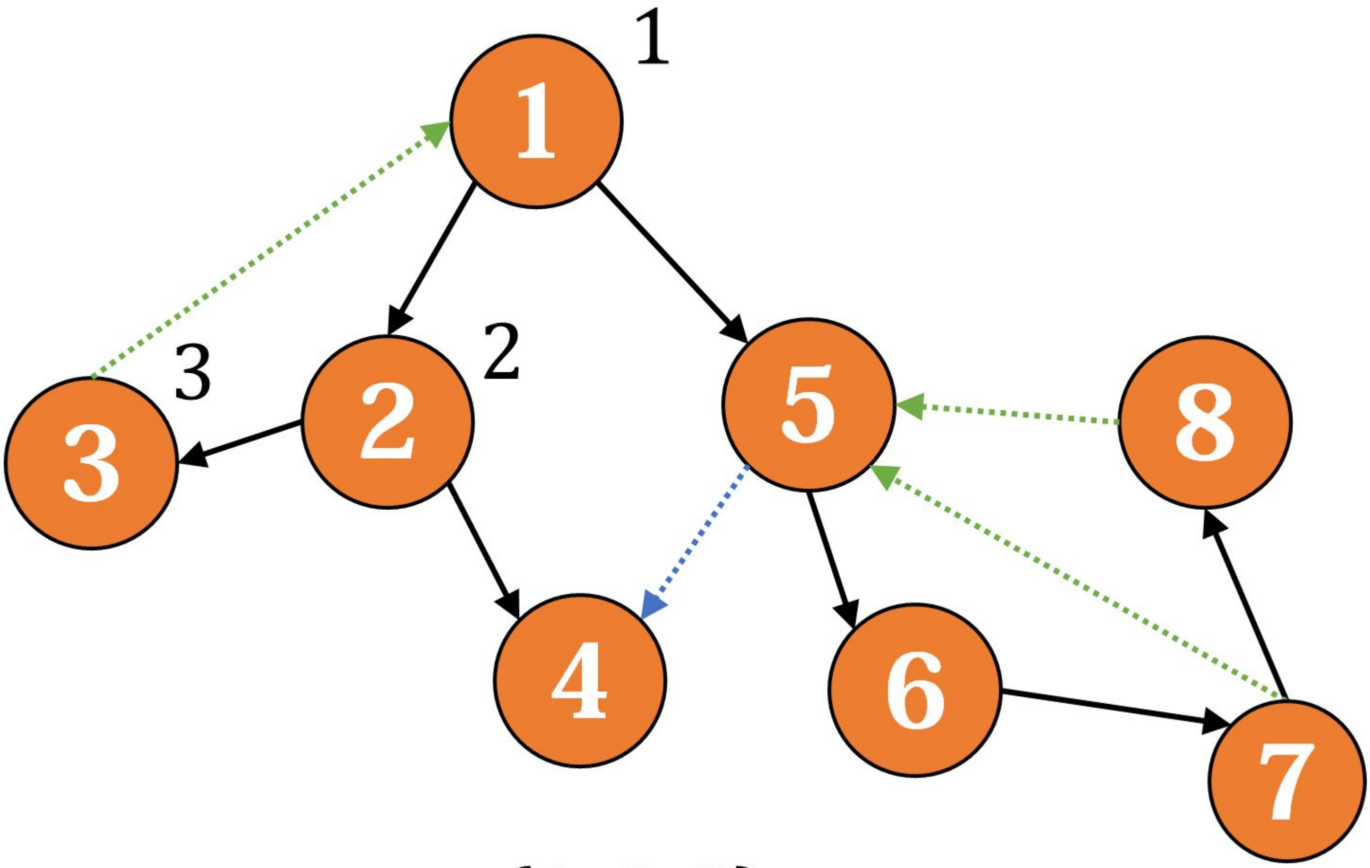
但是我们怎么确认一个点能不能到达另一个点呢？因为**反向边**和**横叉边**都指向dfs序较小的节点，而**前向边**的存在又不影响状态转移方程，所以我们只需要确认**比该点dfs序小**的哪些点能到达该点即可，这可以用一个**栈**动态地维护。

