OT1-Tarjan’s Algorithm算法介绍

——戴一帆

目录

1、Tarjan’s Algorithm算法作用

Tarjan’s Algorithm是Robert Tarjan提出的一系列基于深度优先搜索算法的解决图论问题的线性时间算法。包括求有向图的强连通分量、必经点、必经边，求无向图的割边、割点、双联通分量。

本次OT会详细介绍如何用Tarjan’s Algorithm求解割边集与割点集，并且会粗略地讲解如何用Tarjan’s Algorithm求解双联通分量。

2、Tarjan’s Algorithm算法实现

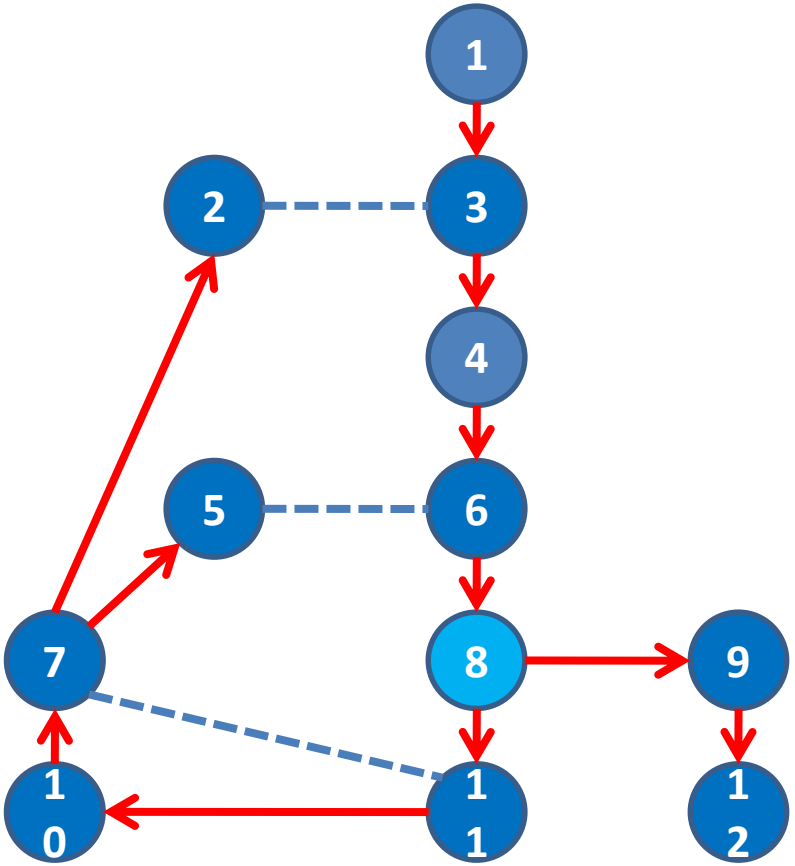
定义：
dfn[u]表示编号为**u**的结点在**dfs**过程中，是第几个被搜索到的。（时间戳）

low[u]表示以编号为**u**的结点的**dfs**树中所有结点、以及与这些结点以一条边相邻的结点中，时间戳最小的结点。

low[u]搜索的结点包括：
(1)、以**u**为根的**dfs**树上的所有结点
(2)、与**(1)**中的结点以一条边相邻的所有结点
例外：不搜索在**dfs**树上以**u**与**u**的父结点为端点的边。

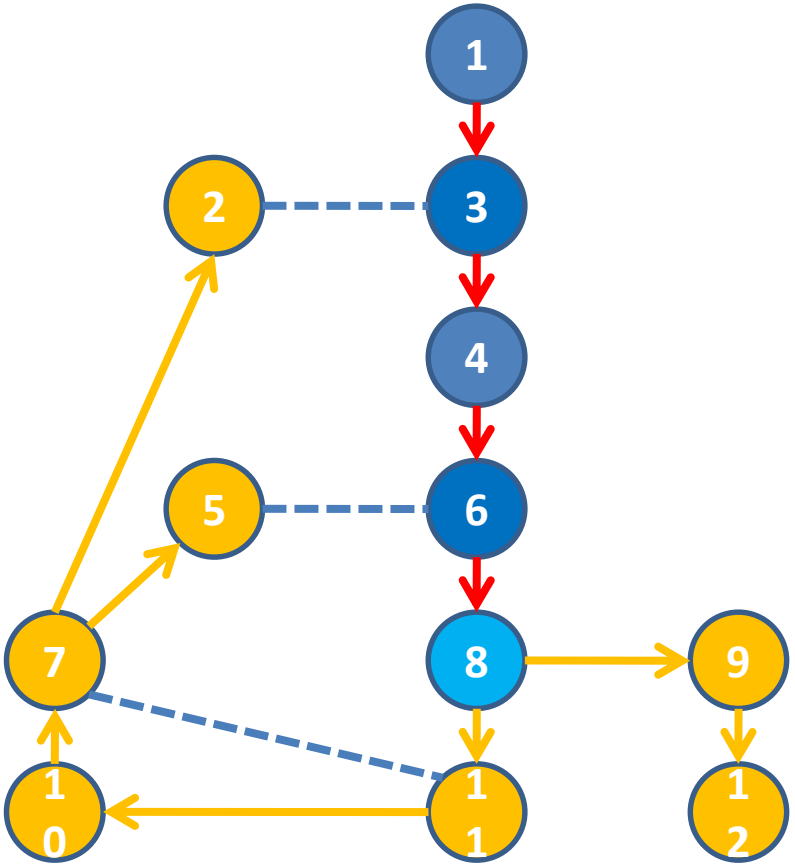
2、Tarjan’s Algorithm算法实现

low[u]取下述结点中时间戳最小的值
(1)、以u为根的dfs树上的所有结点
(2)、与(1)中的结点以一条边相邻的所有结点



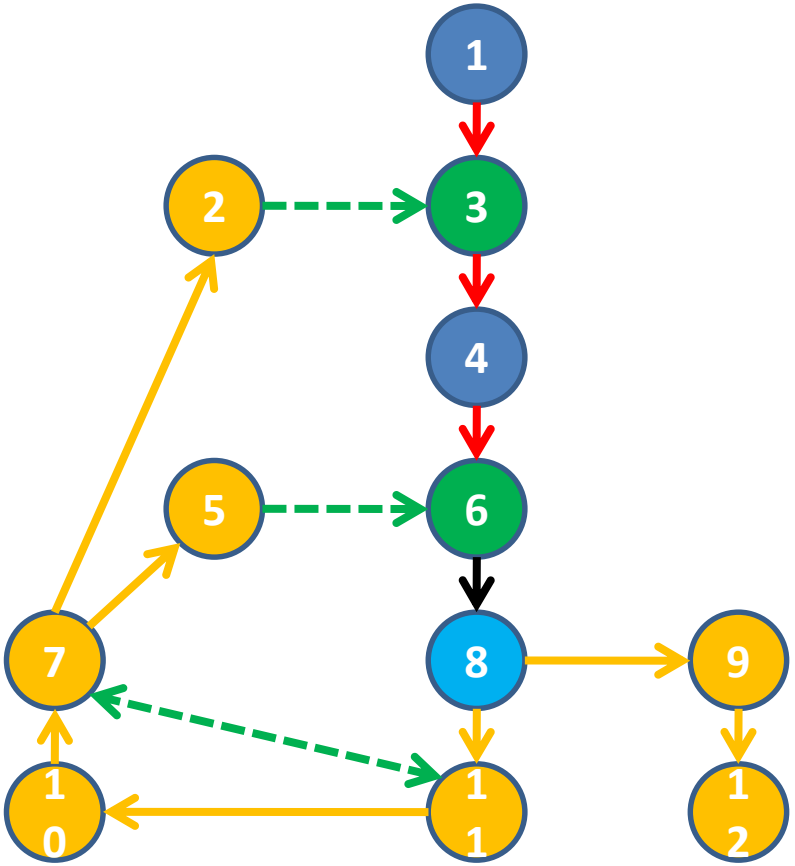
2、Tarjan’s Algorithm算法实现

low[u]取下述结点中时间戳最小的值
(1)、以u为根的dfs树上的所有结点
(2)、与(1)中的结点以一条边相邻的所有结点



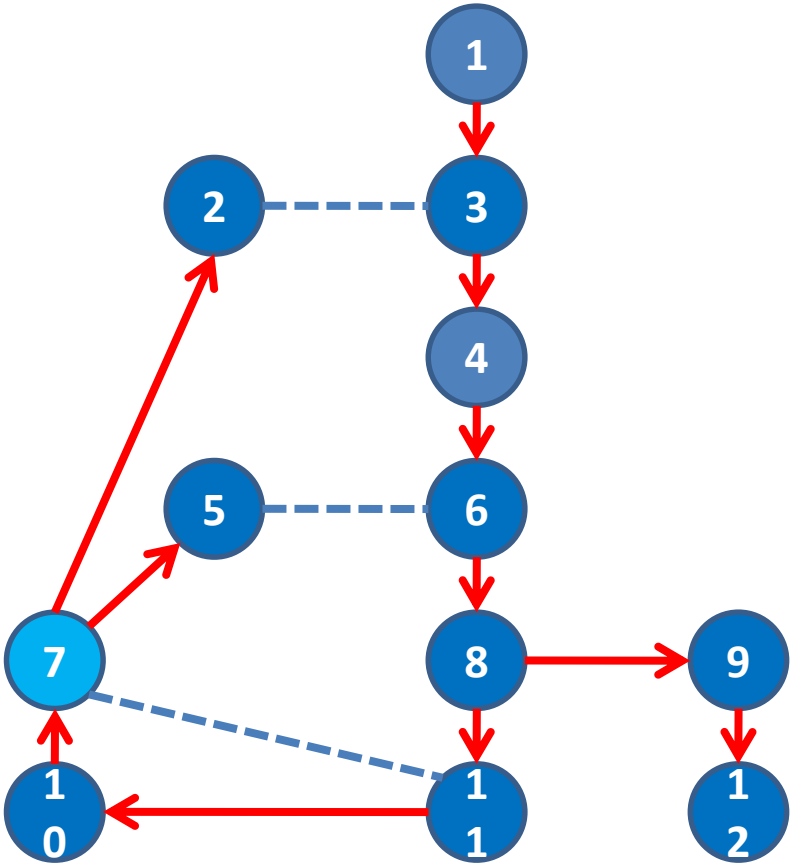
2、Tarjan’s Algorithm算法实现

low[u]取下述结点中时间戳最小的值
(1)、以u为根的dfs树上的所有结点
(2)、与(1)中的结点以一条边相邻的所有结点



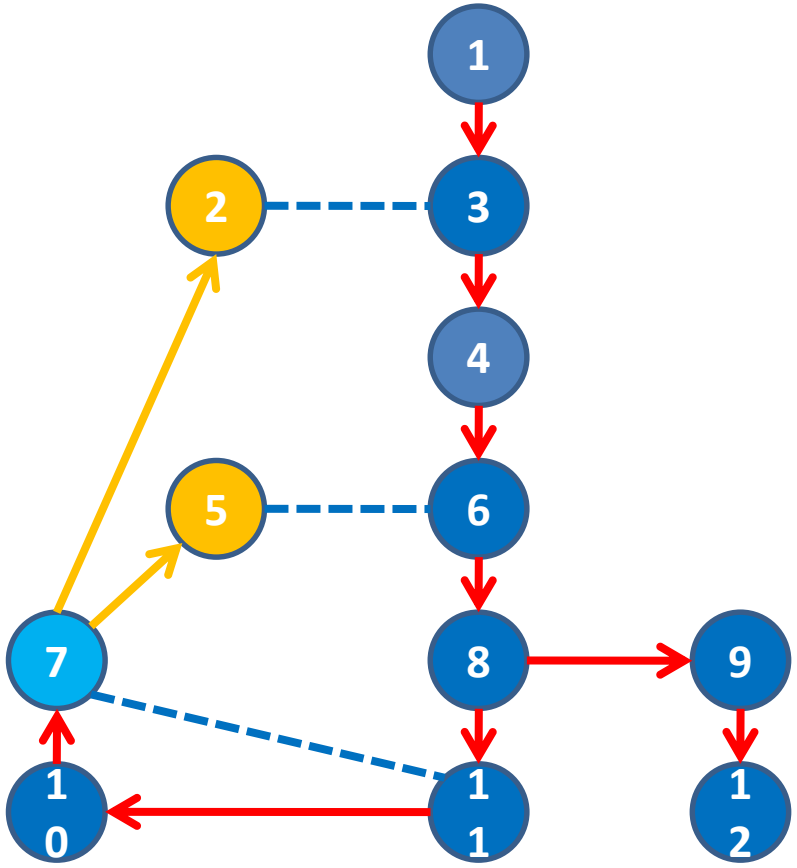
2、Tarjan’s Algorithm算法实现

low[u]取下述结点中时间戳最小的值
(1)、以u为根的dfs树上的所有结点
(2)、与(1)中的结点以一条边相邻的所有结点



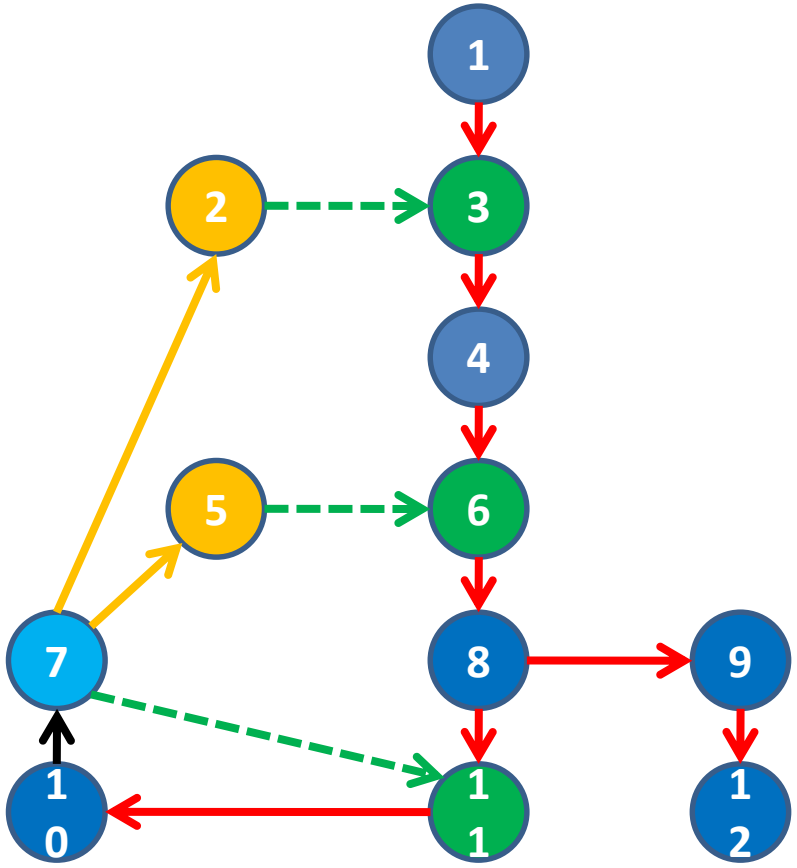
2、Tarjan’s Algorithm算法实现

low[u]取下述结点中时间戳最小的值
(1)、以u为根的dfs树上的所有结点
(2)、与(1)中的结点以一条边相邻的所有结点



2、Tarjan’s Algorithm算法实现

low[u]取下述结点中时间戳最小的值
(1)、以u为根的dfs树上的所有结点
(2)、与(1)中的结点以一条边相邻的所有结点



2、Tarjan’s Algorithm算法实现

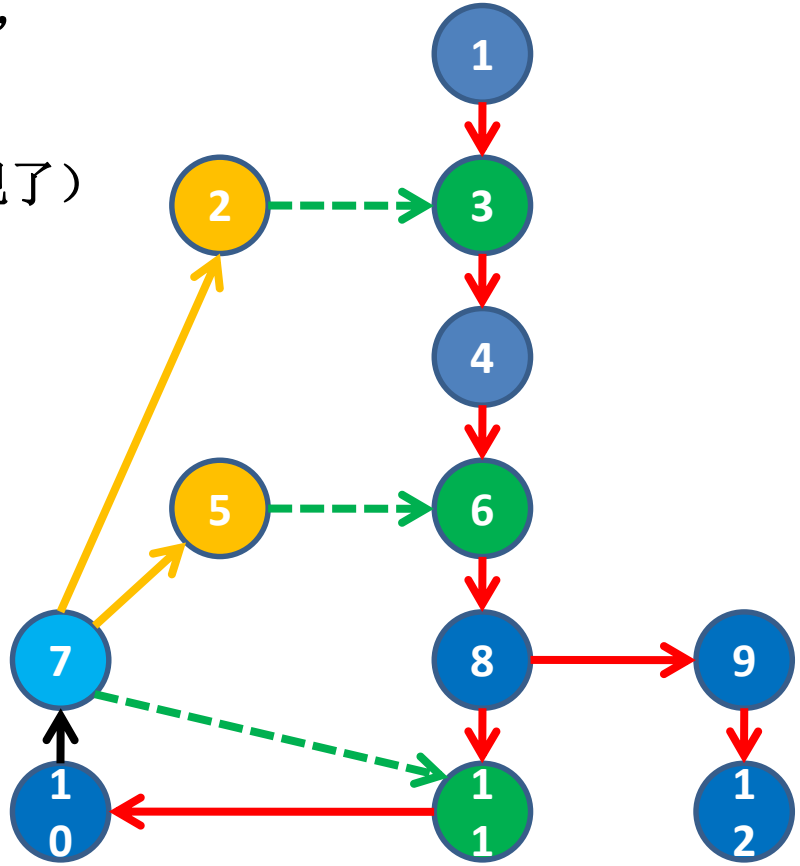
维护dfn[u]：
在dfs的过程中，
当搜索到一个新的结点u时，令dfn[u]=++time。

维护low[u]：
当搜索到一个新的结点u时，令low[u]=dfn[u]
在dfs的过程中，遍历所有以u为端点的边 (u,v)时，
如果是dfs树边，则low[u]=min(low[u],low[v])
如果不是dfs树边，则low[u]=min(low[u], dfn[v])
（如果v是u的父结点则跳过，这在dfs中已经实现了）

2、Tarjan’s Algorithm算法实现

维护low[u]：
当搜索到一个新的结点u时，令low[u]=dfn[u]
在dfs的过程中，遍历所有以u为端点的边 (u,v)时，
如果是dfs树边，则low[u]=min(low[u],low[v])
如果不是dfs树边，则low[u]=min(low[u], dfn[v])
（如果v是u的父结点则跳过，这在dfs中已经实现了）

low[2]=min{dfn[2],dfn[3]}
low[5]=min{dfn[5],dfn[6]}
low[7]=min{low[2],low[5],dfn[11],dfn[7]}



2、Tarjan’s Algorithm算法实现

割点判断：

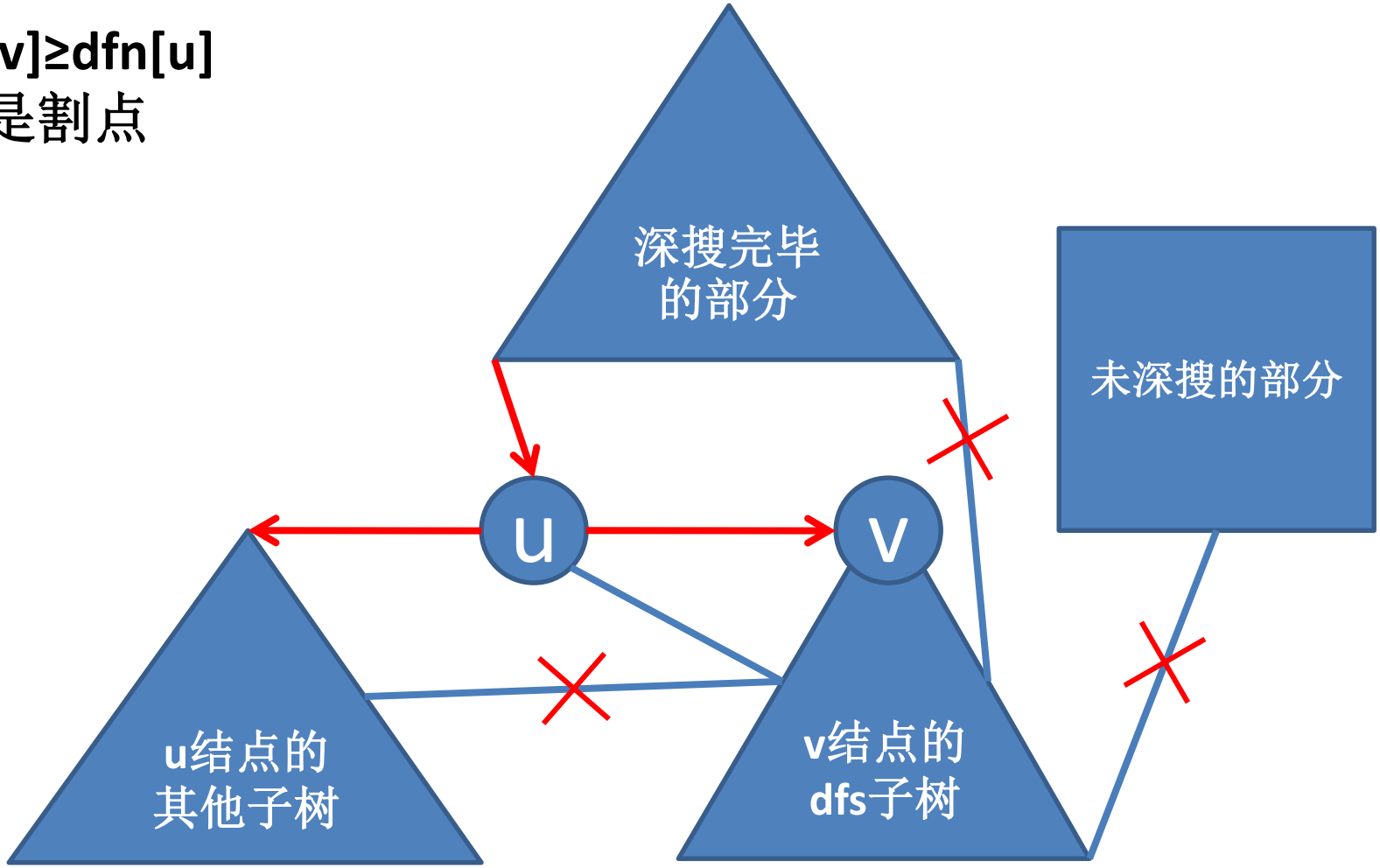
如果在dfs树里，u的所有子结点中，存在某个结点v，
有low[v]≥dfn[u]，
则，以v为根结点的dfs树中的所有结点，不能跳过u与其他结点连通。可知u是一个割点。

对于u的其他子树，显然v树与这些子树上的结点只能通过u连通。
对于u的祖先结点和这些结点的其他子树上的结点x。
case1:dfn[x]>dfn[u]
则因为low[v]≥dfn[u]，故v树不与比v先搜索到的结点连通，故v不与x连通。
case2:dfn[x]<dfn[u]
结点x后于v搜索到，根据dfs的特性，x与v显然不能跳过u连通。

2、Tarjan’s Algorithm算法实现

割点判断：

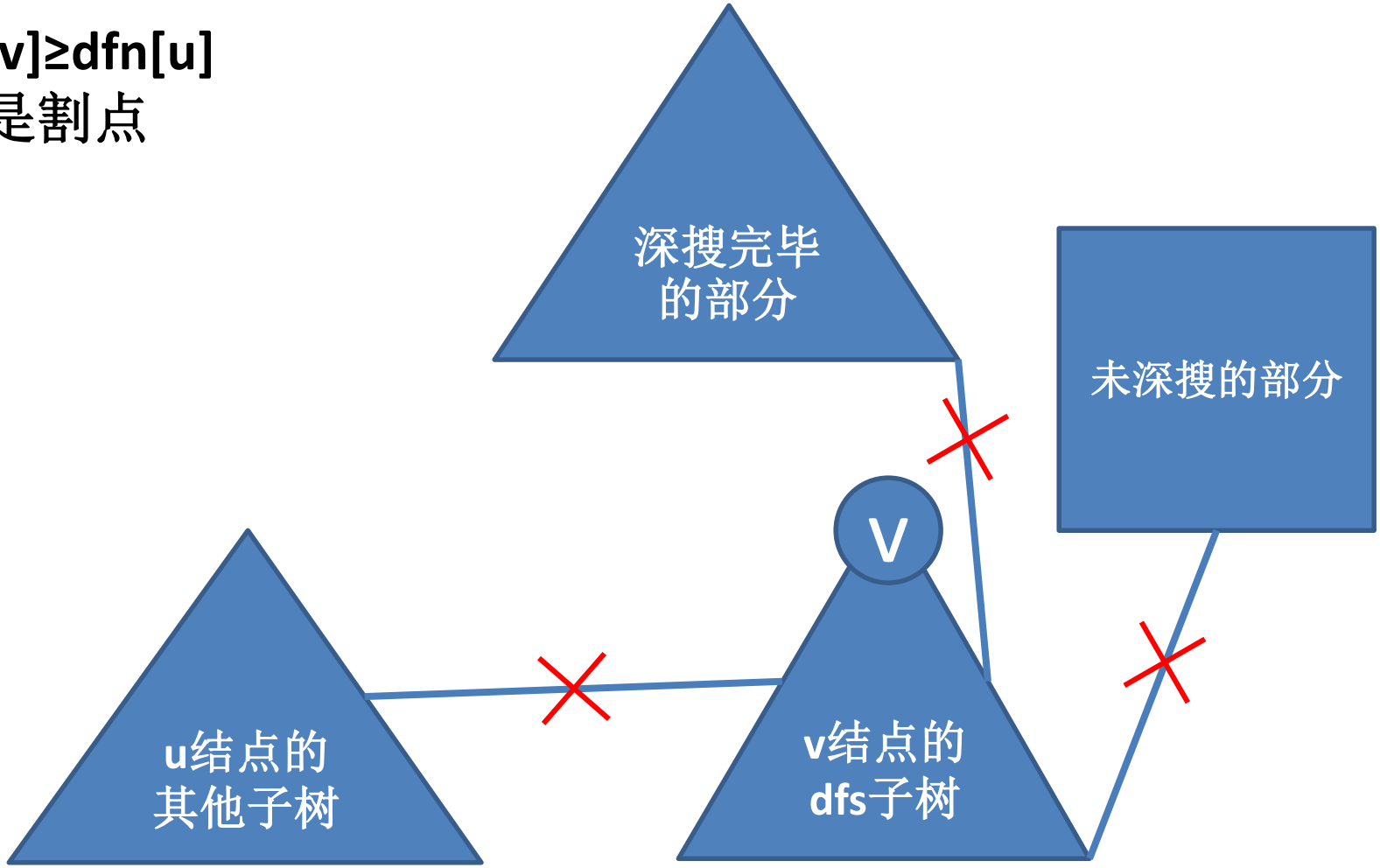
low[v]≥dfn[u]
则u是割点



2、Tarjan’s Algorithm算法实现

割点判断：

low[v]≥dfn[u]
则u是割点



2、Tarjan’s Algorithm算法实现

割点判断：

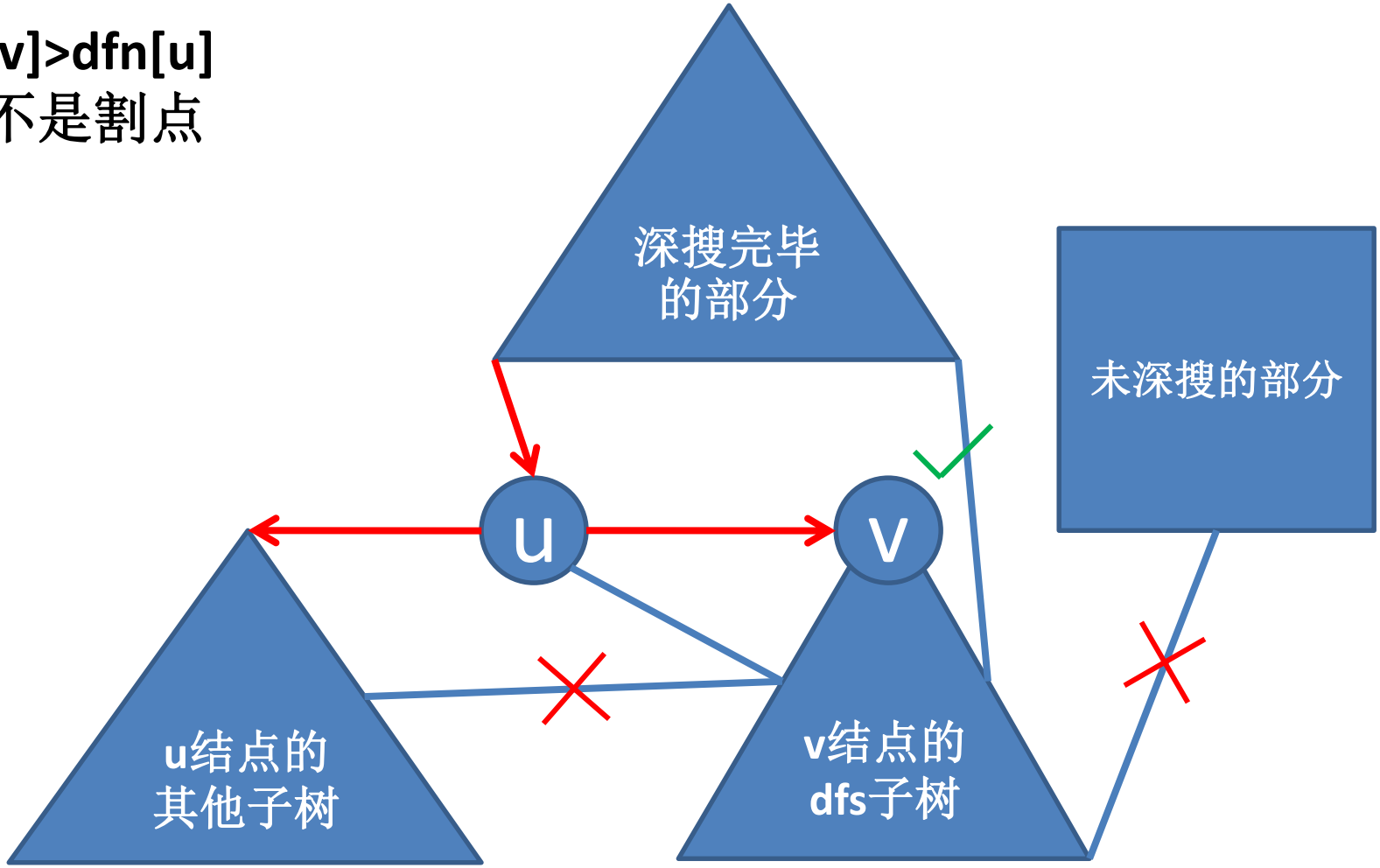
如果在dfs树里，u的所有子结点中，对于所有的结点v，
有low[v]<dfn[u]，
也就意味着v树上的结点可以跳过u与u的祖先结点或祖先结点的子树连通。

则删除u，u的所有子树依旧与u的父结点连通，整个图依旧连通，因此u不是割点。

2、Tarjan’s Algorithm算法实现

割点判断：

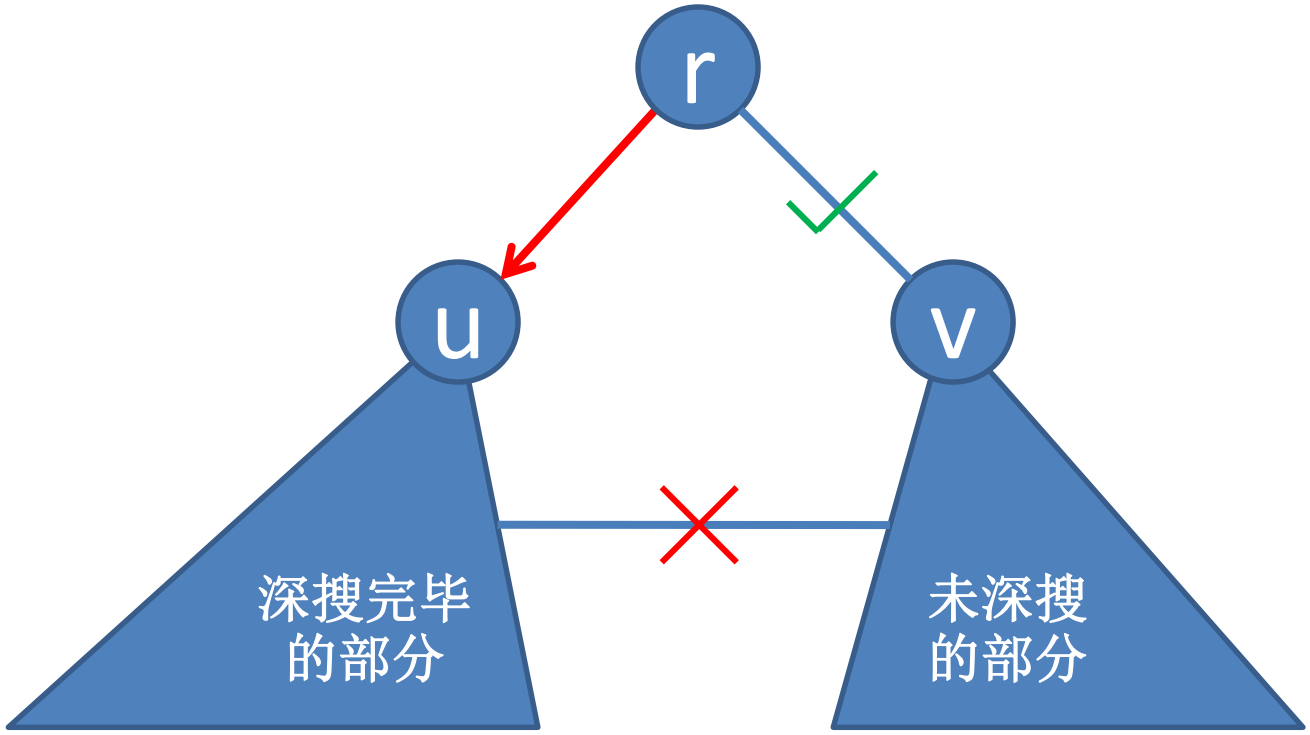
low[v]>dfn[u]
则u不是割点



2、Tarjan’s Algorithm算法实现

割点判断：

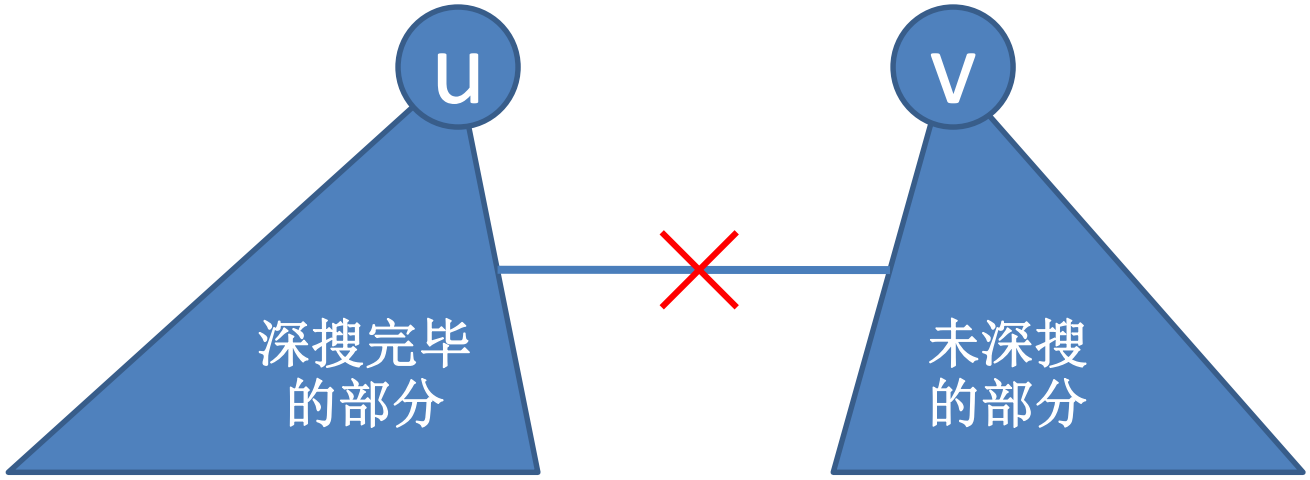
对根节点特判



2、Tarjan’s Algorithm算法实现

割点判断：

对根节点特判



2、Tarjan’s Algorithm算法实现

割点判断：

因此，判断一个点u是不是割点的方式是：

对于结点u，存在一个子结点v，有low[v]≥dfn[u]，
则u是割点。反之不是。

特别地，对于dfs树的根节点r，判断其是否是割点的方式是：

如果r有多于1个的子树，则r是割点，反之不是。

2、Tarjan’s Algorithm算法实现

割边判断：

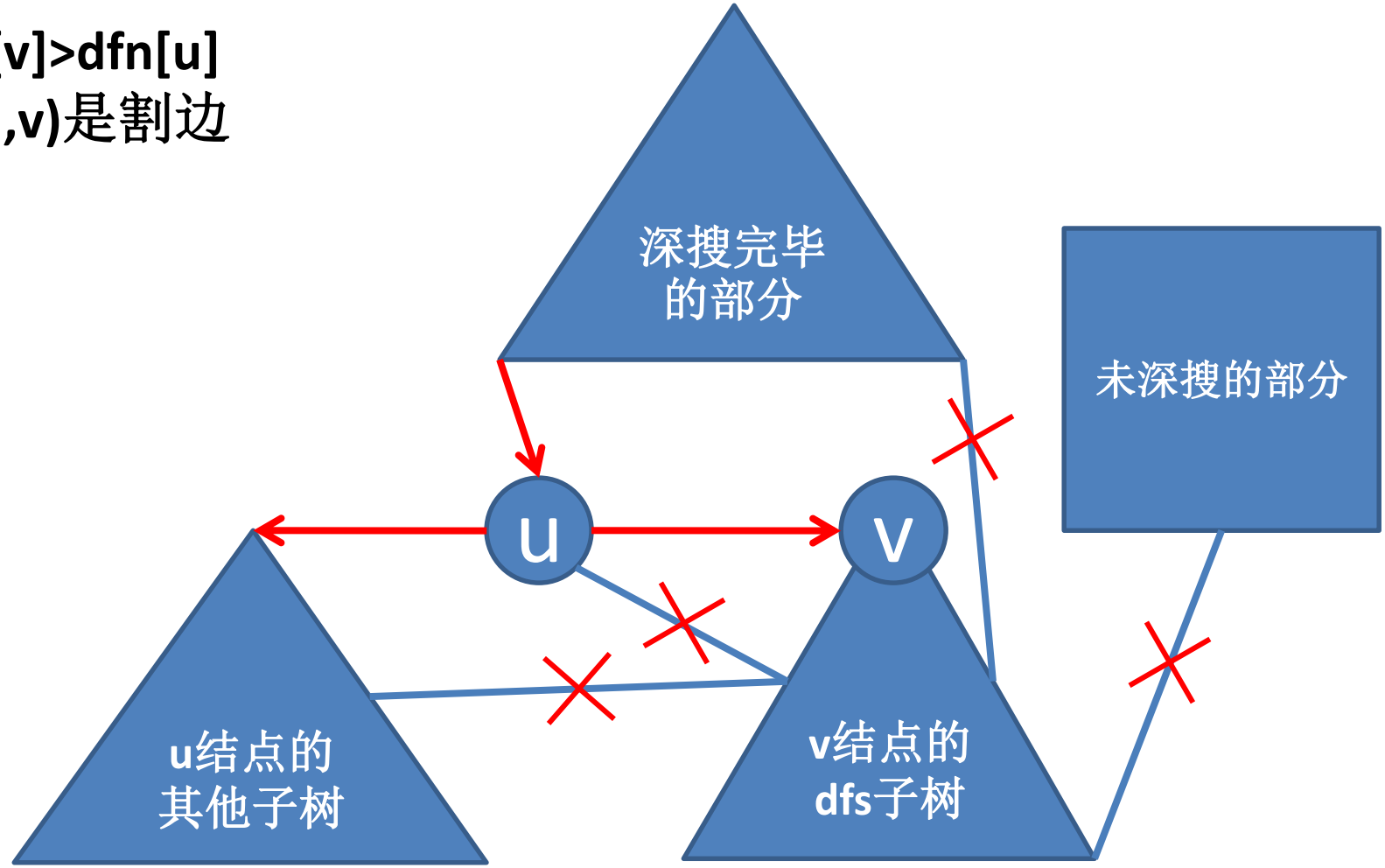
如果在dfs树里，对于边(u,v)（u是父结点），
有low[v]>dfn[u]，
则v树不能跳过(u,v)与其他结点连通。

对于u的其他子树，显然v树与这些子树上的结点只能通过u连通。
对于u以及u的祖先结点和这些结点的其他子树上的结点x。
case1:dfn[x]>dfn[u]
则因为low[v]>dfn[u]，故v树不与比v先搜索到的结点连通，故v不与x连通。
case2:dfn[x]<dfn[u]
结点x后于v搜索到，根据dfs的特性，x与v显然不能跳过u连通。

2、Tarjan’s Algorithm算法实现

割边判断：

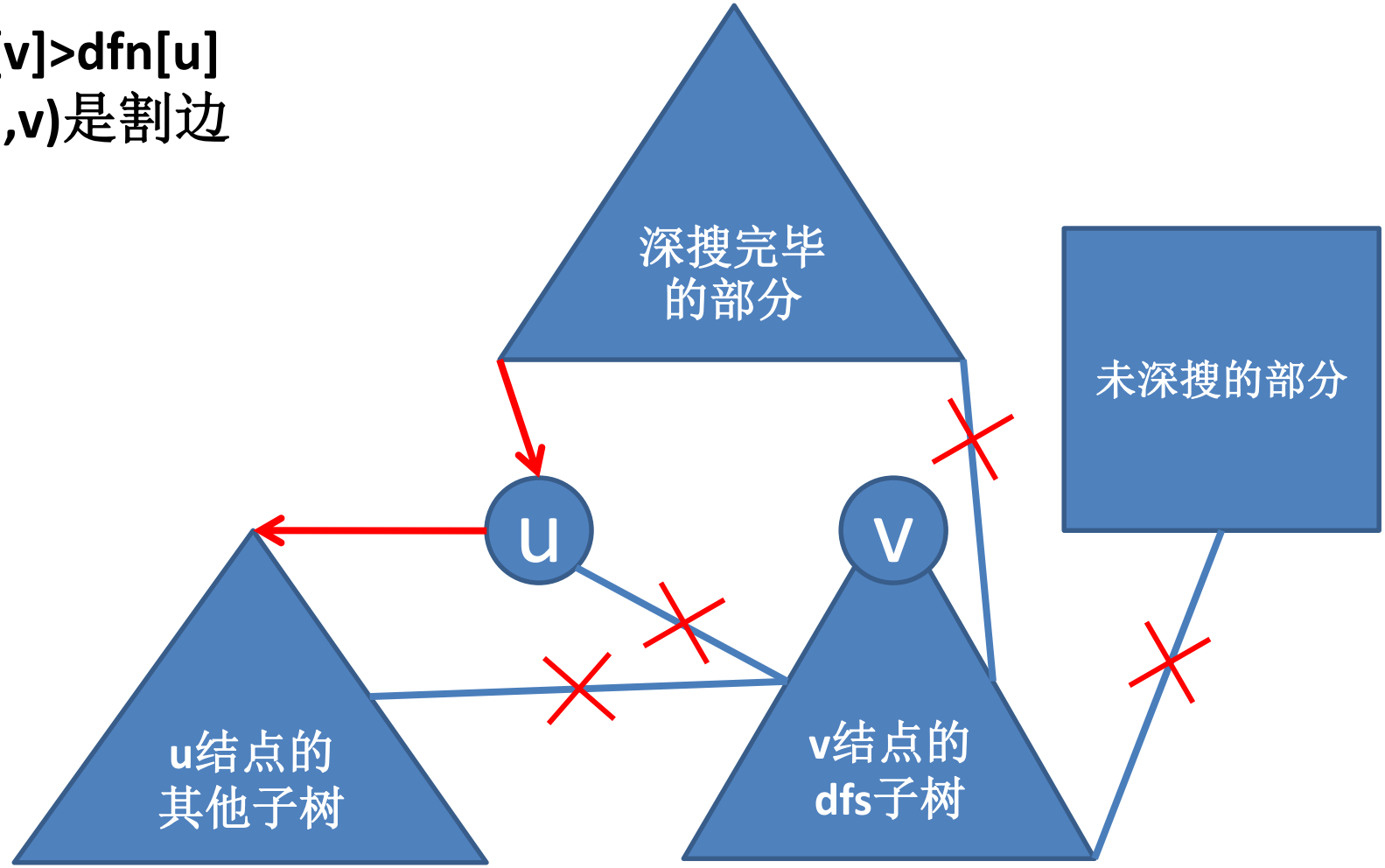
low[v]>dfn[u]
则(u,v)是割边



2、Tarjan’s Algorithm算法实现

割边判断：

low[v]>dfn[u]
则(u,v)是割边



2、Tarjan’s Algorithm算法实现

割边判断：

如果在dfs树里，对于边(u,v)（u是父结点），
有low[v]≤dfn[u]，
也就意味着v树上的结点可以跳过(u,v)与u或u的祖先结点或祖先结点的子树连通。