每当搜索到新点，就令它入栈。在对点 p 的搜索结束时，如果 $\operatorname{low}(p) = n < \operatorname{dfsn}(q)$ ，设dfs序为 n 的点为 q ，则 p 点可达的点 q 点都可达，考虑到对 q 点的搜索很可能还没有结束，所以 p 应当留在栈中。而如果发现 p 满足 $\operatorname{low}(p) = \operatorname{dfsn}(p)$ ，则说明 p 是某个强连通分量的根，它和栈中的子孙节点相互可达。但同时，它和栈中的子孙节点也无法到达 p 的祖先节点，以及祖先节点其他尚未搜索的分支了，所以不断从栈顶弹出元素，直到弹出 p （注意这样维护的栈中节点的dfs序是单调增的），同时记录答案。

我们以下图来举例（这里为了方便，直接用dfs序作为点的编号）。

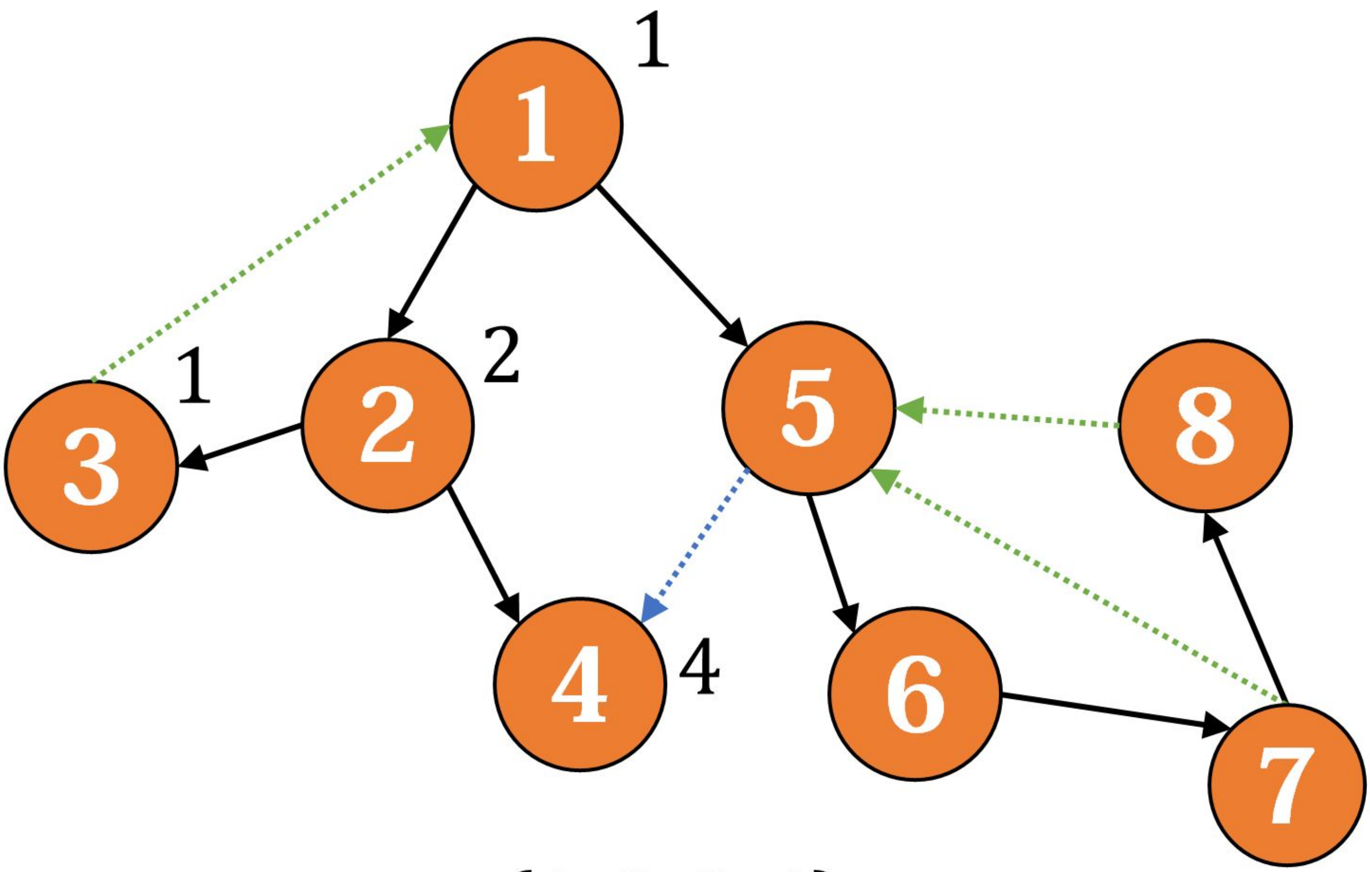


我们沿着 $1 \to 2 \to 3$ 搜索（节点右上角表示当前的 low ，下方为栈）：