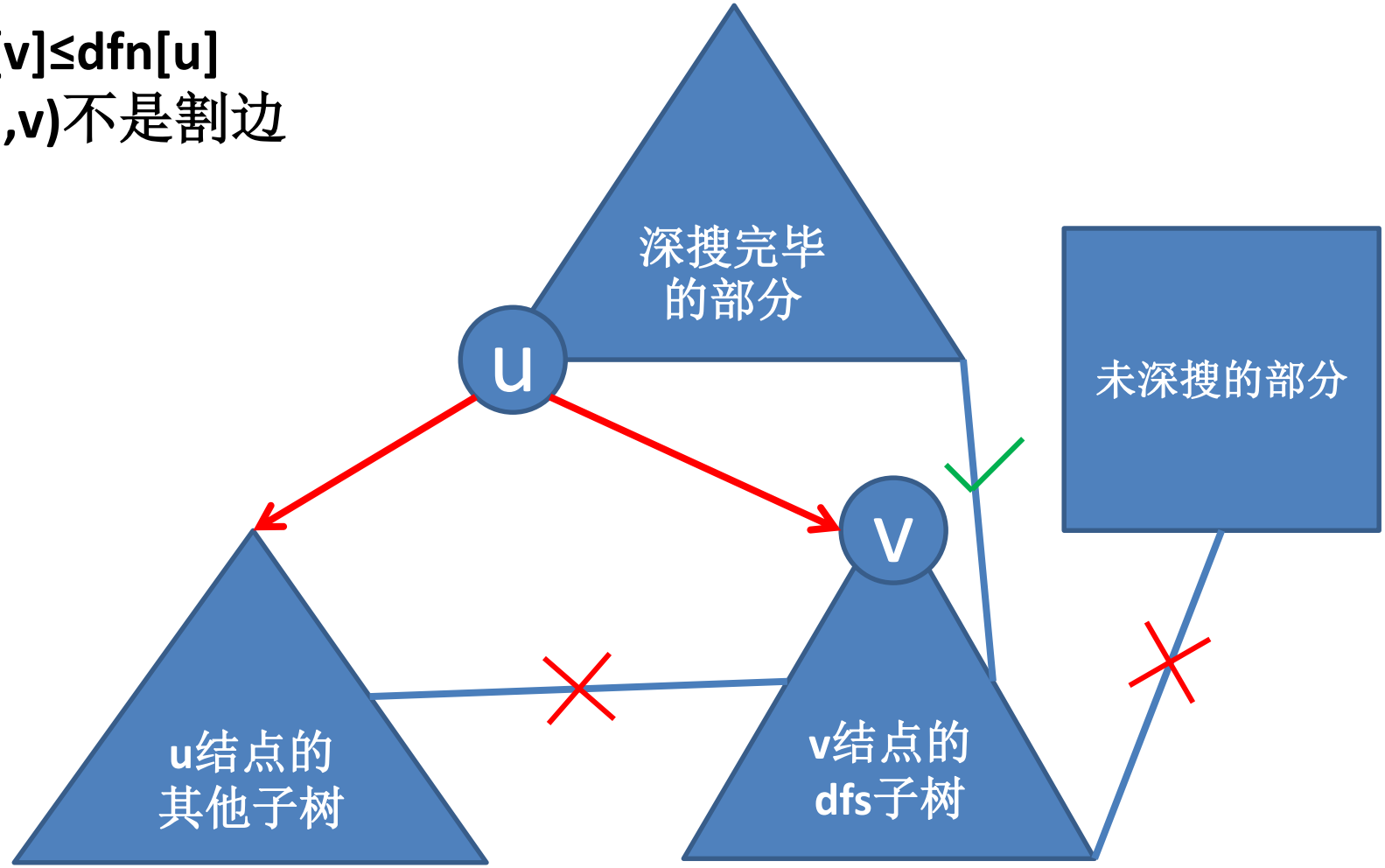
则删除(u,v)，u依旧与v连通，(u,v)不是割边。

如果某条边不在dfs树里，那么这条边显然不是割边。

2、Tarjan’s Algorithm算法实现

割边判断：

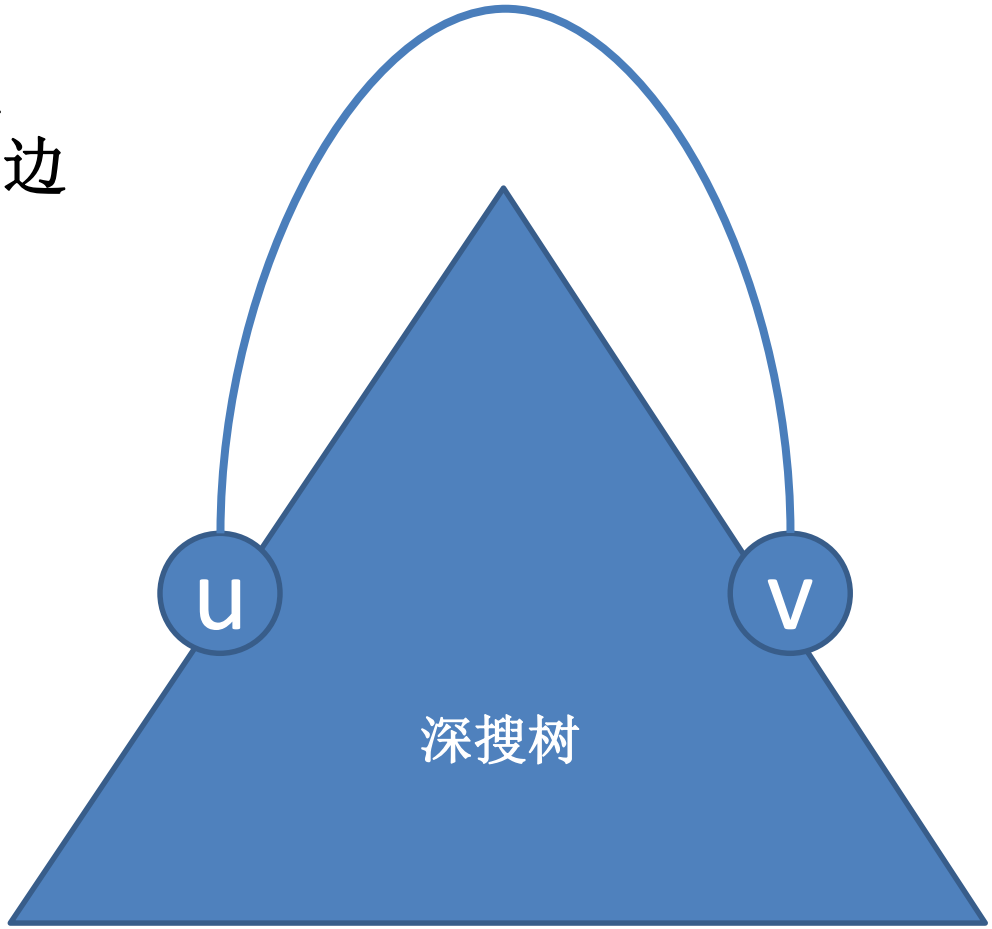
low[v]≤dfn[u]
则(u,v)不是割边



2、Tarjan’s Algorithm算法实现

割边判断：

(u,v)不是树边
则(u,v)不是割边



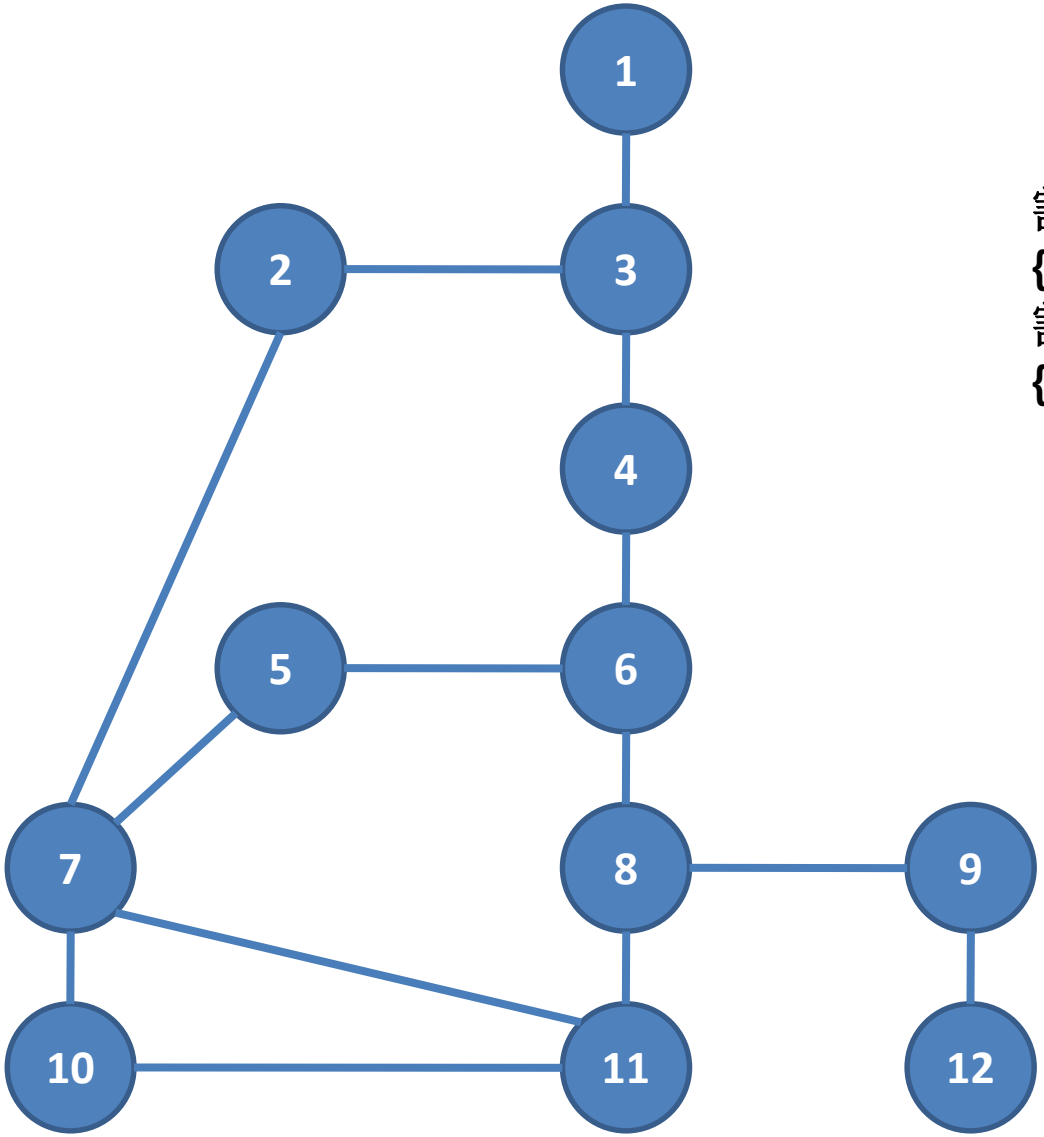
2、Tarjan’s Algorithm算法实现

割边判断：

因此，判断一条边(u,v)是不是割边的方式是：
（u是父结点）

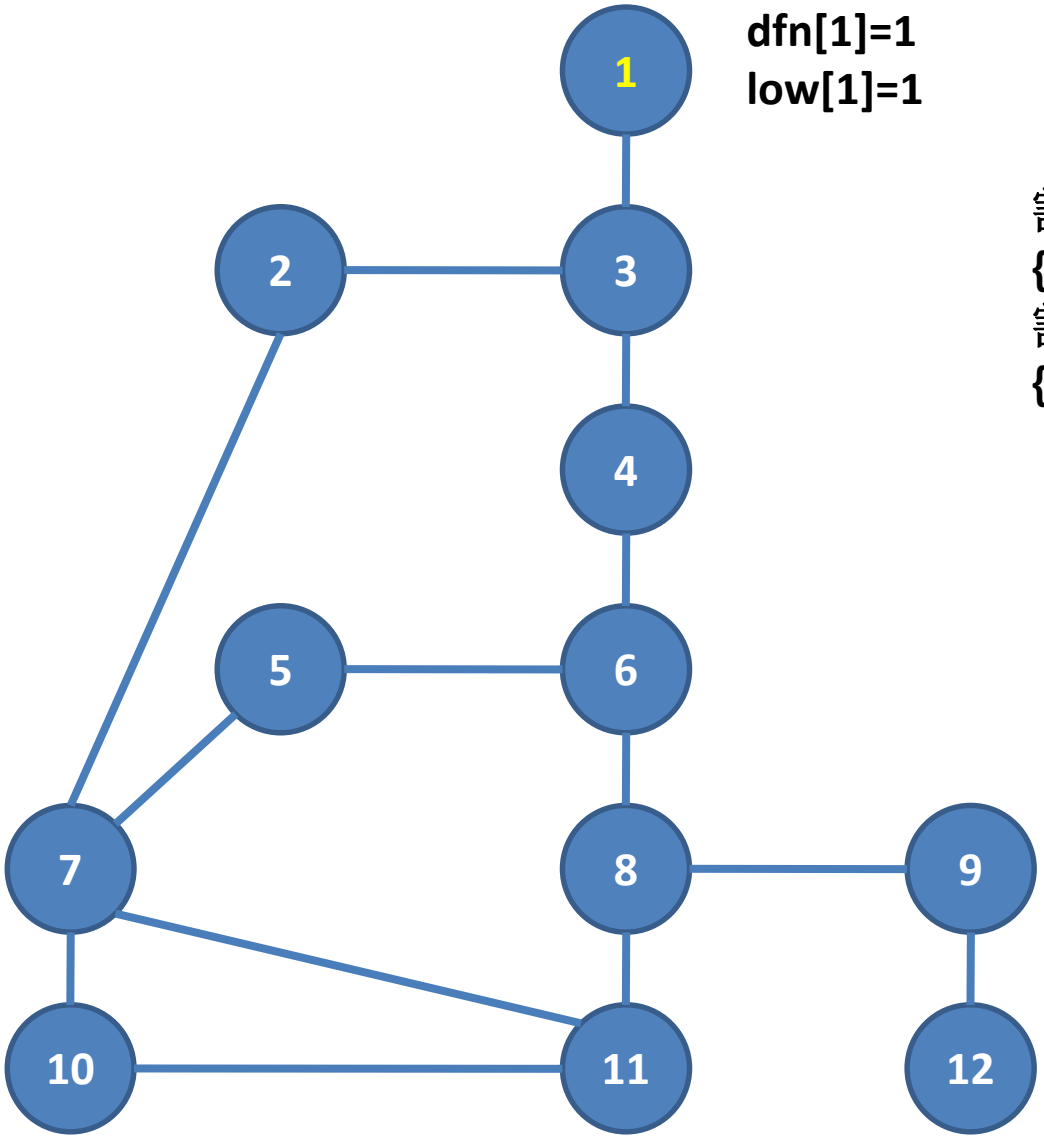
(u,v)是树边，且有low[v]>dfn[u]，则(u,v)是割边。反之不是。

3、Tarjan’s Algorithm算法流程演示



割边集：
{}
割点集：
{}

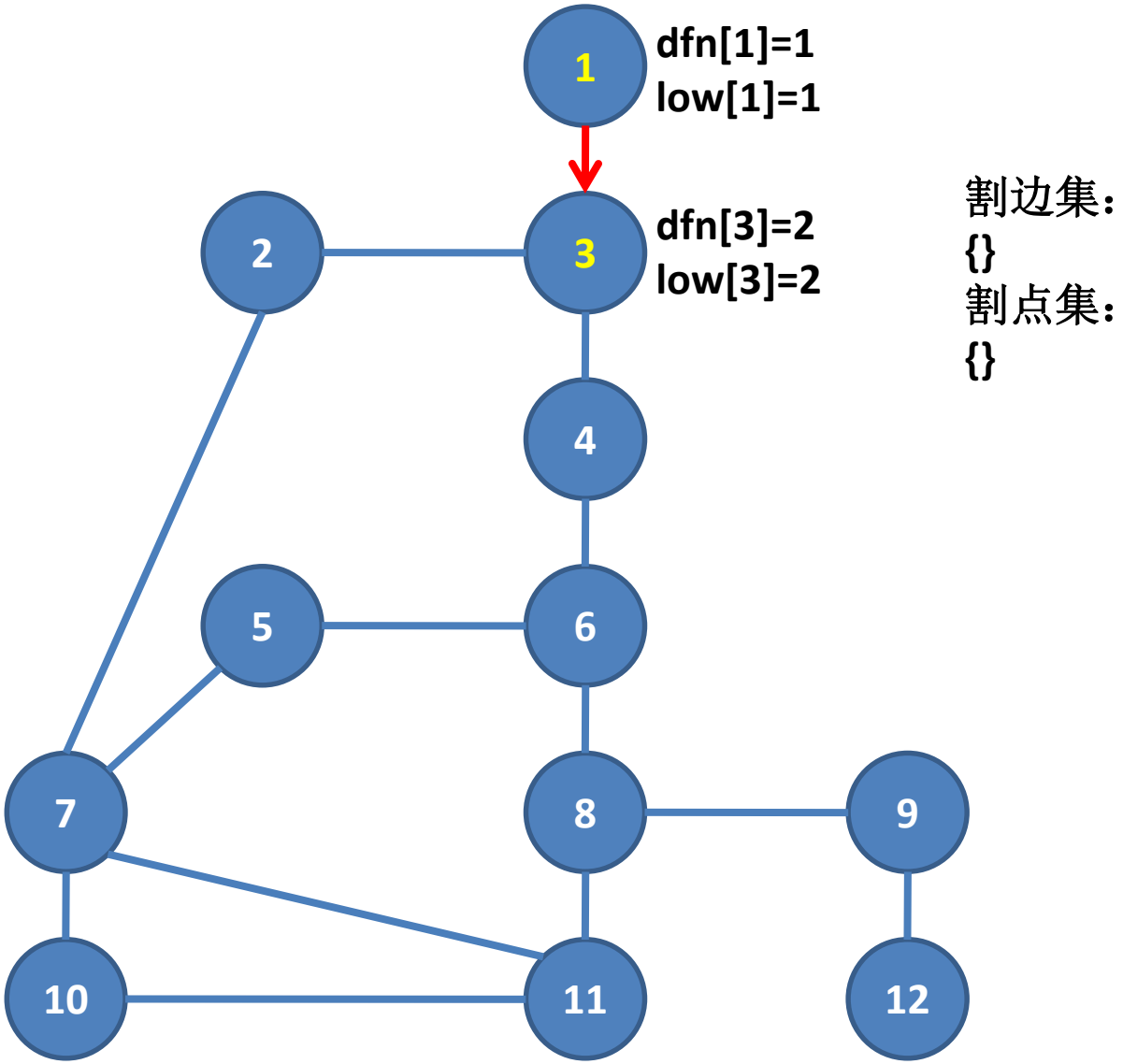
3、Tarjan’s Algorithm算法流程演示



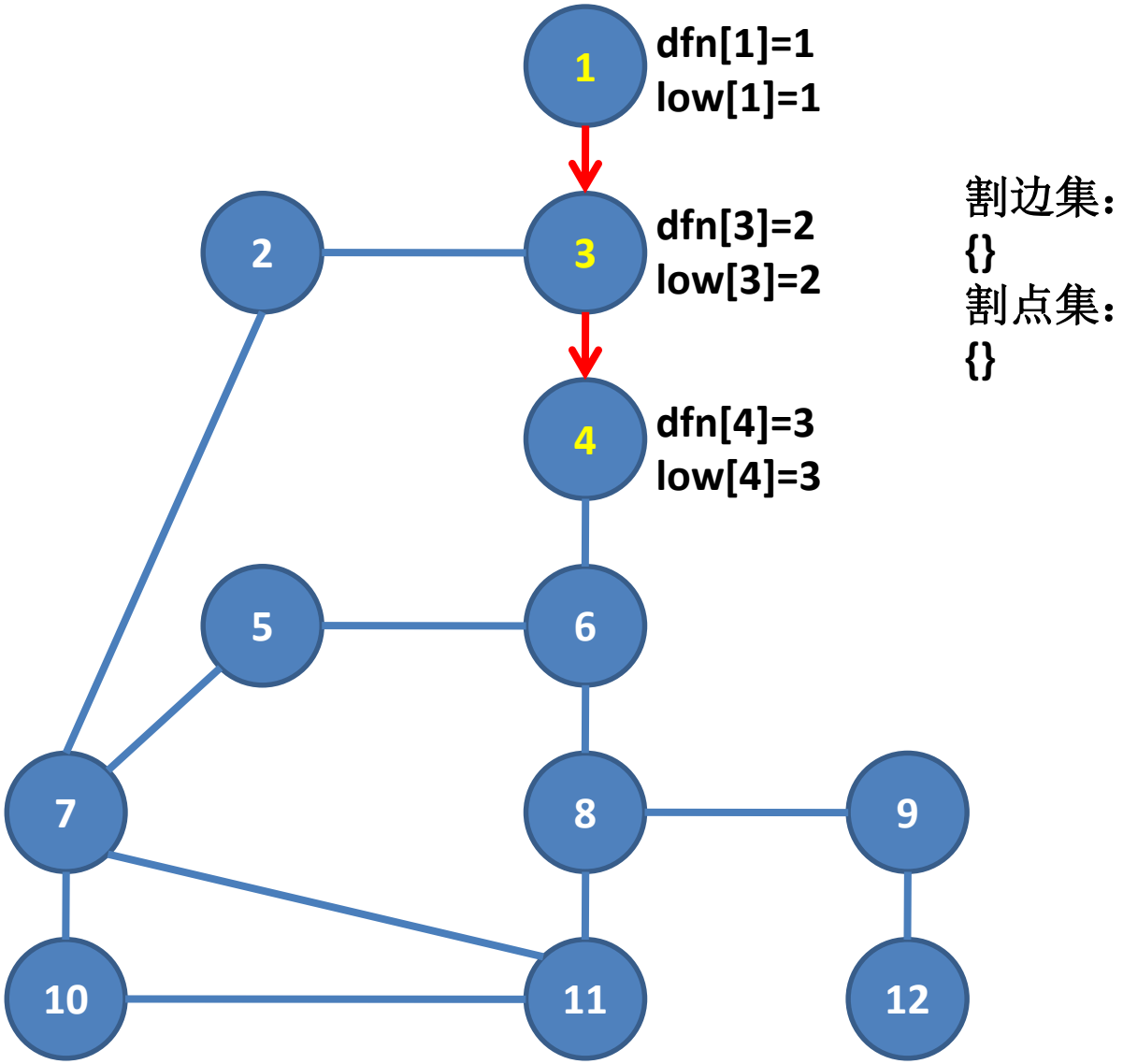
dfn[1]=1
low[1]=1

割边集：
{}
割点集：
{}

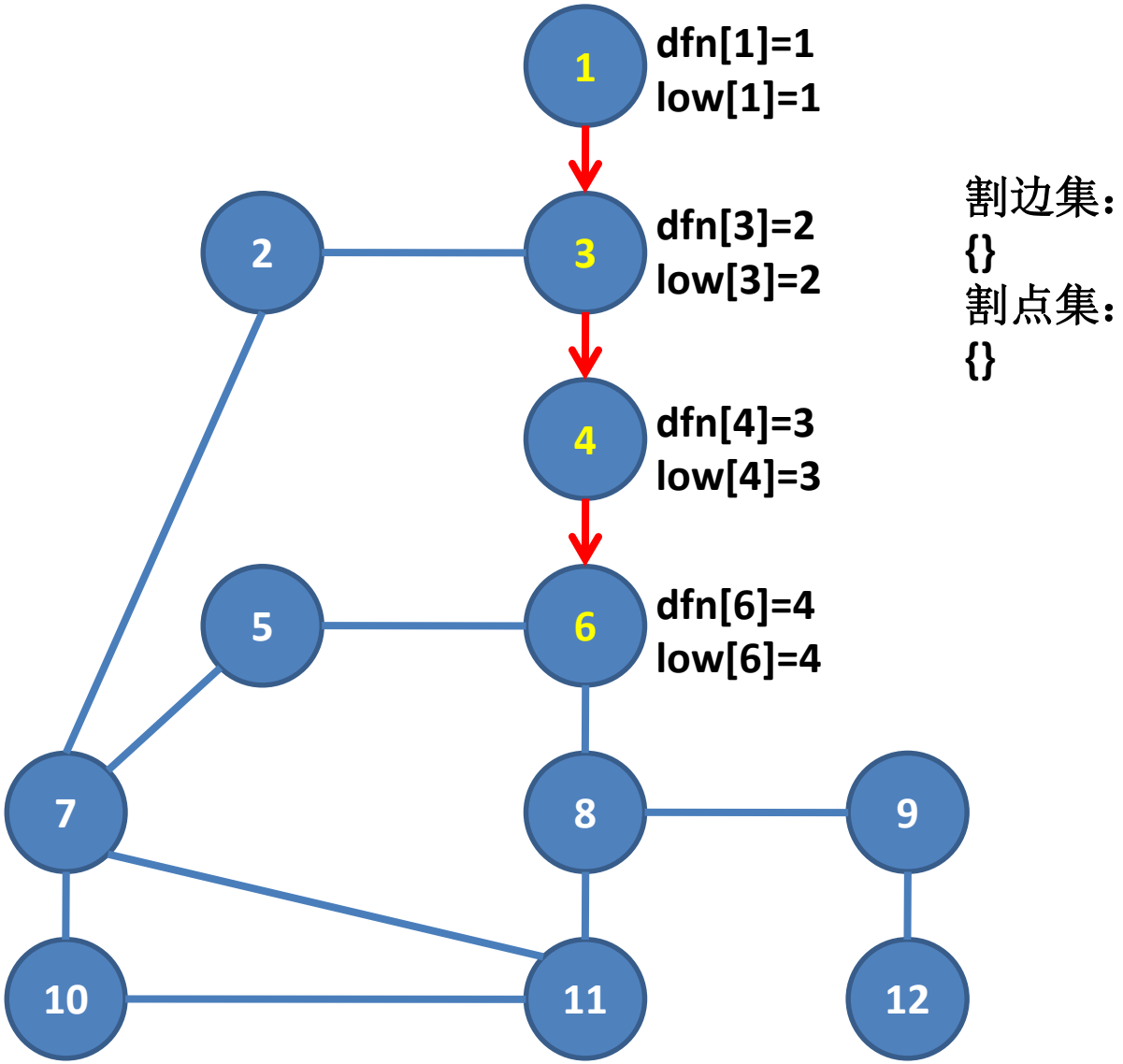
3、Tarjan’s Algorithm算法流程演示



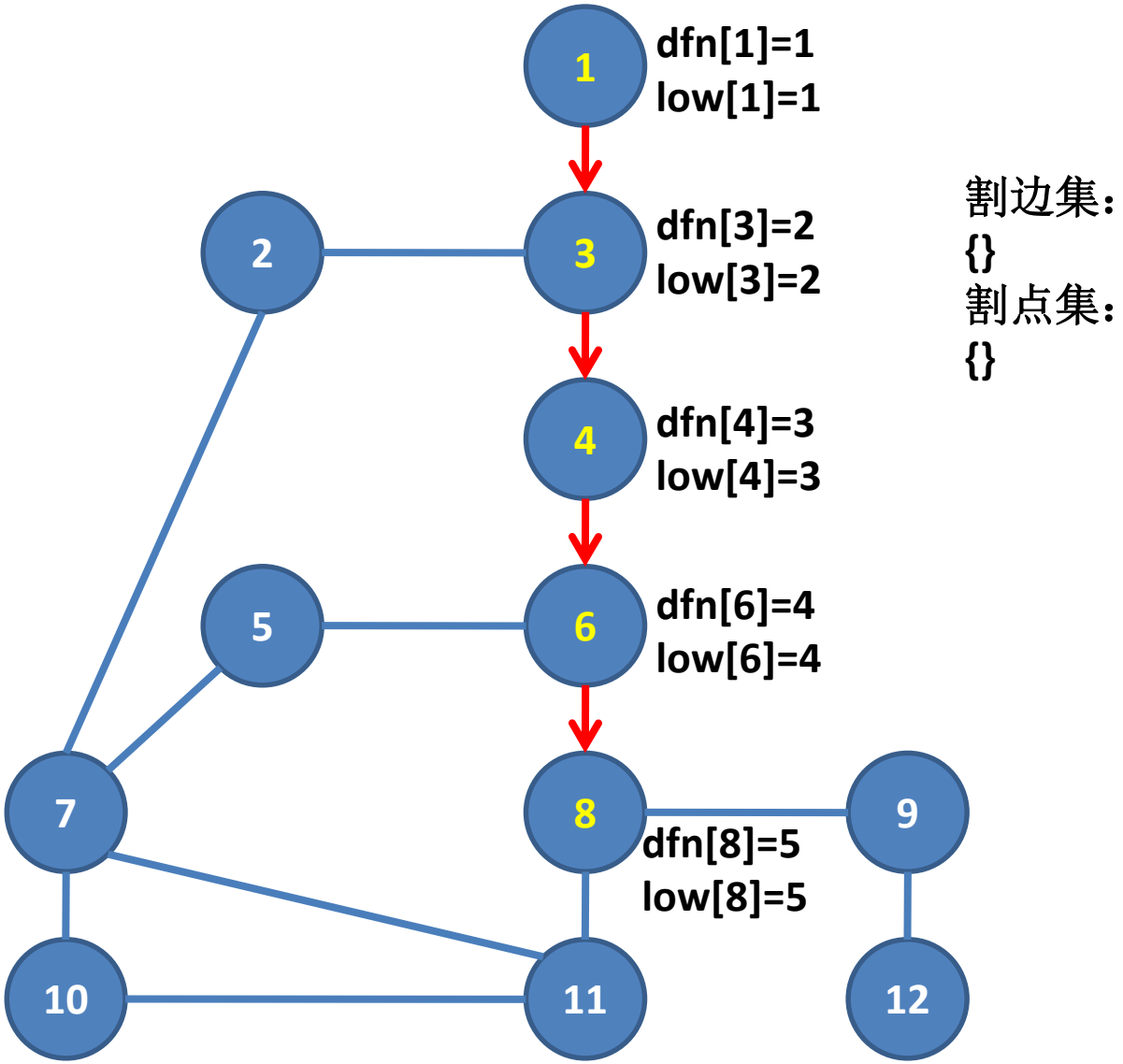
3、Tarjan’s Algorithm算法流程演示



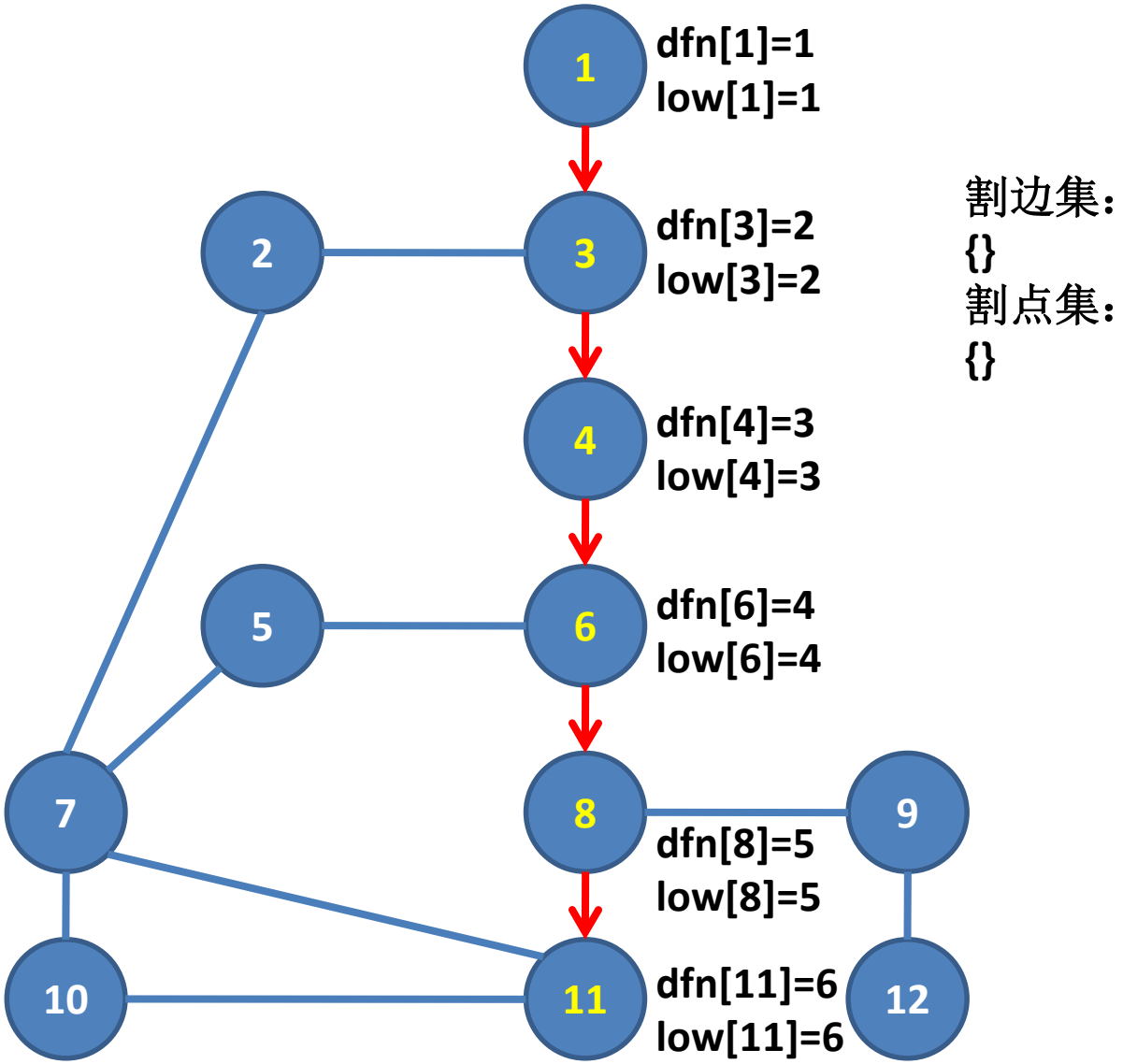
3、Tarjan’s Algorithm算法流程演示



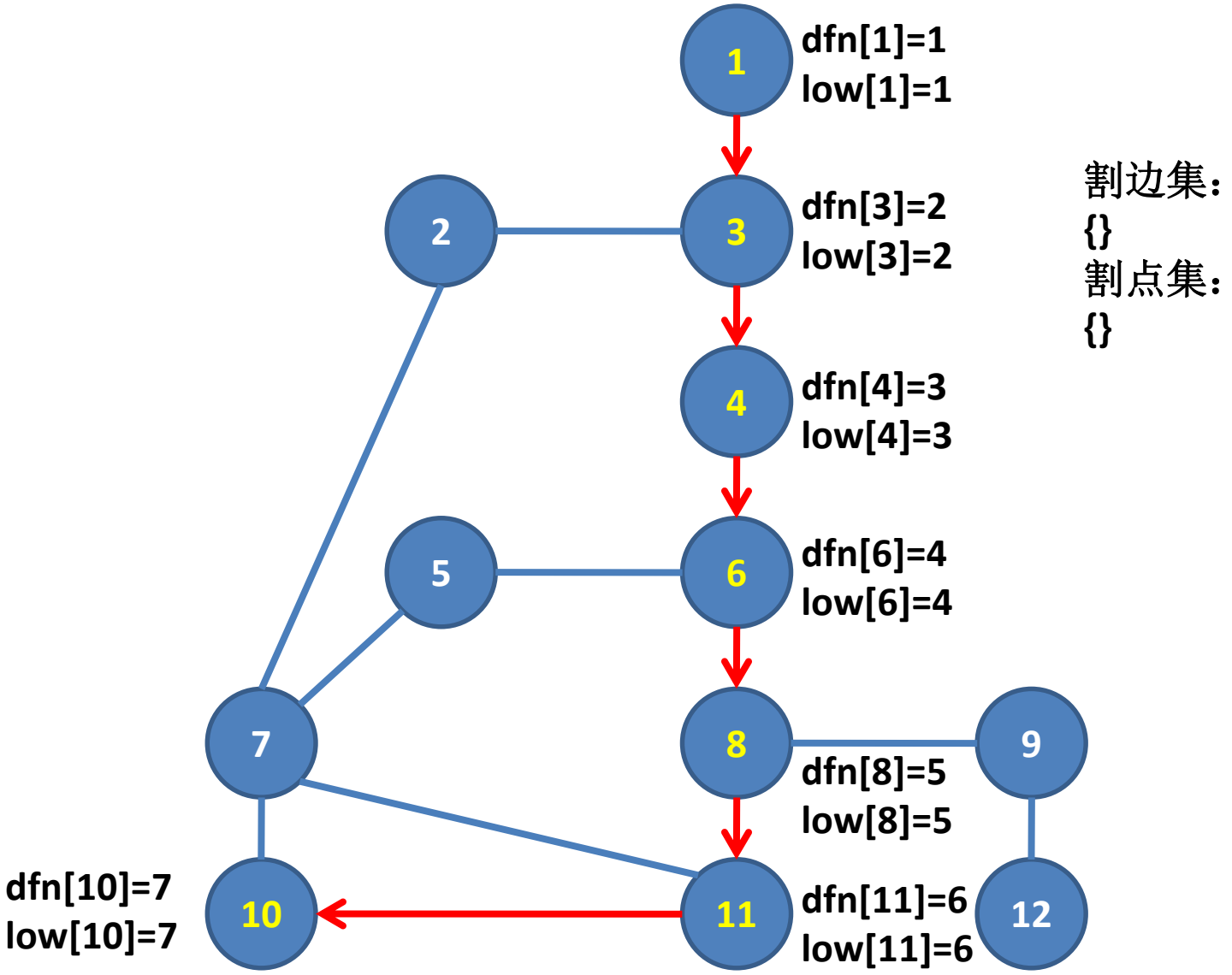
3、Tarjan’s Algorithm算法流程演示



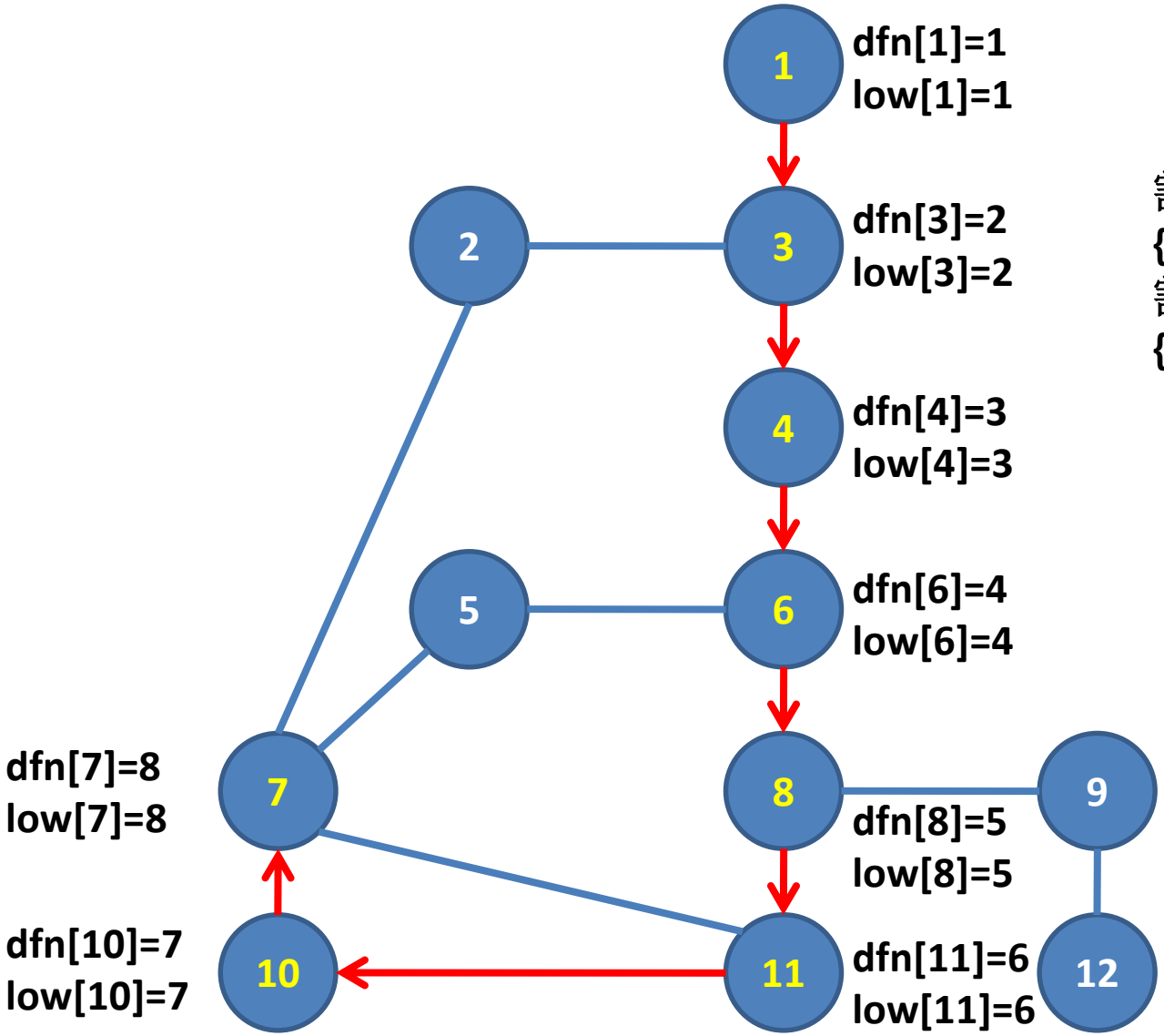
3、Tarjan’s Algorithm算法流程演示



3、Tarjan’s Algorithm算法流程演示

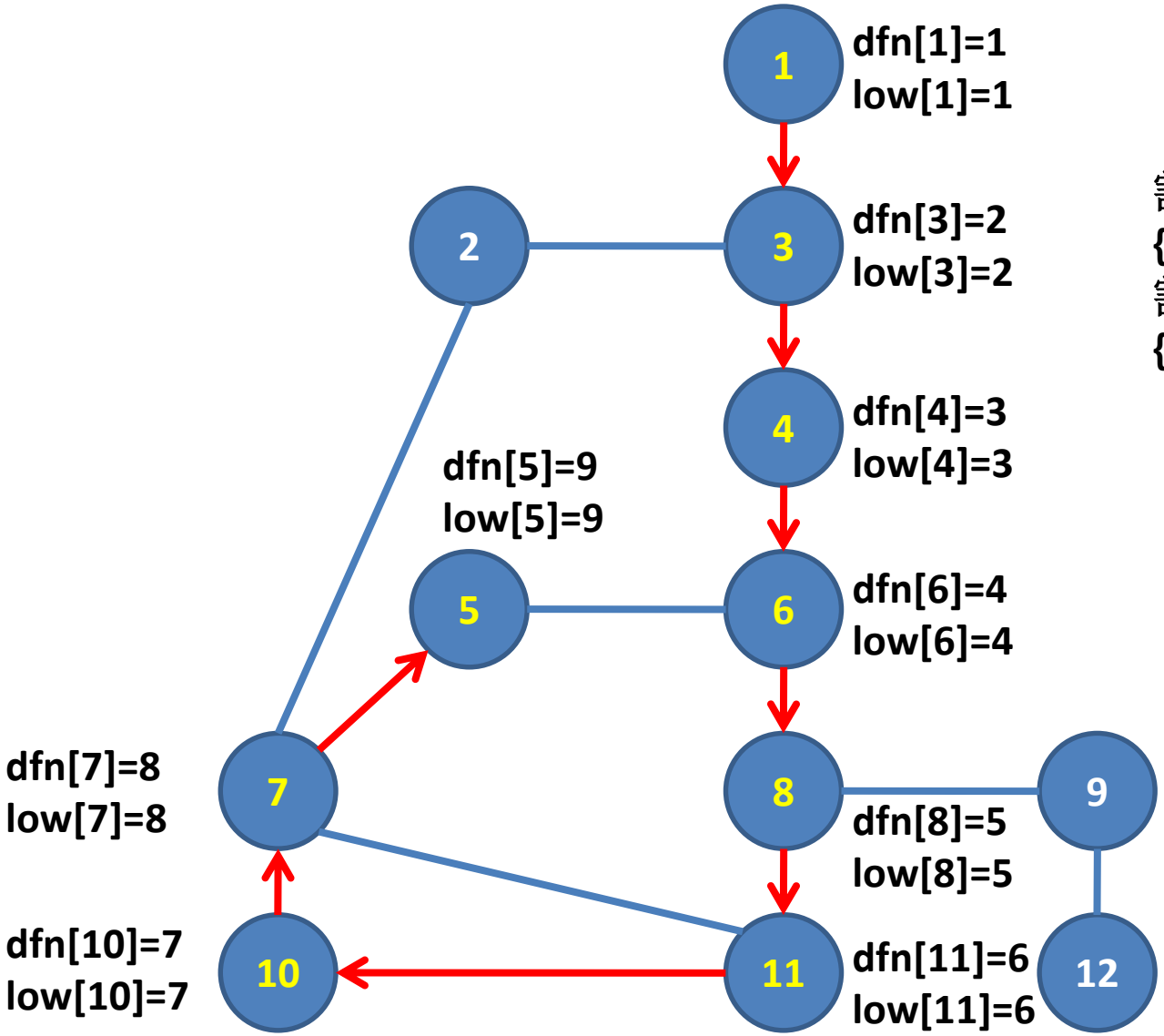


3、Tarjan’s Algorithm算法流程演示

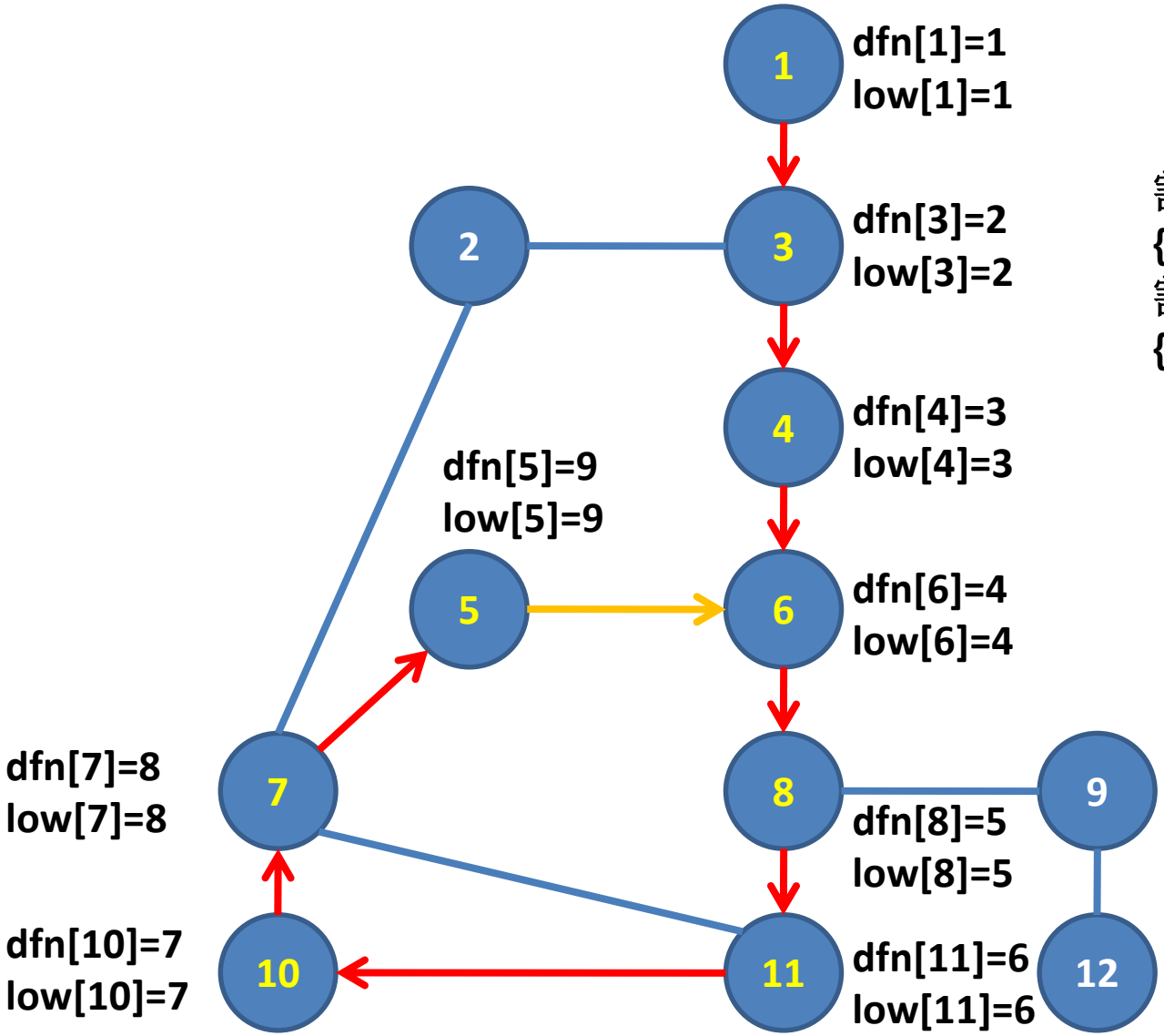


割边集：
{}
割点集：
{}

3、Tarjan’s Algorithm算法流程演示

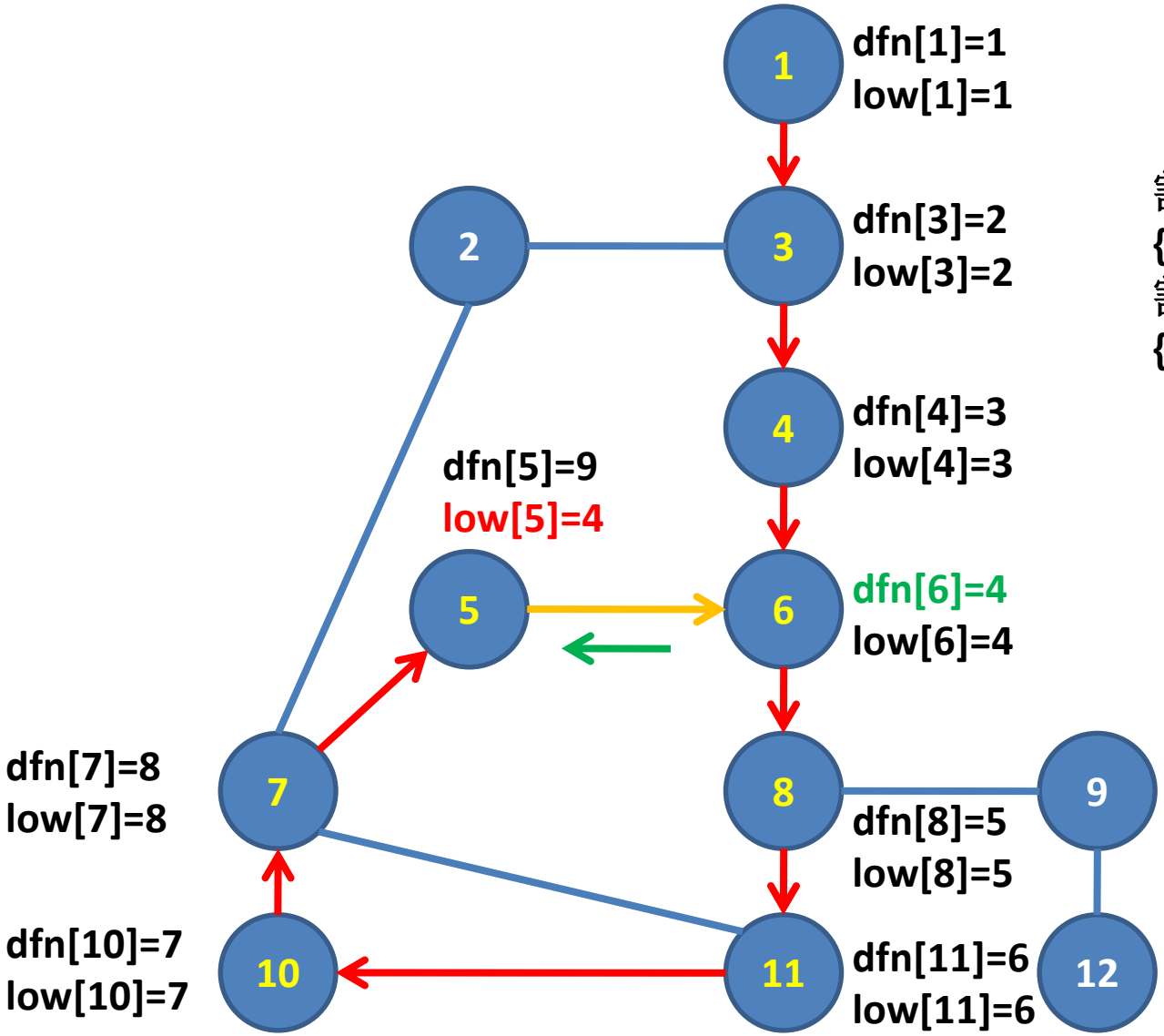


3、Tarjan’s Algorithm算法流程演示

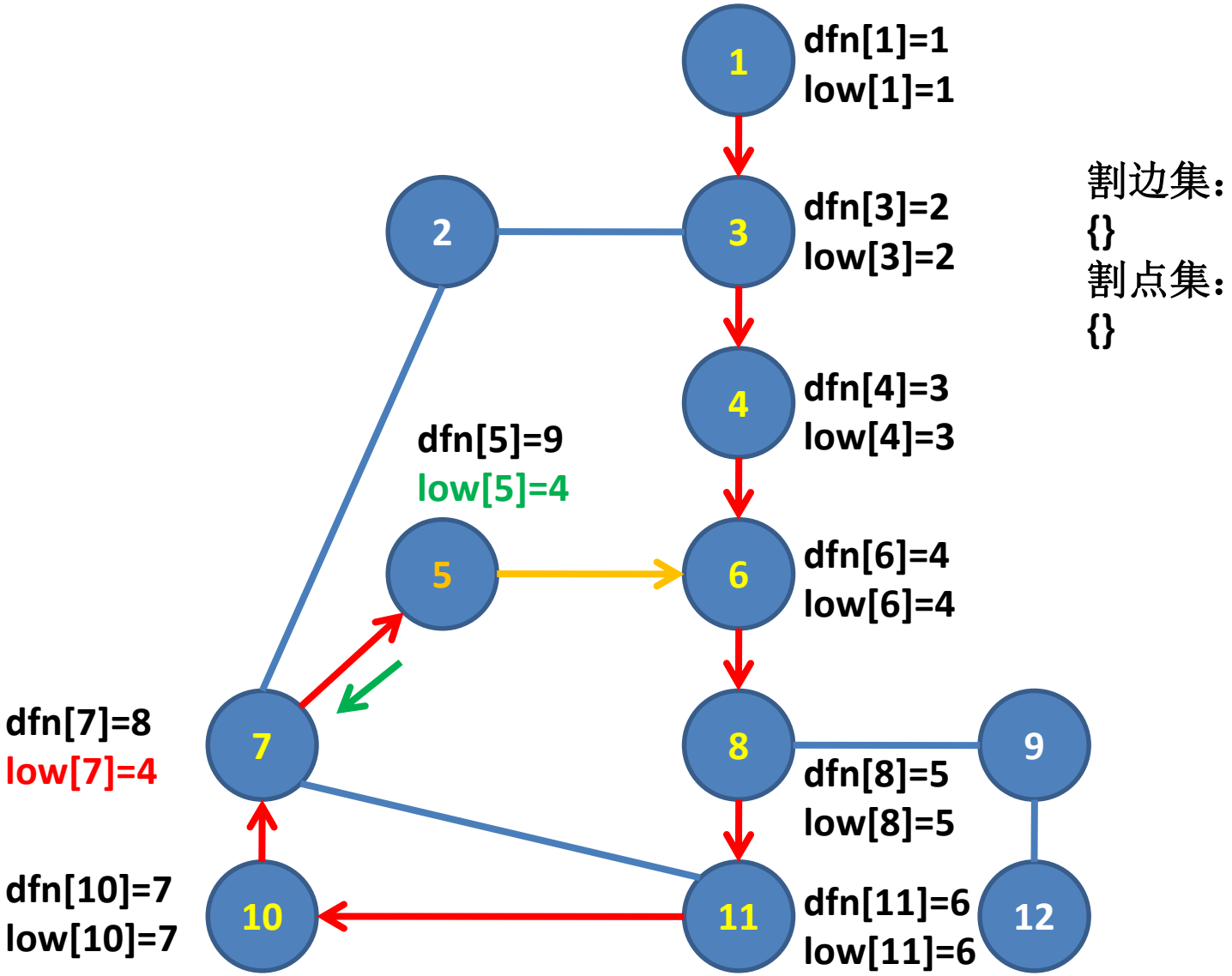


割边集：
{}
割点集：
{}

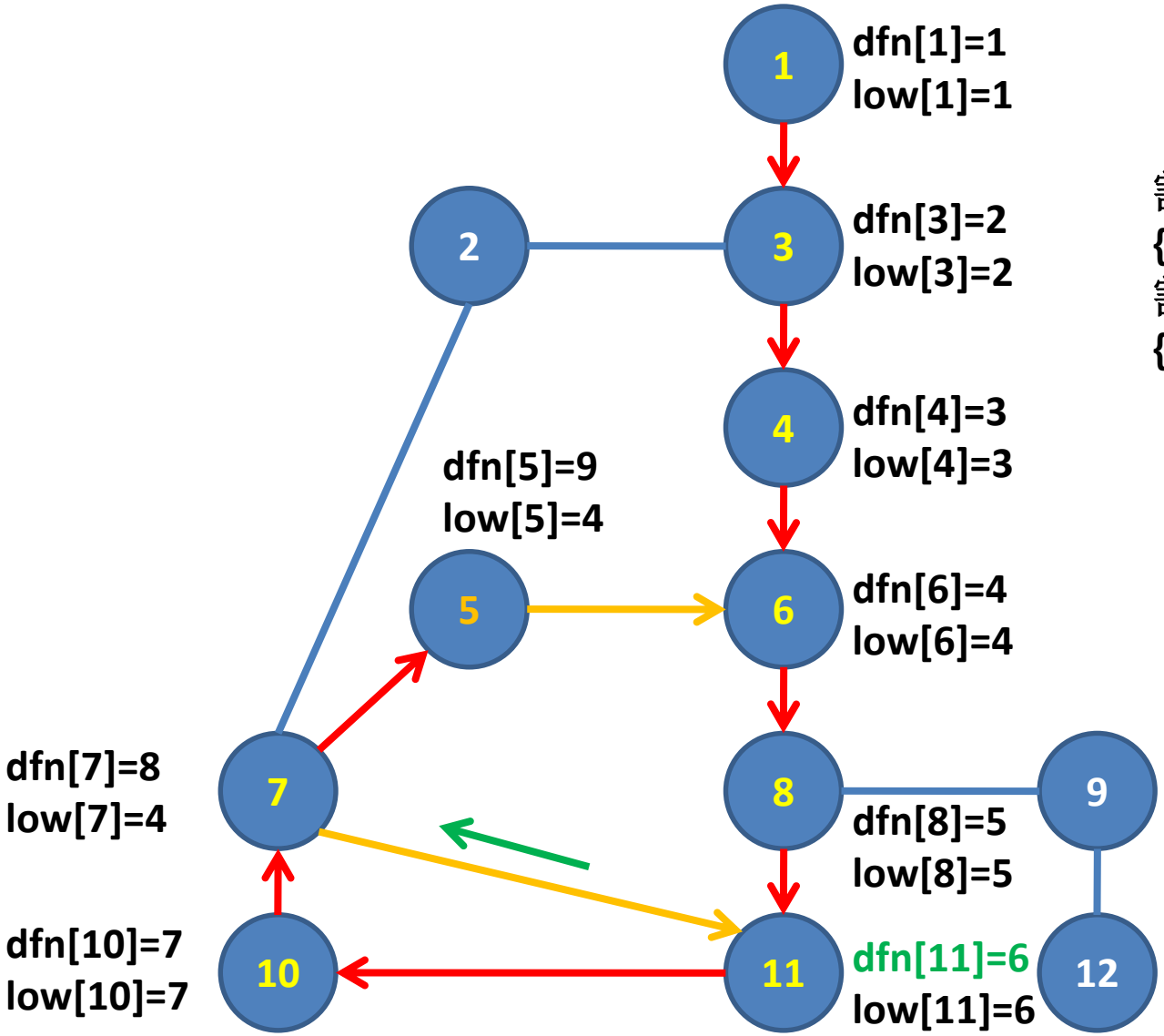