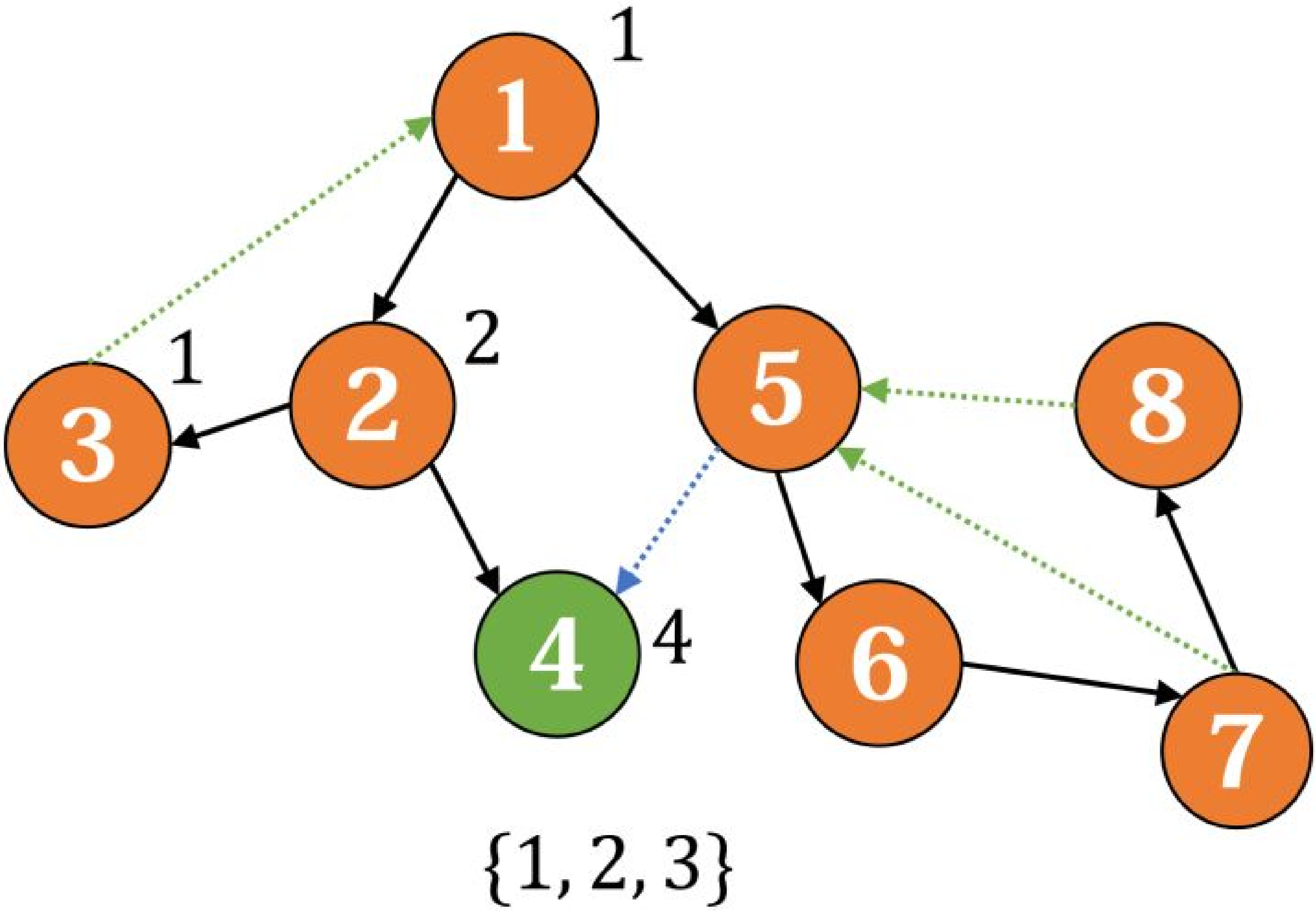
$\{1, 2, 3\}$

发现反向边 $3 \to 1$ ，更新 $\mathrm{low}(3)$ ，然后回到 2 搜索 4 ：

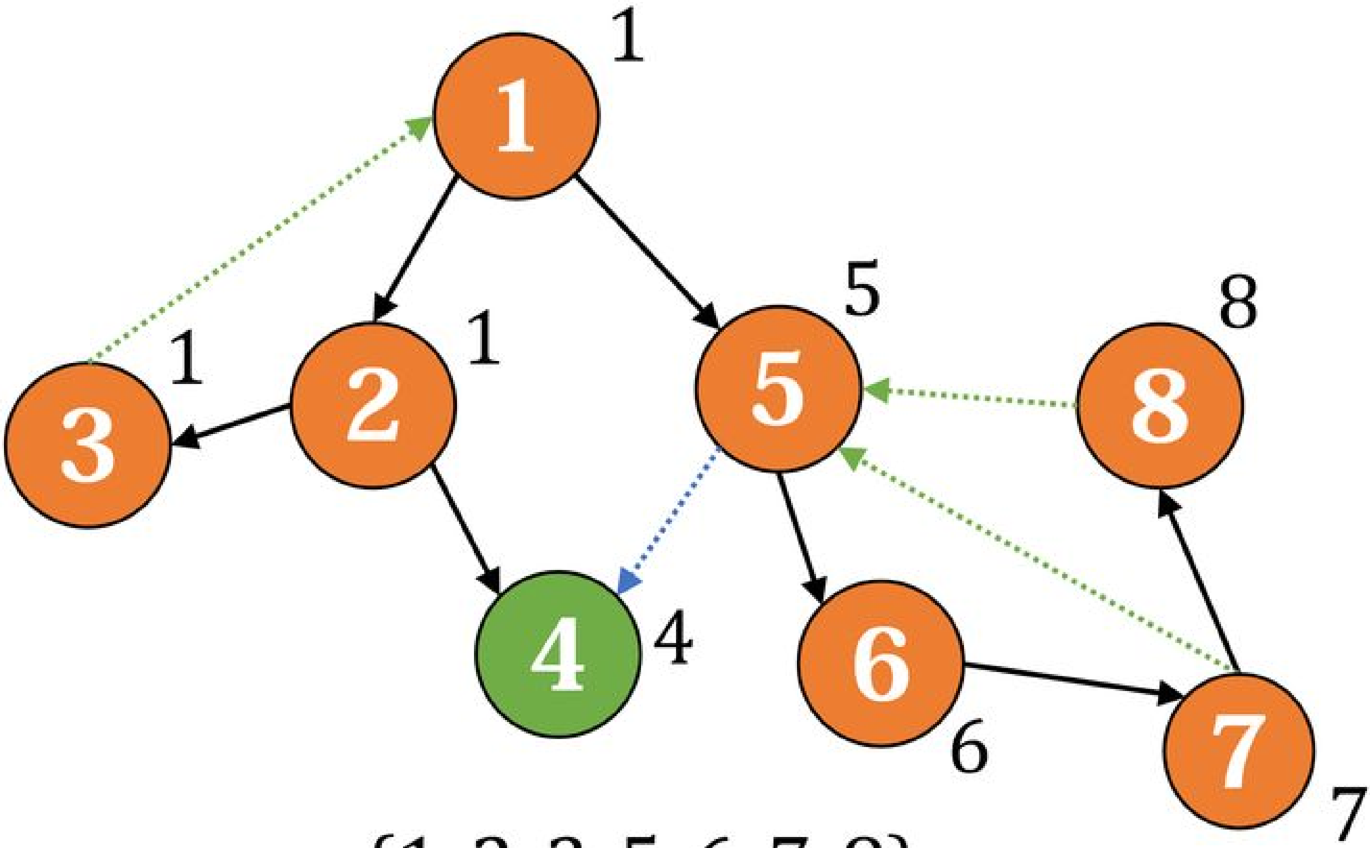


$\{1, 2, 3, 4\}$

4没有出边了，返回，这时发现 $\mathrm{low}(4)=\mathrm{dfsn}(4)=4$ ，于是出栈，标记它属于第一个强连通分量。

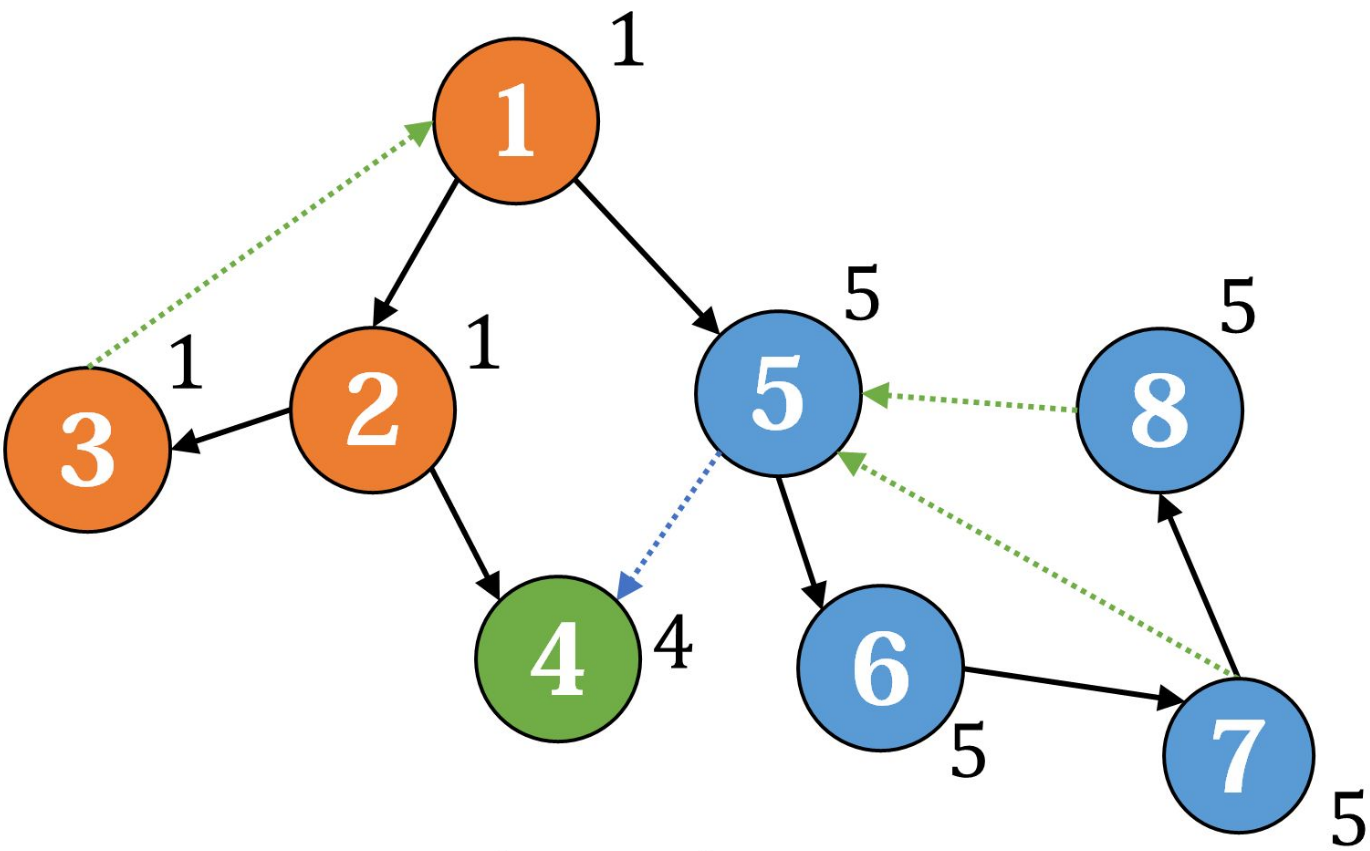


回到 2 ，它对应的子树已搜索完毕，更新 $\mathrm{low}(2)$ ；接下来搜索 $5 \to 6 \to 7 \to 8$ ：



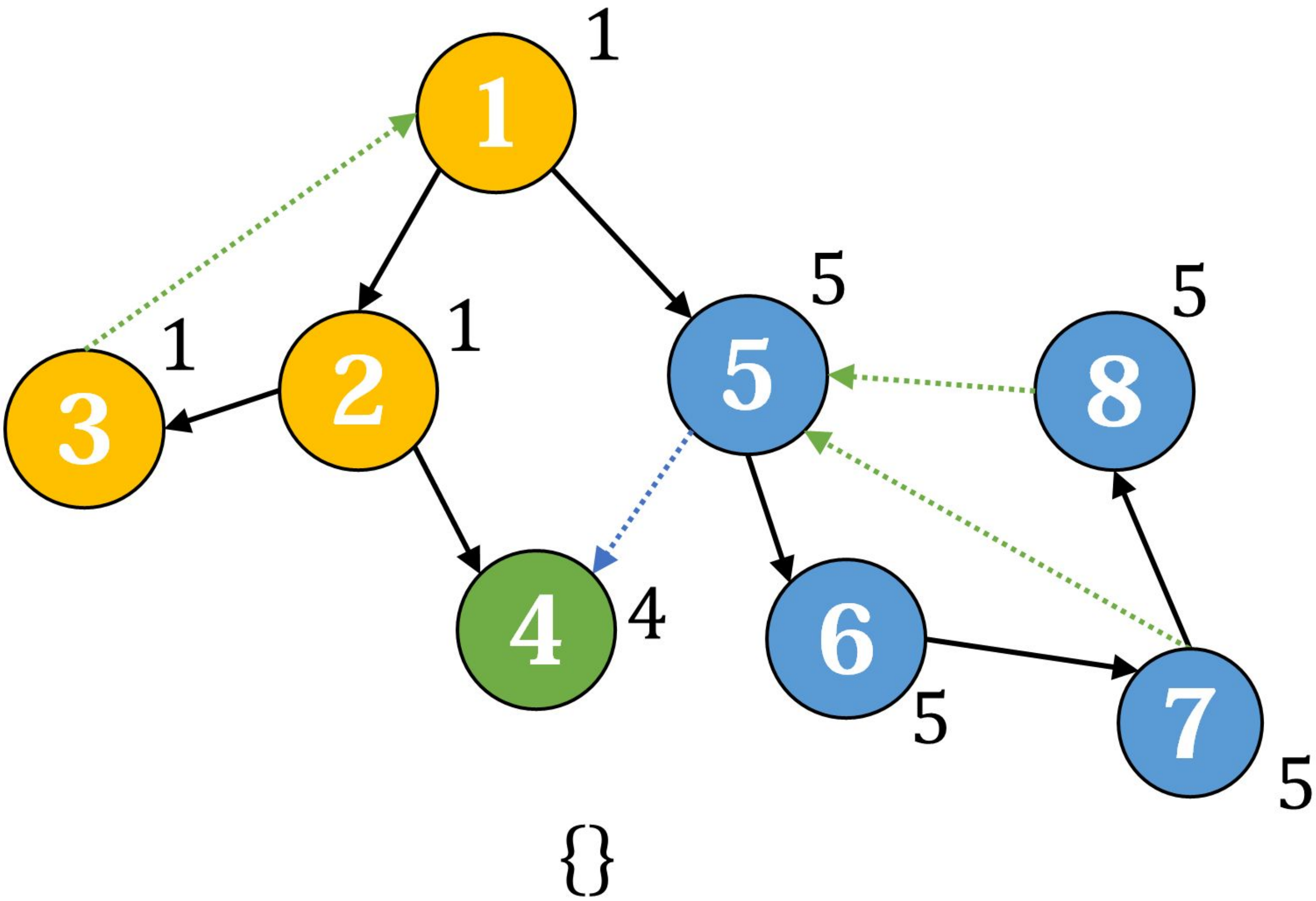
$$\{1,2,3,5,6,7,8\}$$

反顺序更新它们的 low ，到 5 时发现有 $\mathrm{low}(5)=\mathrm{dfsn}(5)=5$ ，于是把它们归入一个新的强连通分量：