3、Tarjan’s Algorithm算法流程演示



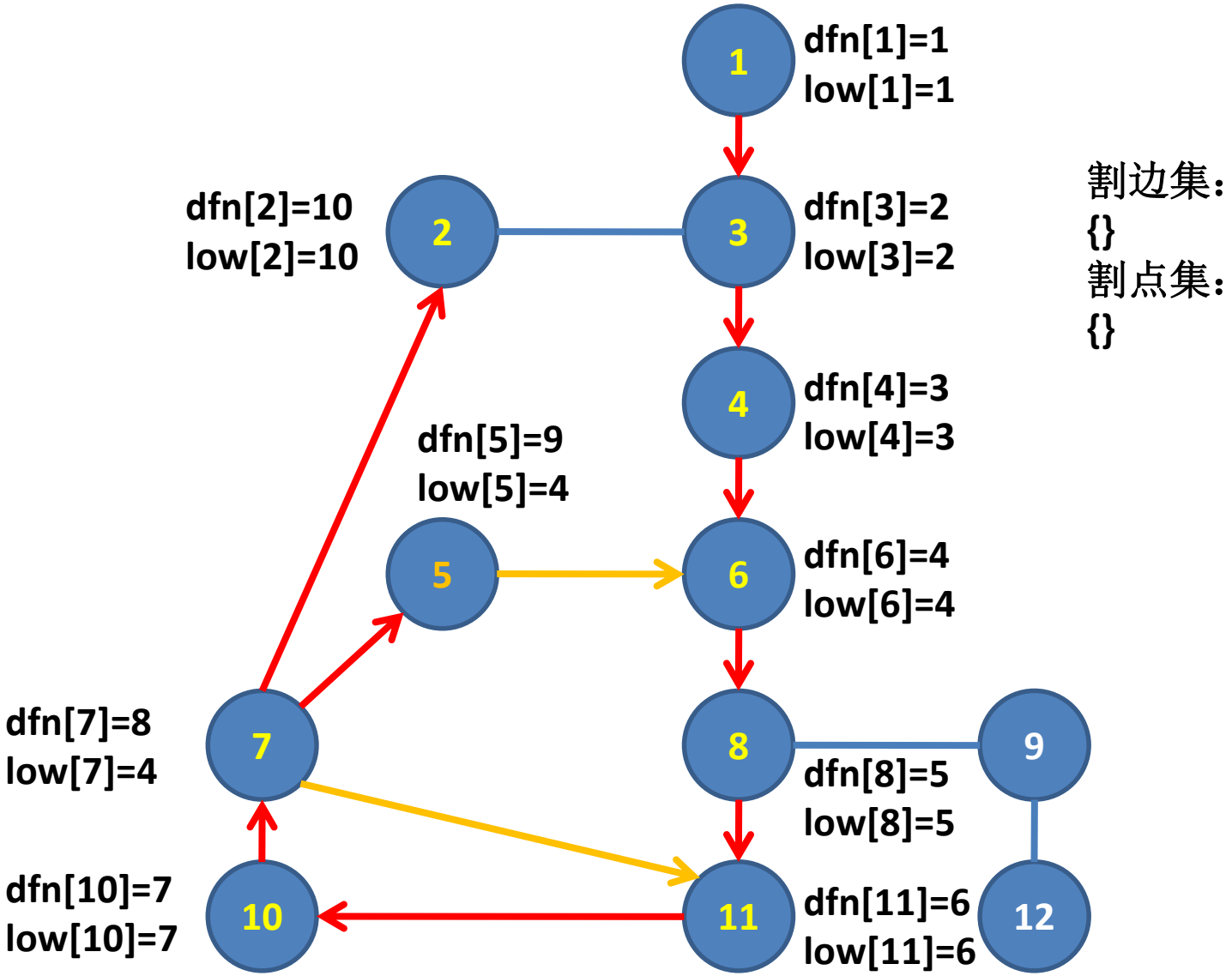
3、Tarjan’s Algorithm算法流程演示



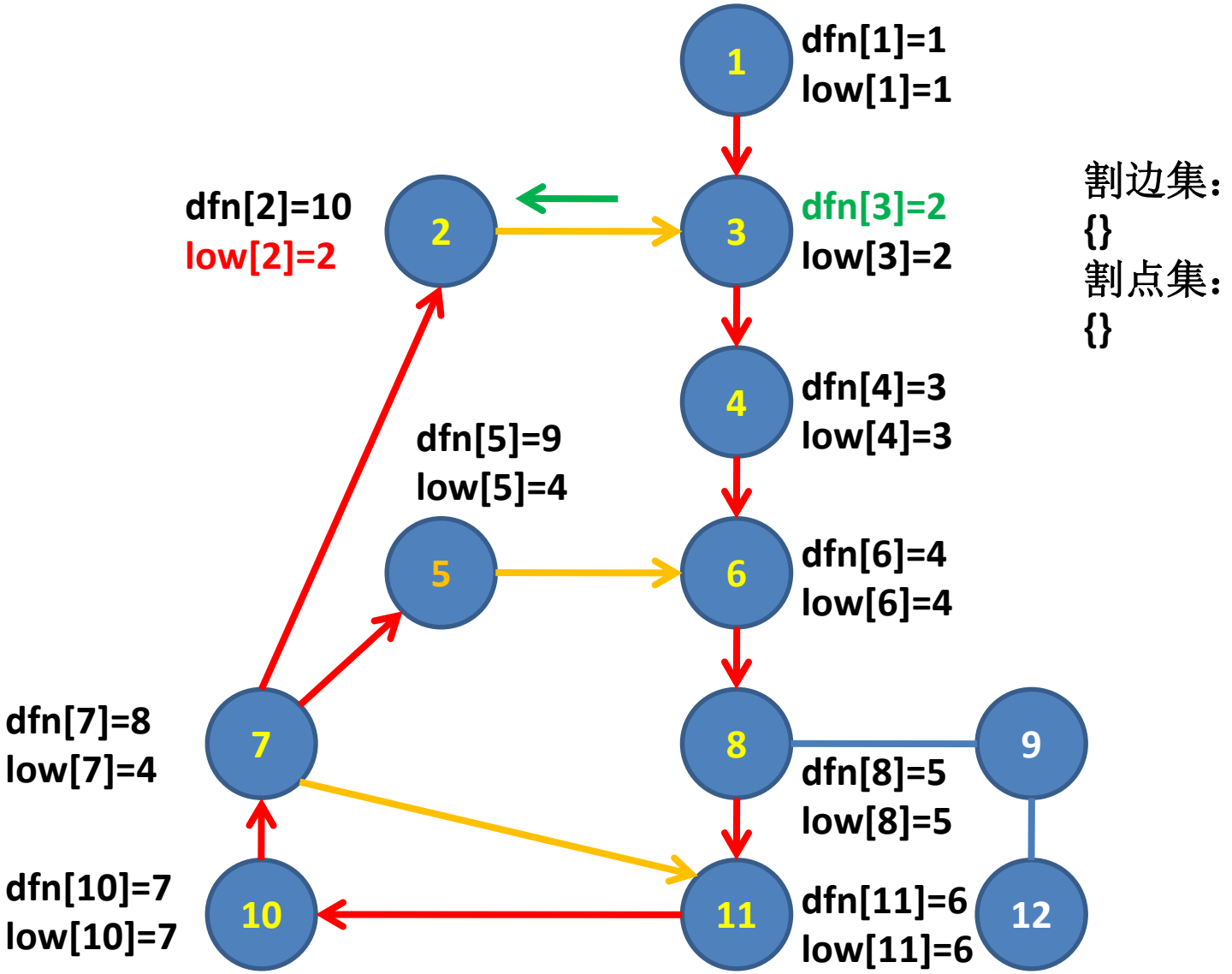
3、Tarjan’s Algorithm算法流程演示



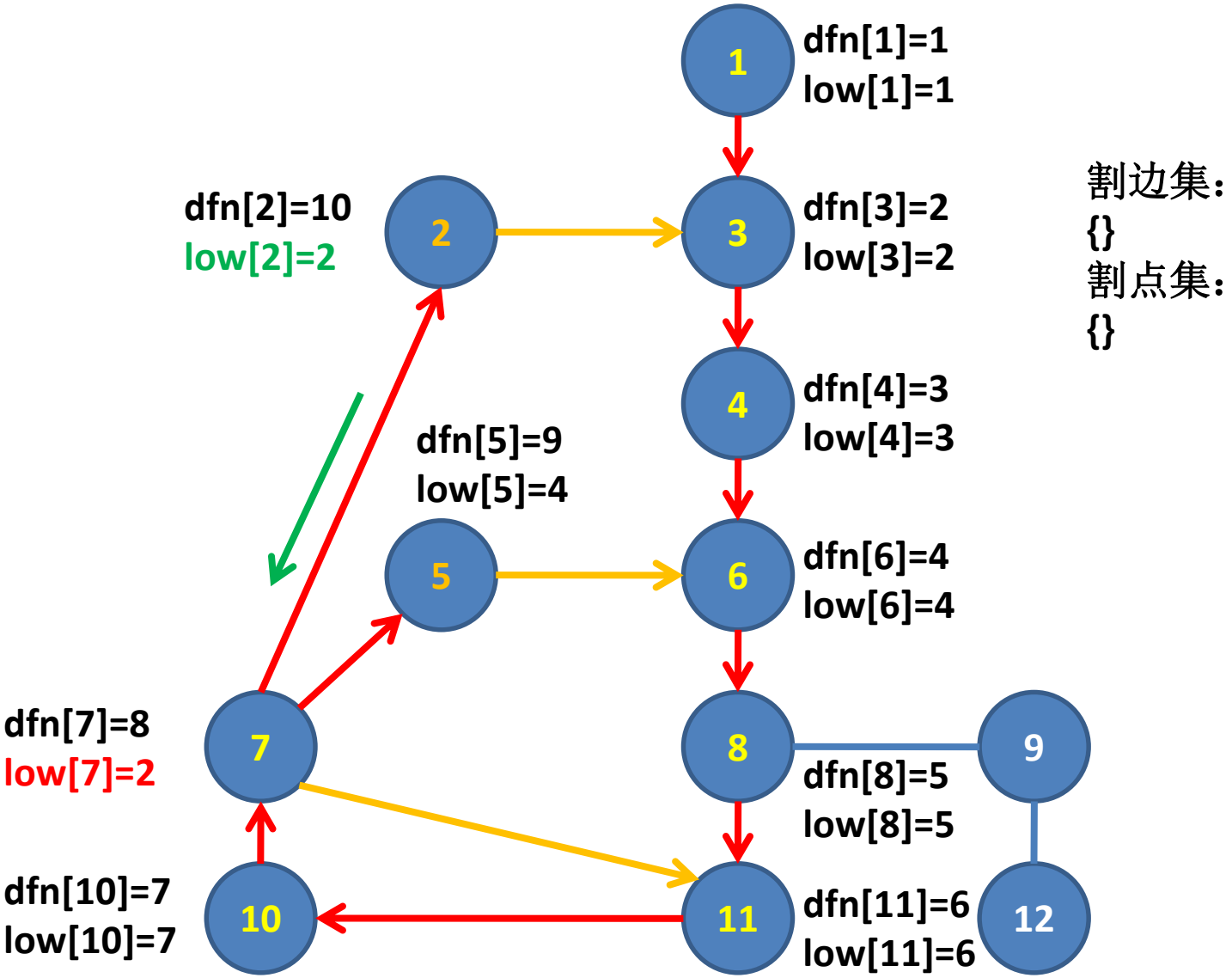
3、Tarjan’s Algorithm算法流程演示



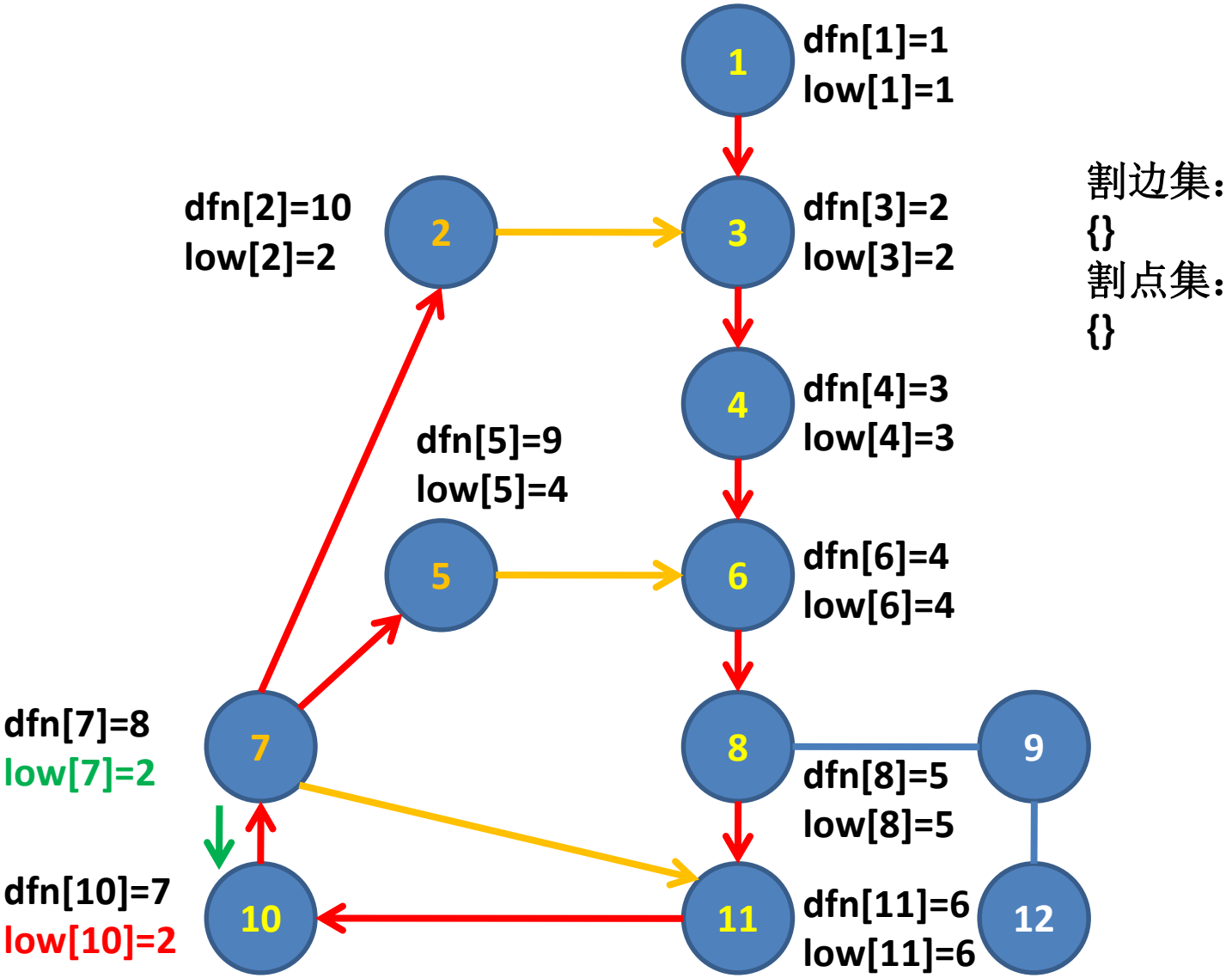
3、Tarjan’s Algorithm算法流程演示



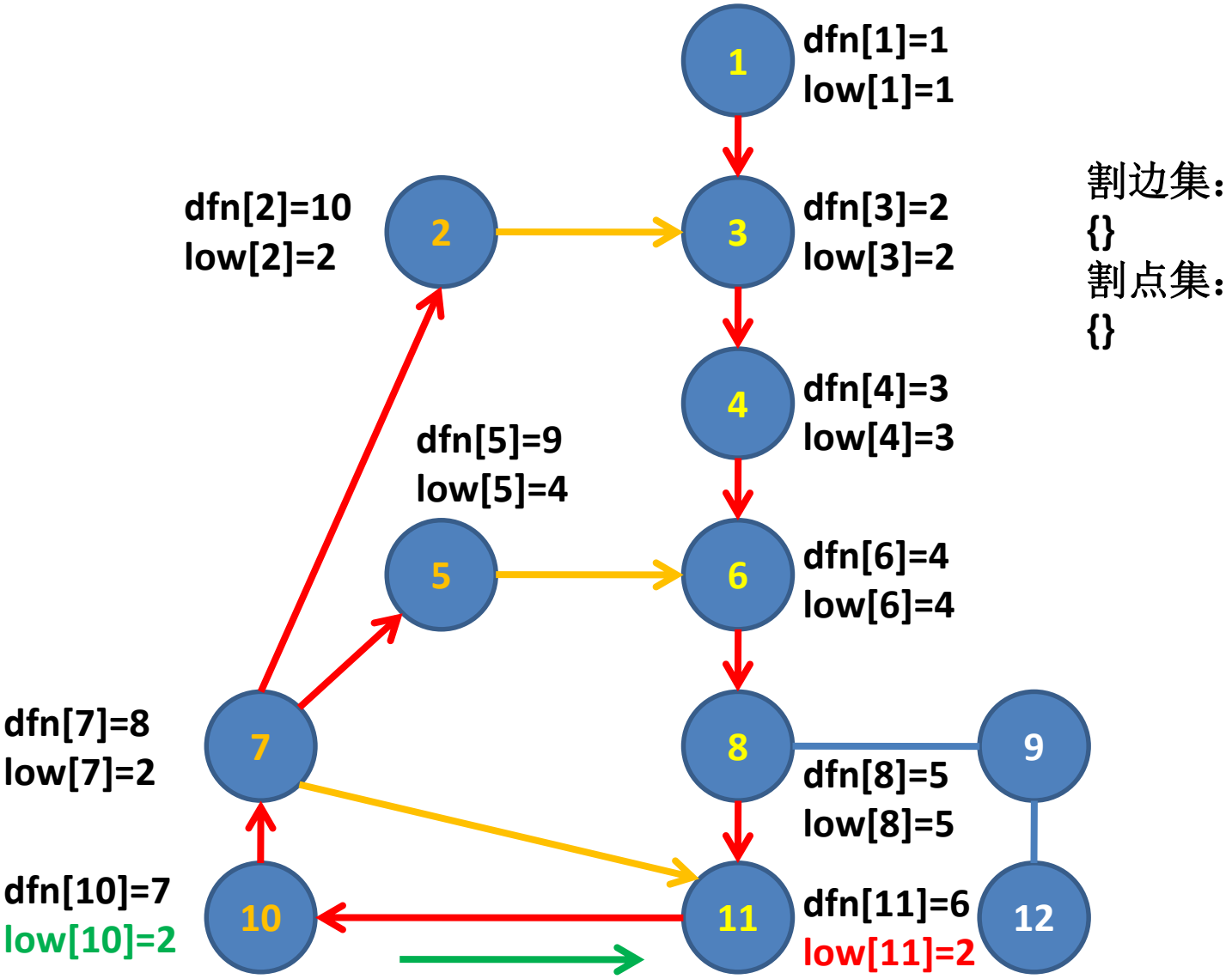
3、Tarjan’s Algorithm算法流程演示



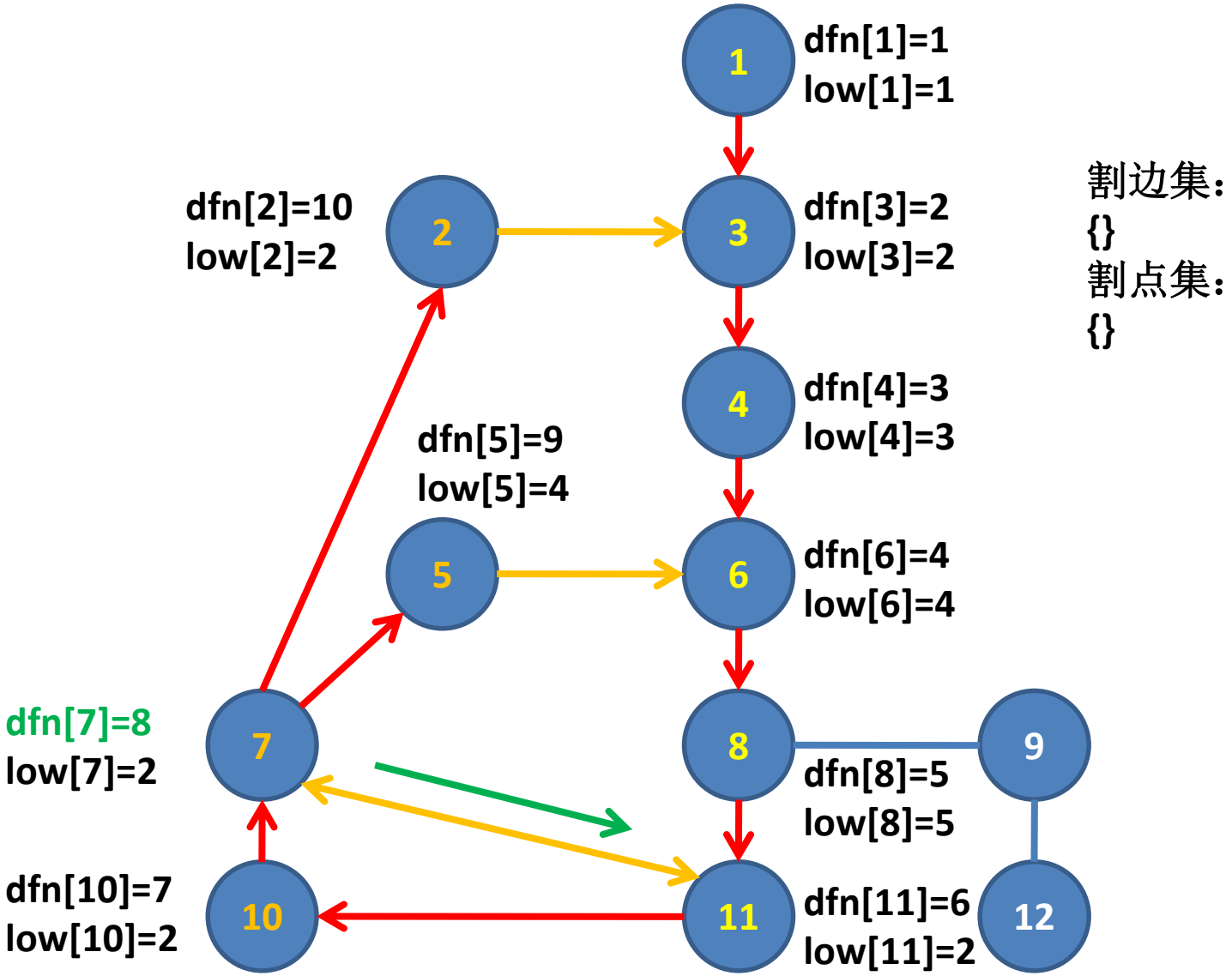
3、Tarjan’s Algorithm算法流程演示



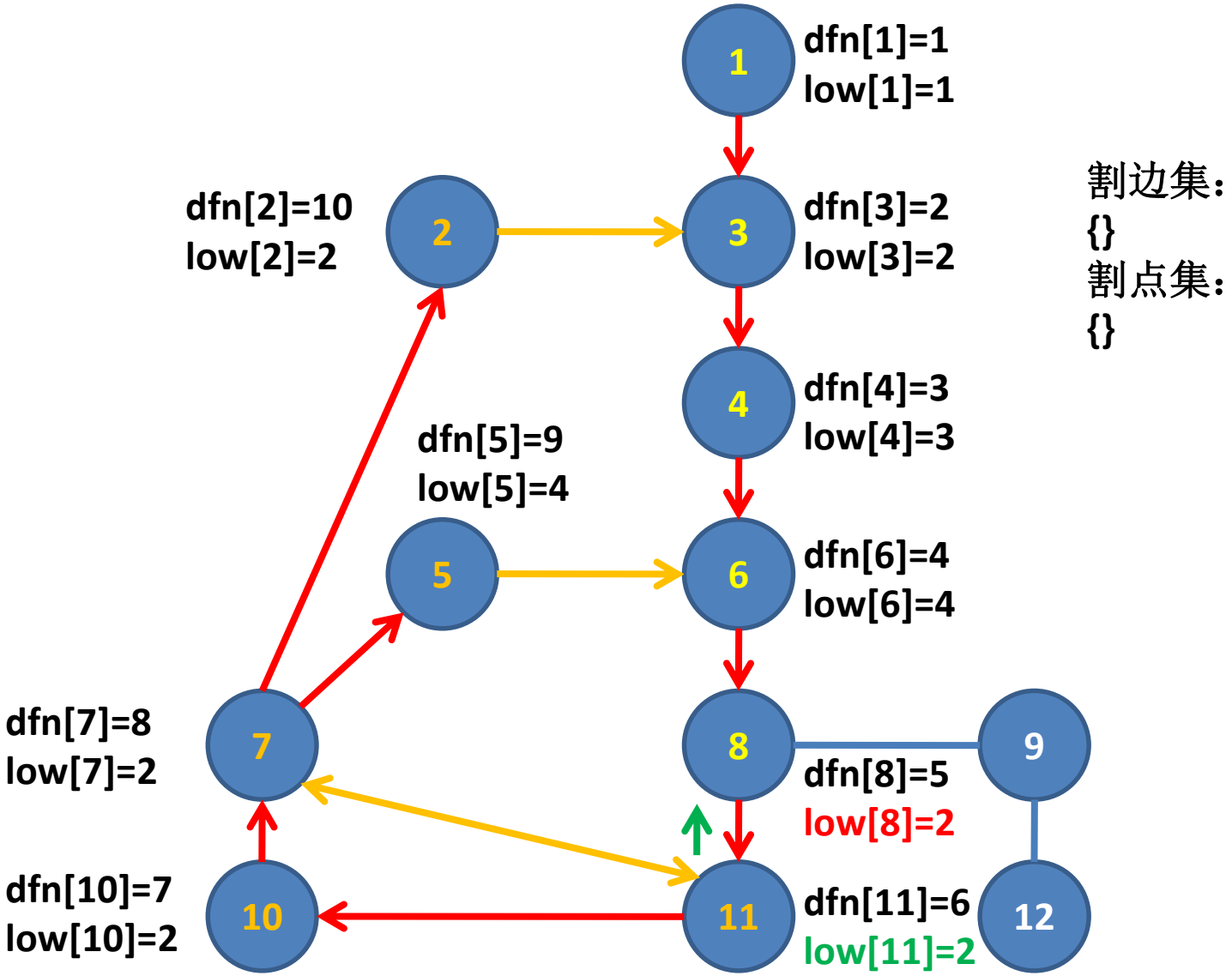
3、Tarjan’s Algorithm算法流程演示



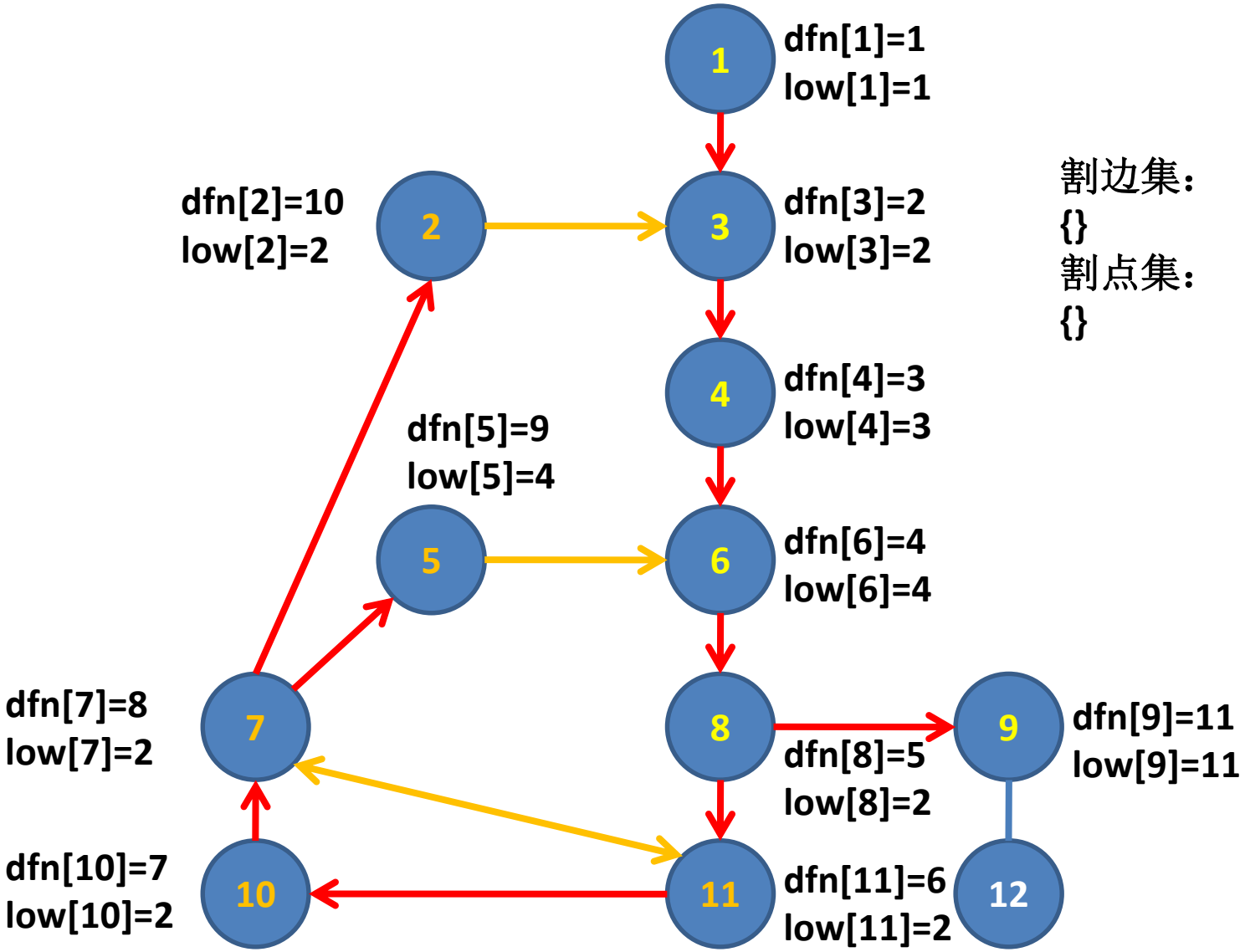
3、Tarjan’s Algorithm算法流程演示



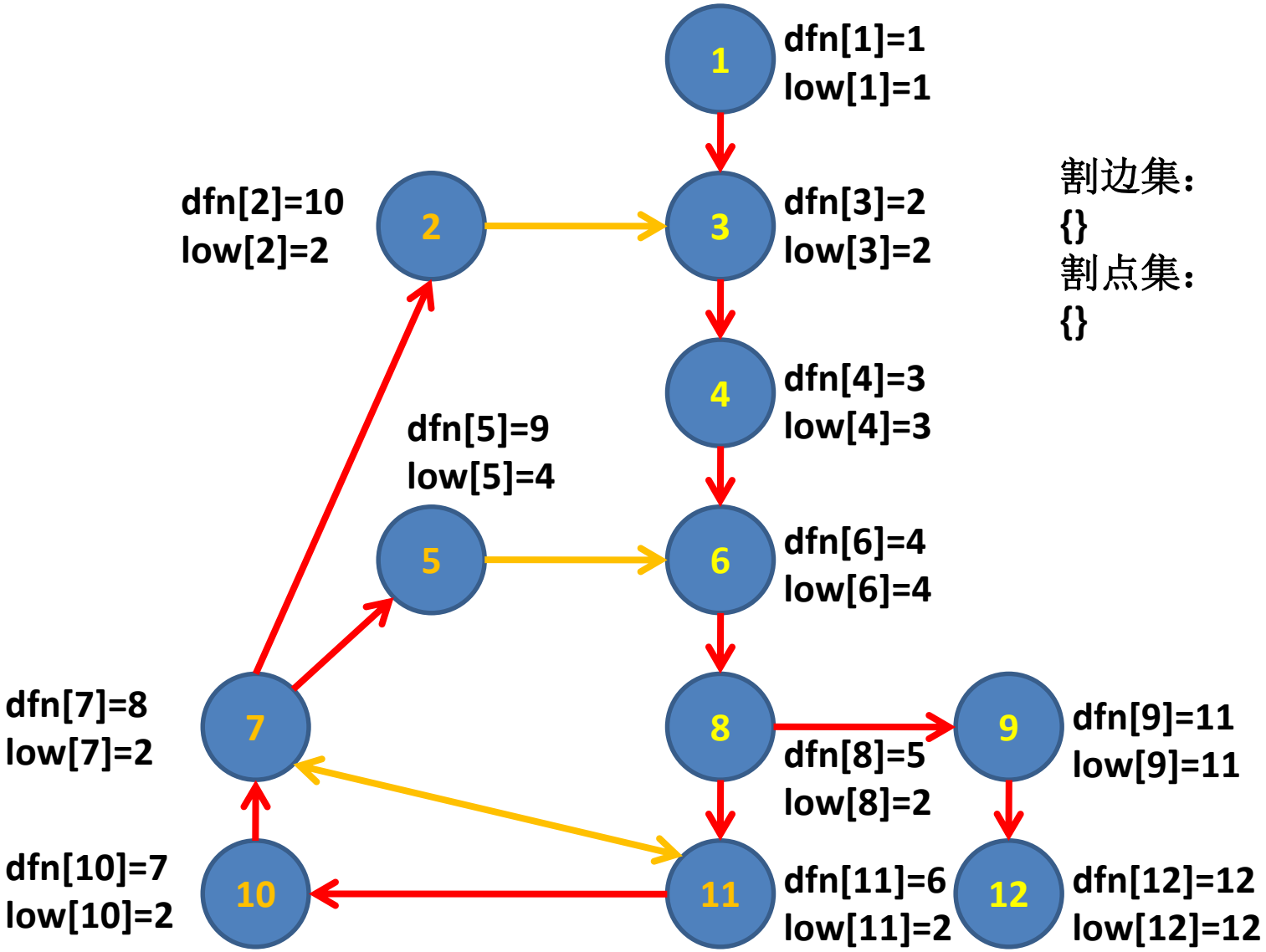
3、Tarjan’s Algorithm算法流程演示



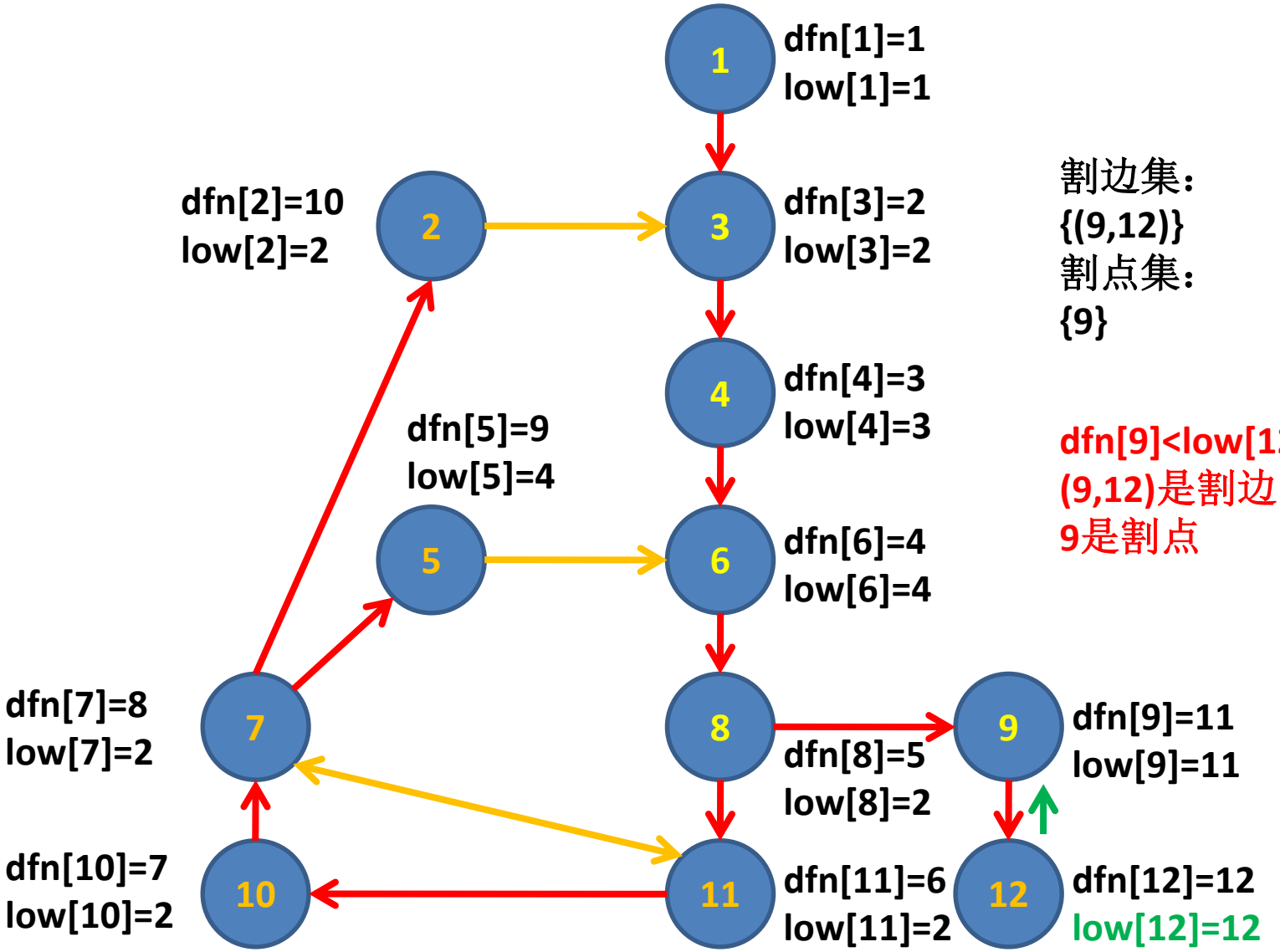
3、Tarjan’s Algorithm算法流程演示



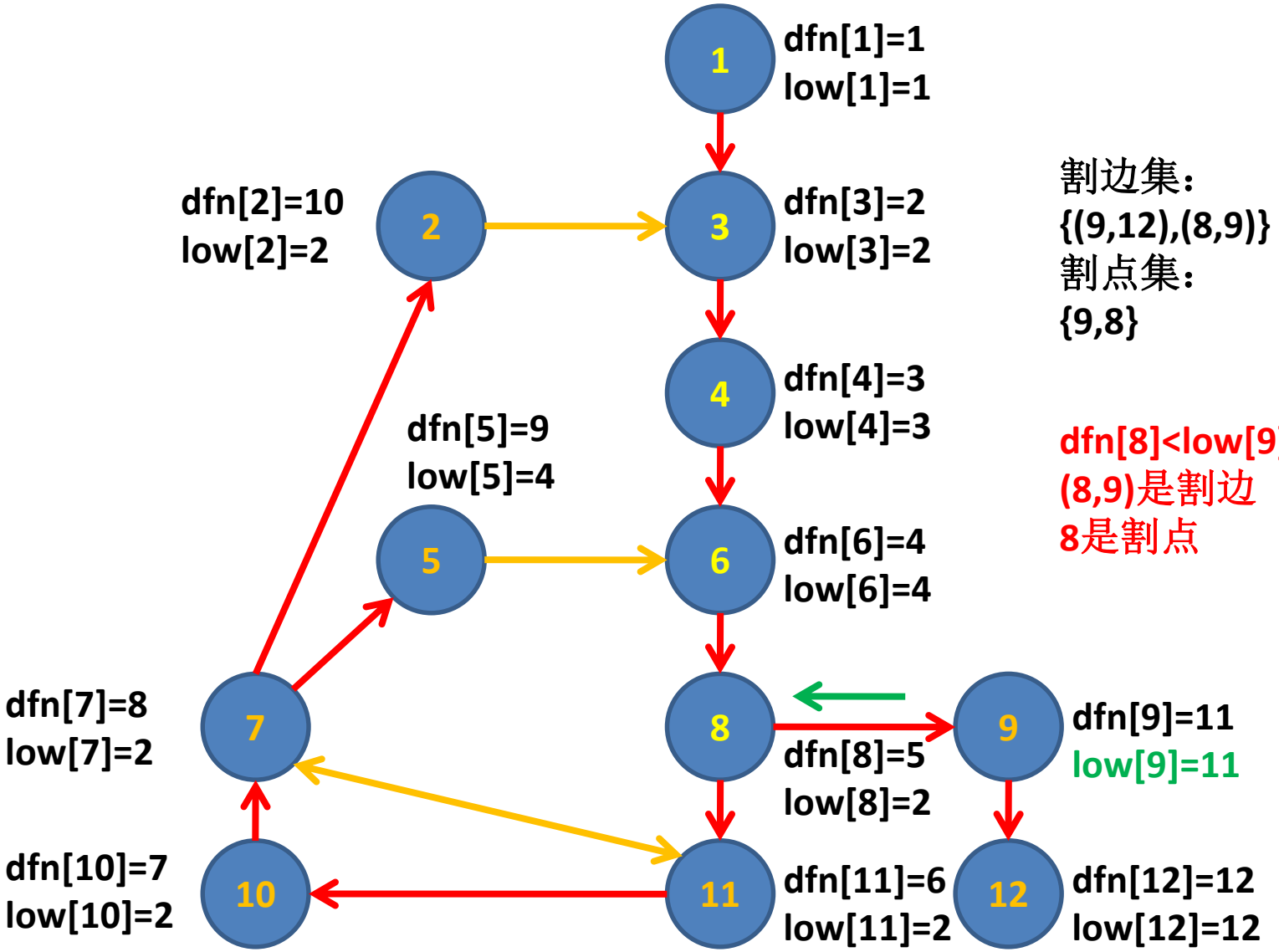
3、Tarjan’s Algorithm算法流程演示



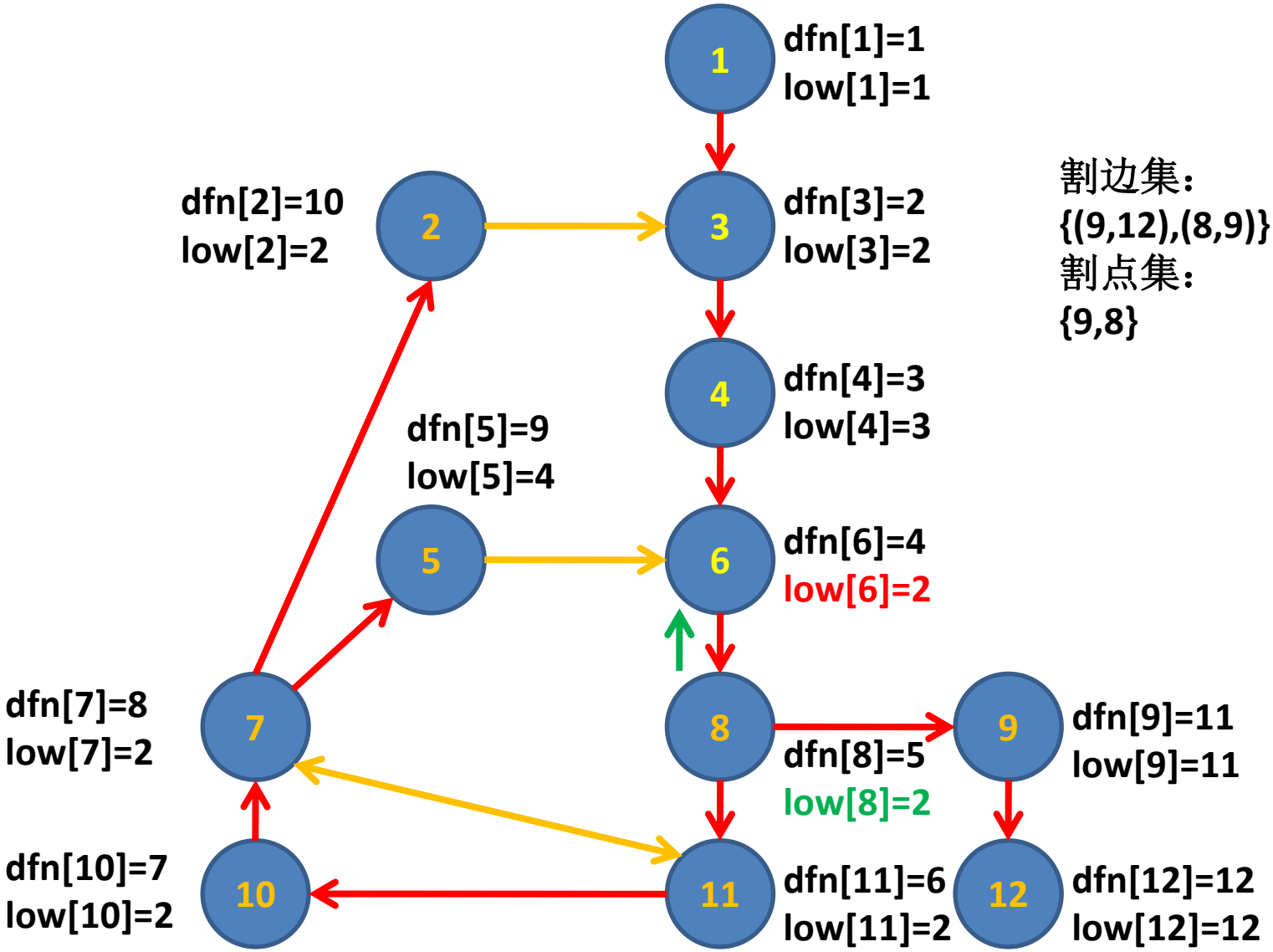
3、Tarjan’s Algorithm算法流程演示