$$\{1,2,3\}$$

最后回到 1 ，把剩下还在栈内的节点都归入最后一个强连通分量：



代码如下：

```
stack<int> stk;
// instk: 是否在栈中  scc: 每个点所属的强连通分量编号  cscc: 强连通分量的数量
int dfsn[MAXN], low[MAXN], instk[MAXN], scc[MAXN], cnt, cscc;
void tarjan(int p)
{
    low[p] = dfsn[p] = ++cnt;
    instk[p] = 1;
    stk.push(p); // 进栈
    for (auto q : edges[p])
    {
        if (!dfsn[q]) // 未访问过
        {
            tarjan(q); // 递归地搜索
            low[p] = min(low[p], low[q]);
        }
        else if (instk[q]) // 访问过，且q可达p
            low[p] = min(low[p], dfsn[q]);
    }
    if (low[p] == dfsn[p]) // 发现强连通分量的根
```

```
    {
        int top;
        cscc++;
        do
        {
            top = stk.top();
            stk.pop();
            instk[top] = 0;
            scc[top] = cscc; // 记录所属的强连通分量
        } while (top != p); // 直到弹出p才停止
    }
}
```

注意，由于原图并不一定是强连通图，所以不能随便找一个点作为根dfs就完事，而要保证每个点都被dfs到：

```
for (int i = 1; i <= n; ++i)
    if (!dfsn[i])
        tarjan(i);
```