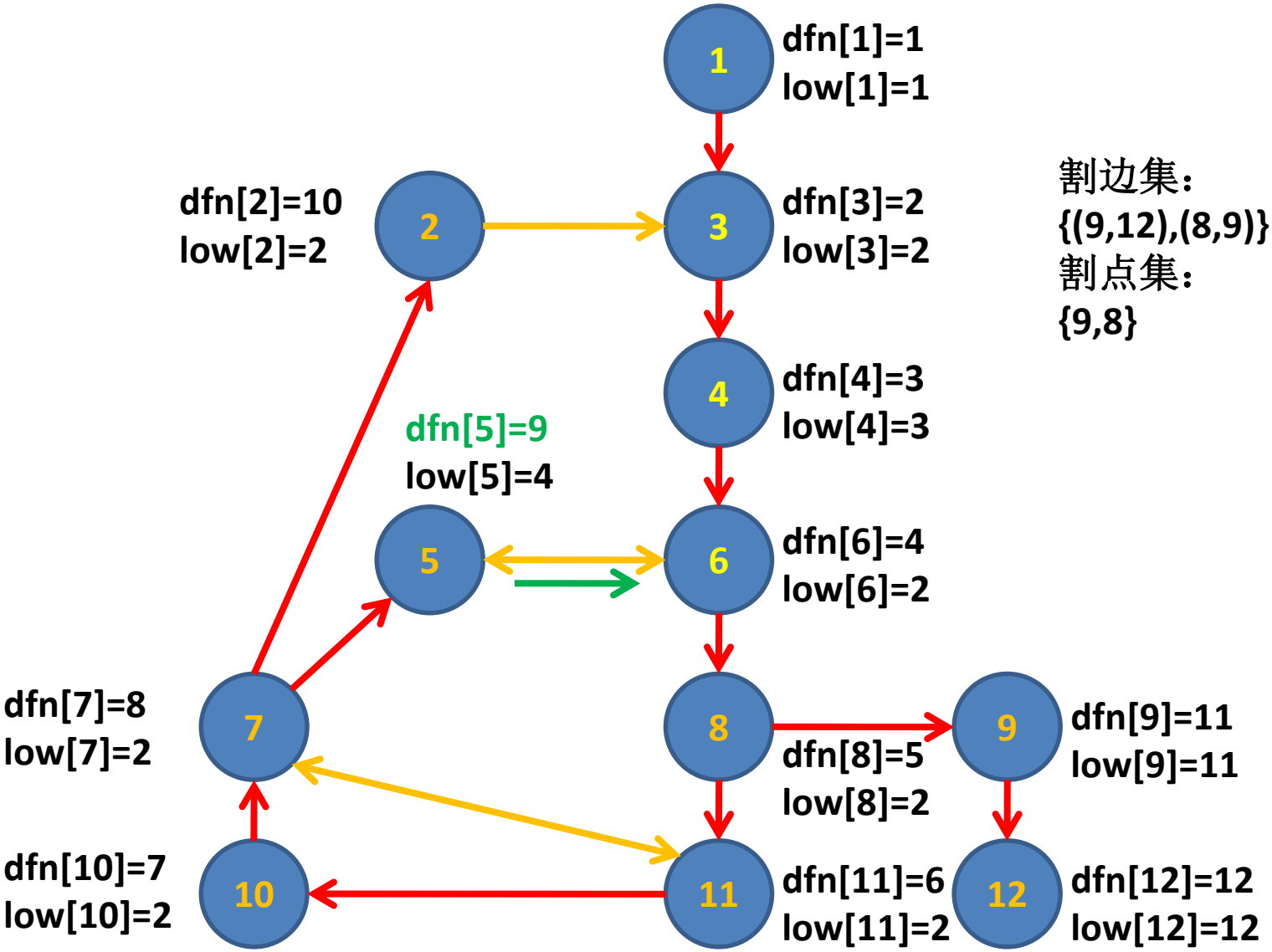
3、Tarjan's Algorithm算法流程演示



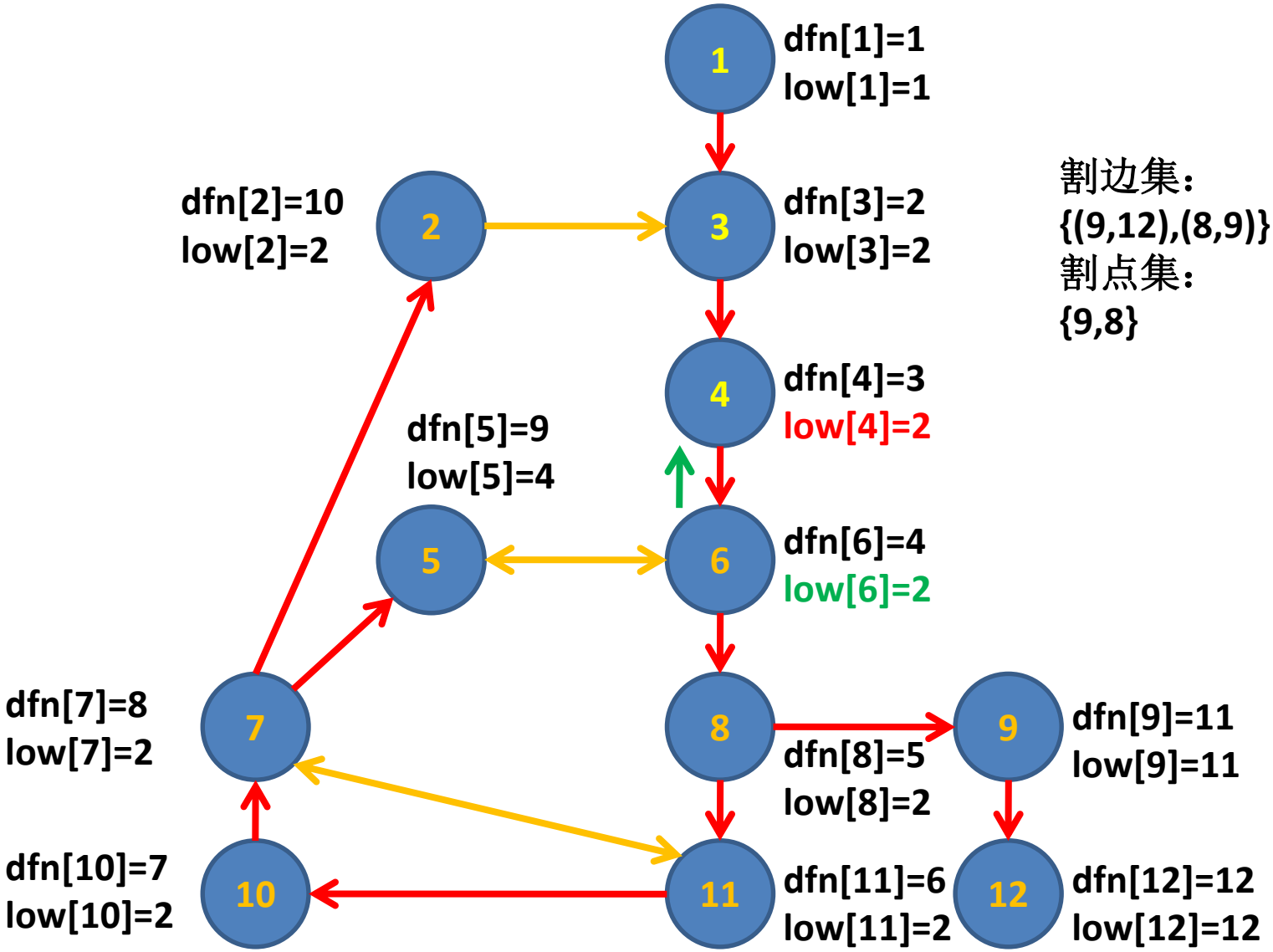
3、Tarjan’s Algorithm算法流程演示



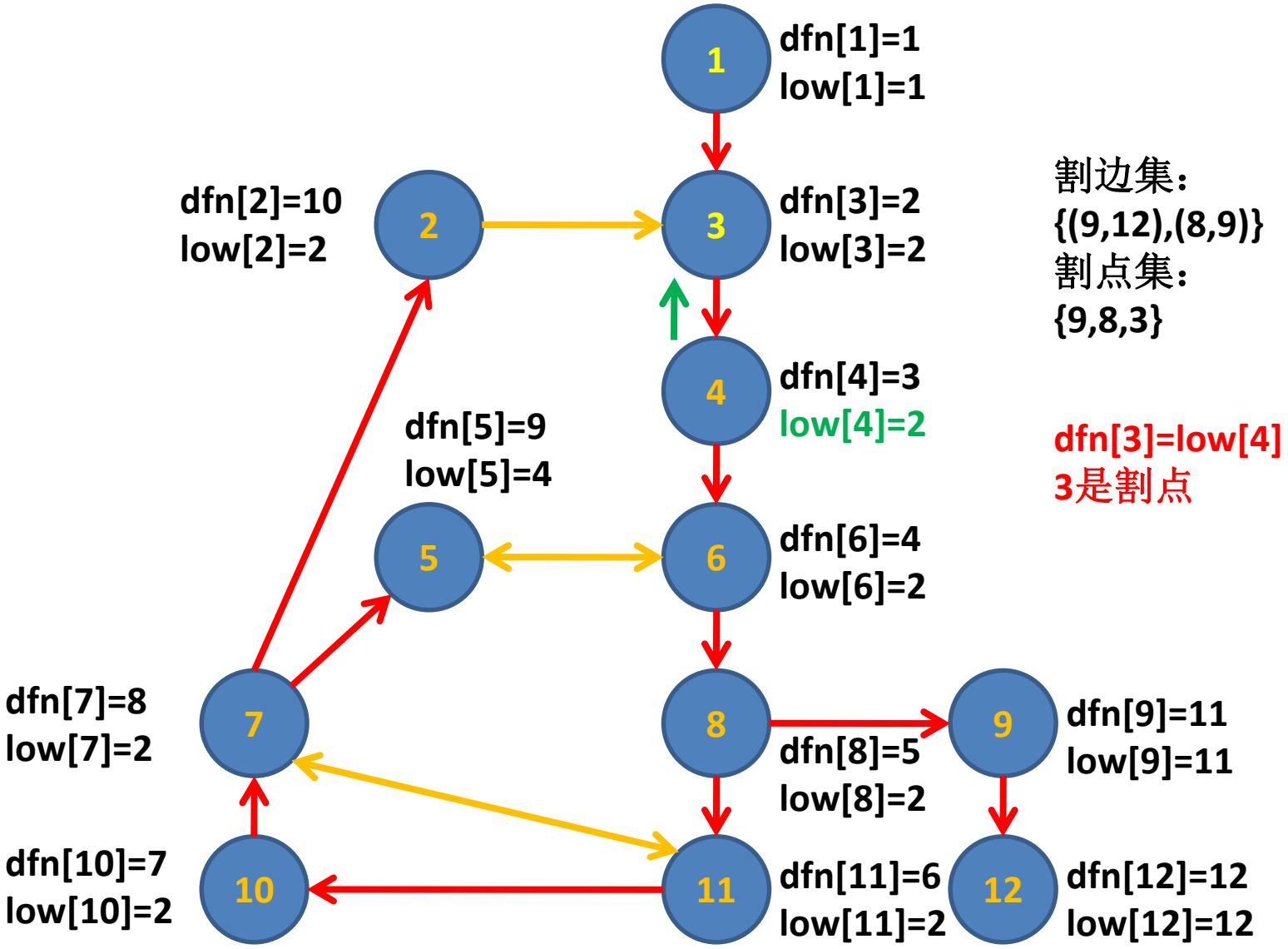
3、Tarjan’s Algorithm算法流程演示



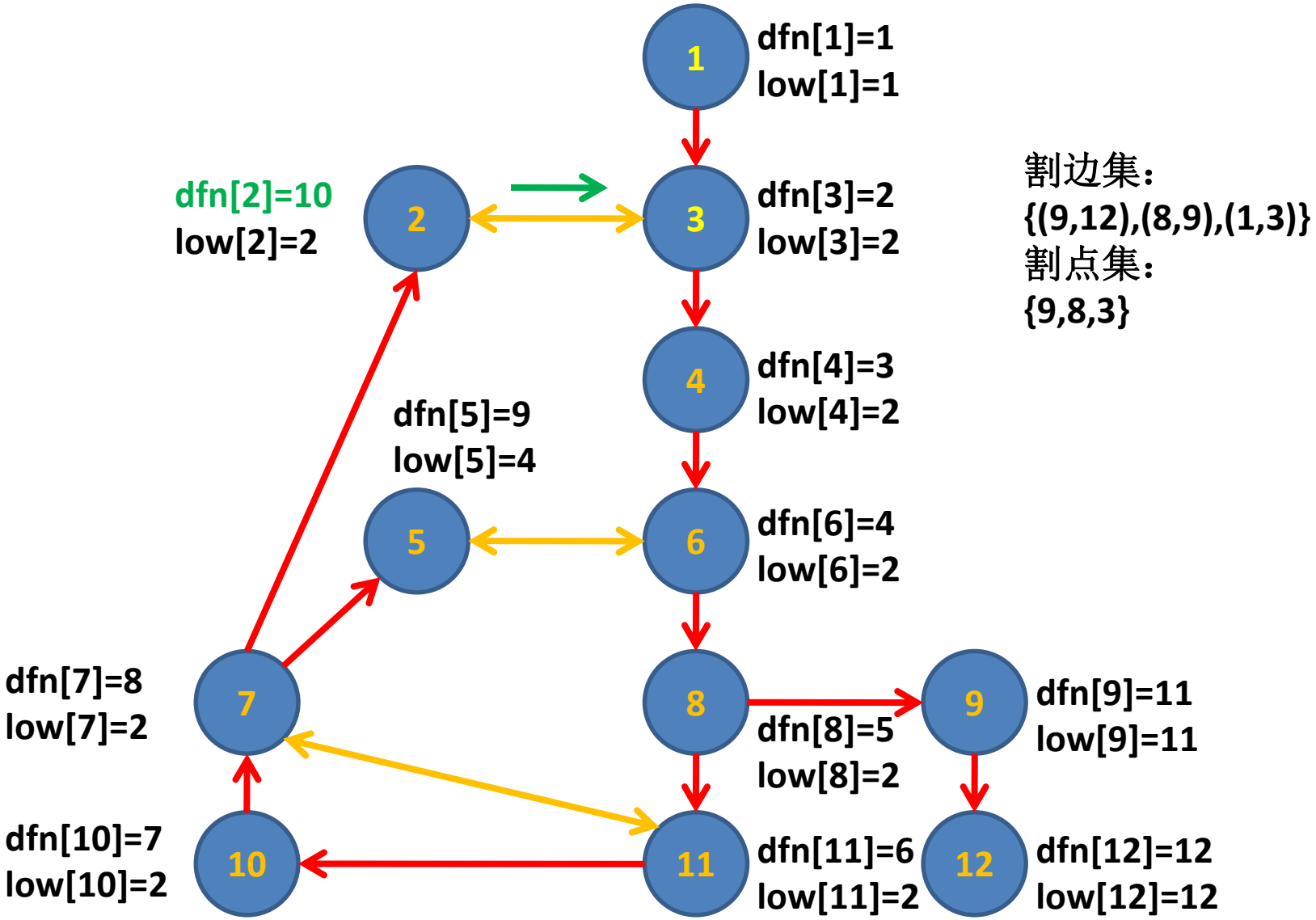
3、Tarjan’s Algorithm算法流程演示



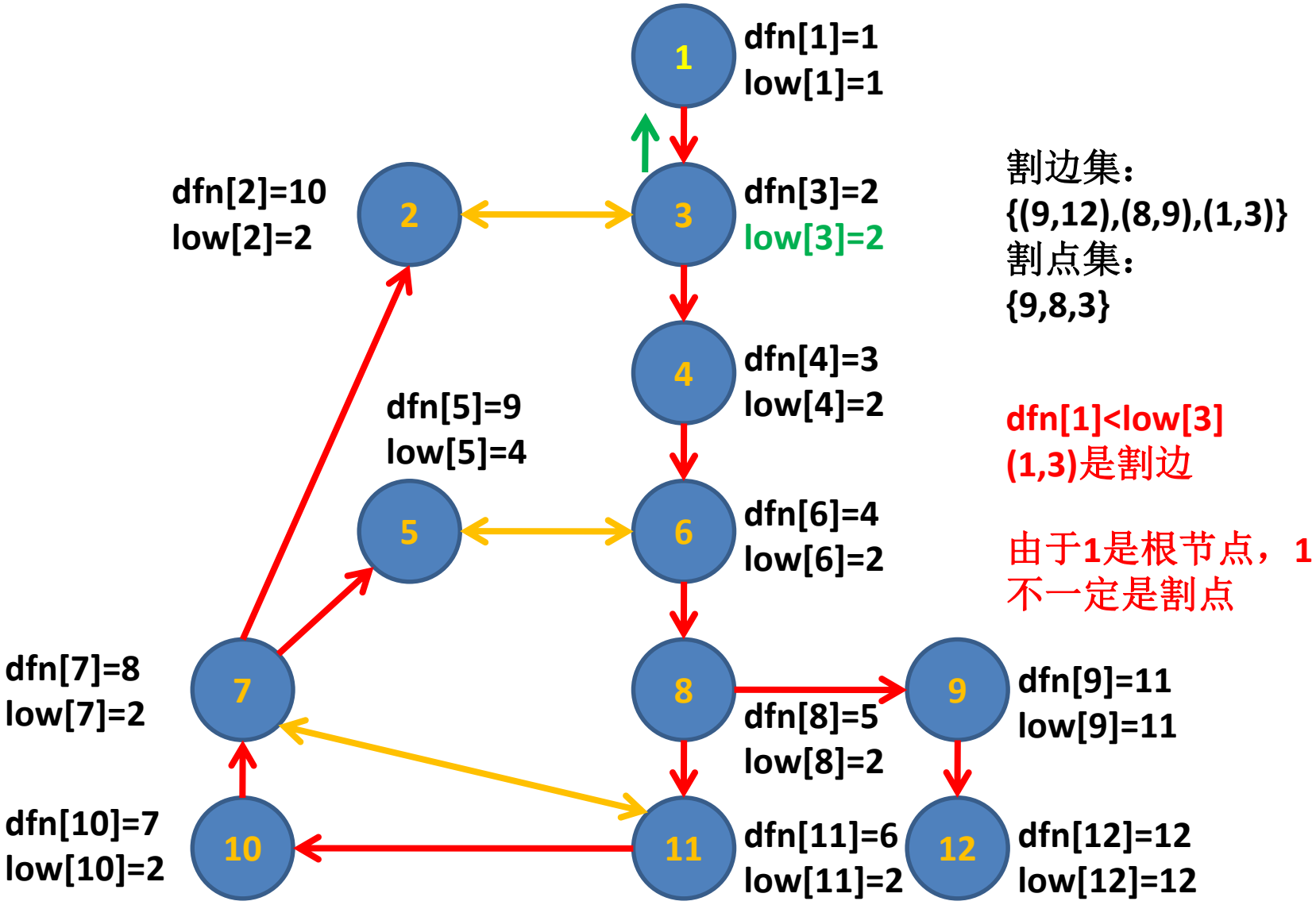
3、Tarjan’s Algorithm算法流程演示



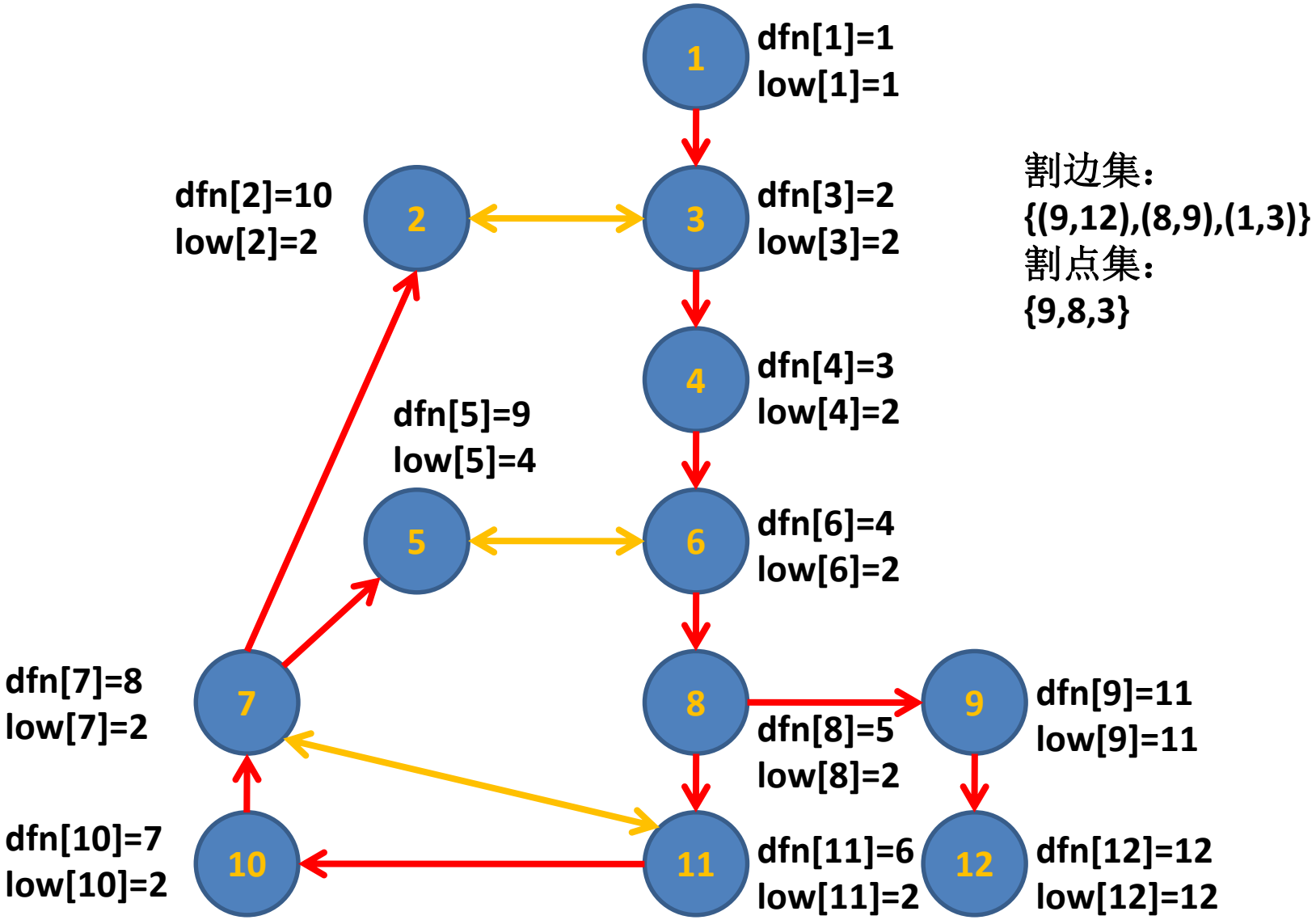
3、Tarjan’s Algorithm算法流程演示



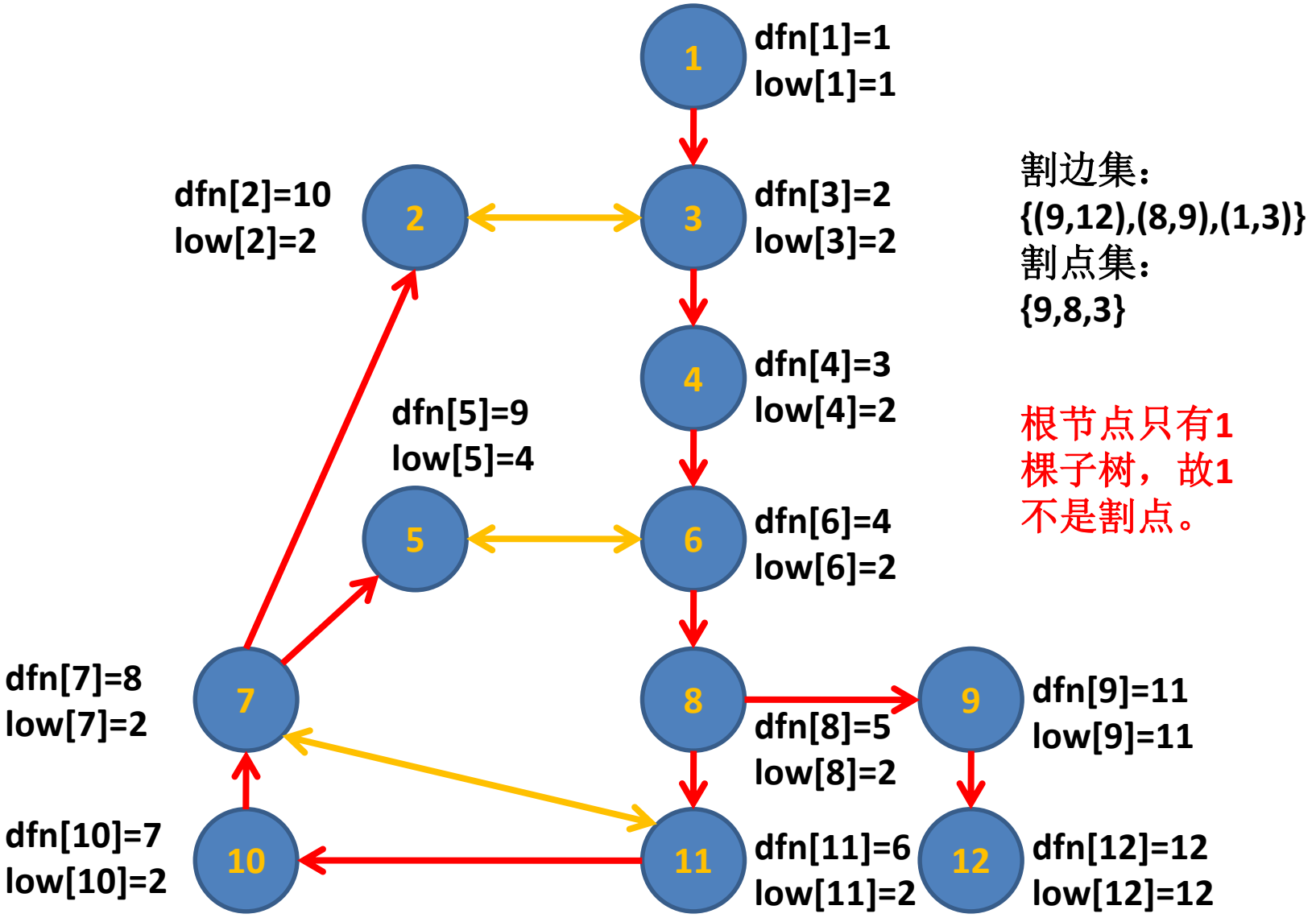
3、Tarjan’s Algorithm算法流程演示



3、Tarjan’s Algorithm算法流程演示



3、Tarjan’s Algorithm算法流程演示



4、Tarjan's Algorithm伪代码示例

```
tarjan(u)
{
    dfn[u]=low[u]=++Index      //为节点u设定次序编号和Low初值
    vis[u]=1                   //标记已访问
    for each(u,v) in E         //枚举每一条边
        if (v == parent[u])    //跳过父结点
            continue
        if (vis[v] == 0)       //如果节点v未被访问过,此边为树边
            parent[v] = u      //父结点记录
            tarjan(v)          //继续DFS
            low[u]=min(low[u],low[v])      //维护low[]
            if(dfn[u] < low[v])            //割边判定
                allCutEdge.push((u,v))     //加入割边集
            if(dfn[u] ≤ low[v] && u ≠ root)//割点判定
                allCutVertex.push(u)       //加入割点集
            if(u = root && Index > 1)
                allCutVertex.push(u)       //对根节点特判
        else                       //此边不是树边
            low[u]=min(low[u],dfn[v])      //维护low[]
}
```

5、Tarjan’s Algorithm复杂度分析

每个结点只被访问一次，在每个结点上算法的时间复杂度为**O(1)**

每条边最多被访问两次（两个端点各一次，树边只有一次），在每条边上算法的时间复杂度也是**O(1)**

因此**Tarjan’s Algorithm**总的时间复杂度为**O(V+E)**

使用的空间相比深度优先搜索算法多了**dfn[]**，**low[]**，两个数组，因此空间复杂度为**O(V+E)**

6、Tarjan’s Algorithm求双连通分量

Tarjan’s Algorithm还可以用来求无向图的点双联通分量和边双联通分量。

点双联通分量：不含割点的极大联通分量。
点双联通分量是对边集的一个划分。
两个点双联通分量由一个割点连接。

边双联通分量：不含割边的极大联通分量。
边双联通分量是对点集的一个划分。
两个边双联通分量由一条割边连接。

6.1、Tarjan’s Algorithm求点双连通分量