由于每次dfs结束后，栈都已清空，所以各次dfs并不会互相影响。

在求出强连通分量后，可以进行**缩点**，即：把处于同一个强连通分量的点当作同一个点处理。这样，我们可以得到一张**有向无环图**。此外，注意到编号较小的点不能到达编号较大的点，于是scc编号的顺序还符合拓扑序（编号越大的点拓扑序越靠前）。

```
for (int u = 1; u <= n; ++u)
    for (auto v : edges[u])
        if (scc[u] != scc[v])
            edges2[scc[u]].push_back(scc[v]);
```

例如，在有向图中求经过的点权值之和最大的路径，就可以缩点，每个点的权值为缩之前的点的权值之和，这是因为如果一条路径经过强连通分量中的某个点，就可以经过其中的所有点。对于缩点后得到的有向无环图，可以按照拓扑序DP。

再例如，求有向图中那些所有点都可达的点，也可以缩点，因为如果 p 可达 q ，那么 p 所在的强连通分量中的所有点都可达 q 。对于得到的有向无环图，只需找到唯一的出度为0的点即可。

参考

^ 对于反向边，由于祖先节点的dfs序小于子孙节点，所以是显然的。对于横叉边u→v，由于v既不是u的祖先、也不是u的子孙，所以必然存在一个不同于u、v的点w=lca(u,v)，u和v分别位于两个分支上。u→v没有成为一条树边，这说明v所在的分支一定在u所在的分支之前被访问过，也就是说，分别在u所在分支和v所在分支上任取点p和q，前者的dfs序都一定比后者大，自然也有dfsn(u)>dfsn(v) 。

^ 充分性：设p是某强连通分量的根，其子孙节点u属于该强连通分量且能通过u->v达到某个dfs序比p小的节点v。v和u互相可达，应属同一个强连通分量，但v的dfs序却比p小，这是不可能的。因此，low(p)≥dfsn(p)。同时，p可以通过0条边到达自己，所以low(p)≤dfsn(p)。所以low(p)=dfsn(p)。

^ 必要性：如果low(p)=dfsn(p)，说明p不能到达dfs序比p小的节点，或者说不存在一个强连通分量同时包含p和某个dfs序比p小的节点。因此，p只能是某个强连通分量的根。