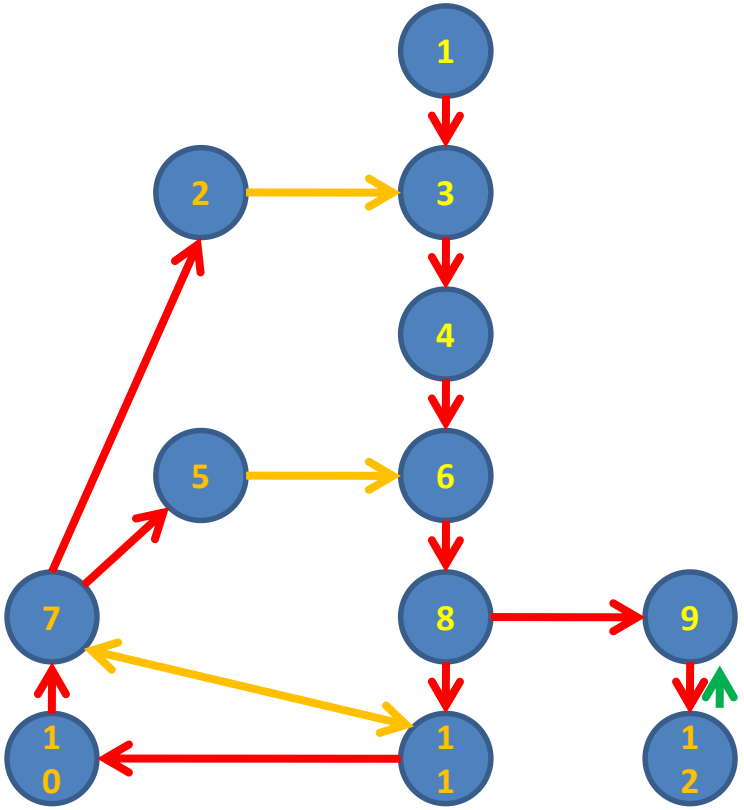
点双联通分量：不含割点的极大联通分量。
点双联通分量是对边集的一个划分。
两个点双联通分量由一个割点连接。

实现方式：
在求割点算法的基础上，每次搜索到一条边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。

6.1、Tarjan’s Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



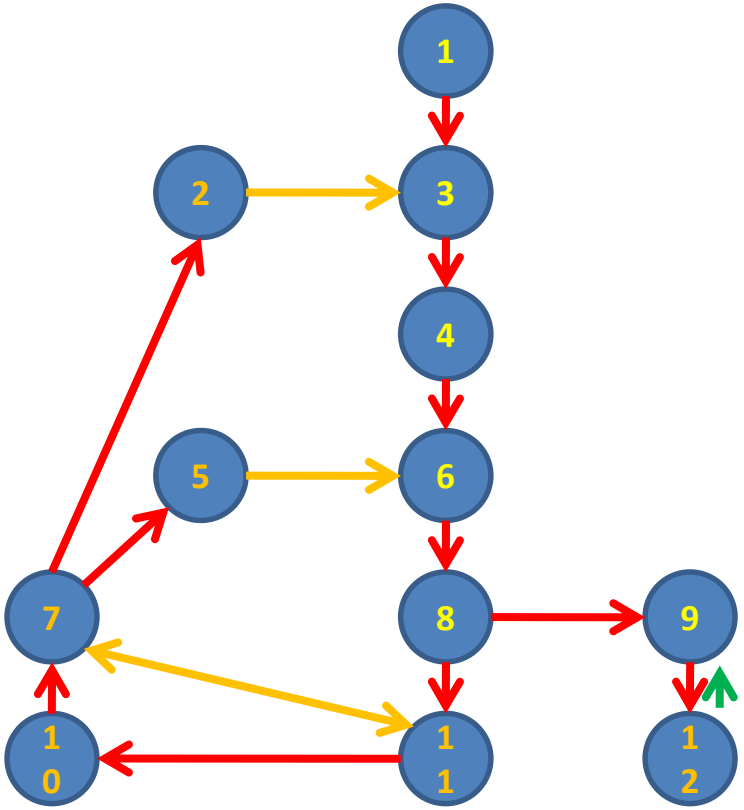
栈

(1,3)
(3,4)
(4,6)
(6,8)
(8,11)
(11,10)
(10,7)
(7,5)
(5,6)
(7,2)
(2,3)
(7,11)
(8,9)
(9,12)

6.1、Tarjan's Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



栈

(1,3)
(3,4)
(4,6)
(6,8)
(8,11)
(11,10)
(10,7)
(7,5)
(5,6)
(7,2)
(2,3)
(7,11)
(8,9)

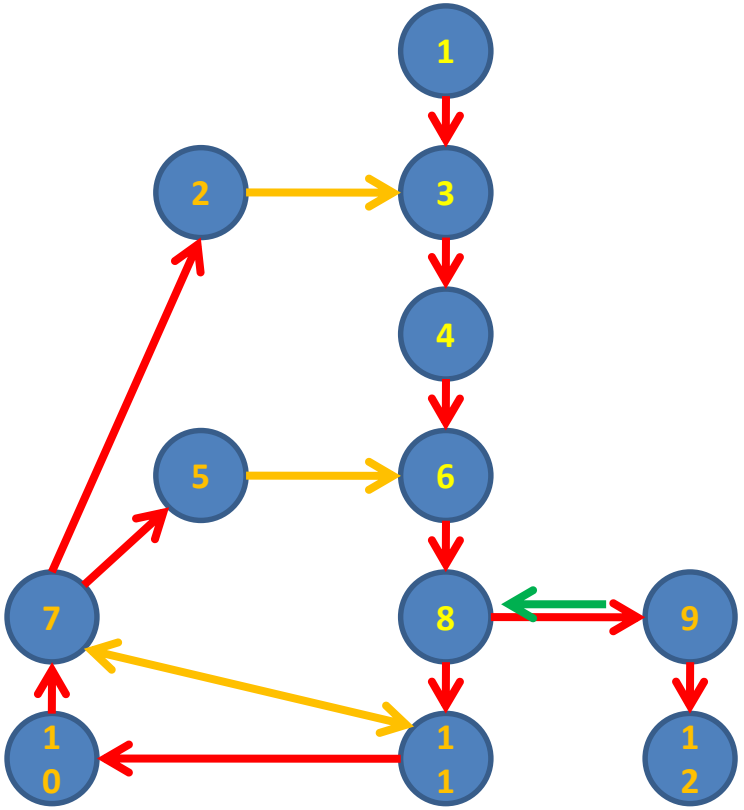
9是割点

(9,12)

6.1、Tarjan's Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



栈

(1,3)
(3,4)
(4,6)
(6,8)
(8,11)
(11,10)
(10,7)
(7,5)
(5,6)
(7,2)
(2,3)
(7,11)
(8,9)

9是割点

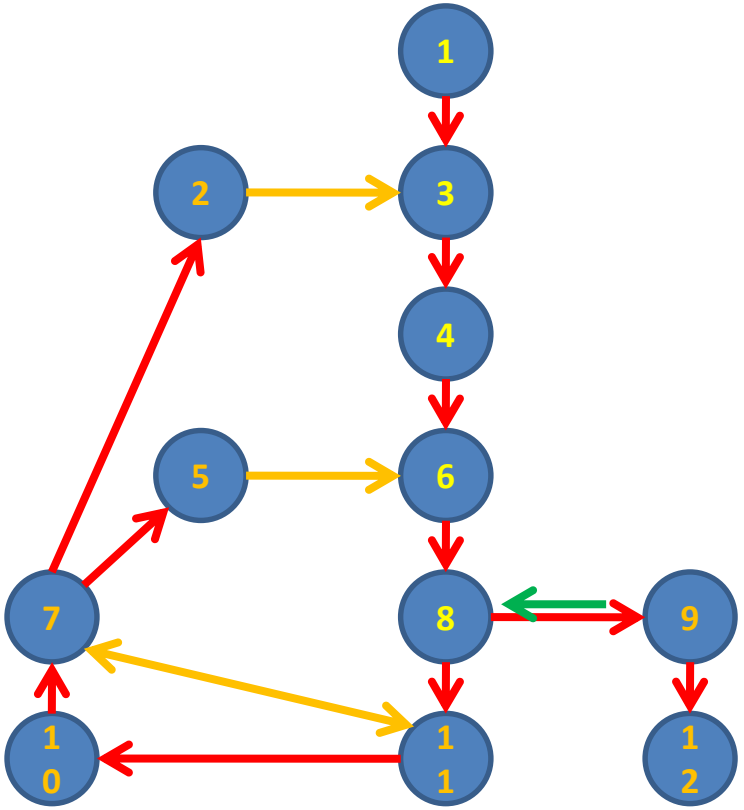
(9,12)

8是割点

6.1、Tarjan’s Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



栈

(1,3)
(3,4)
(4,6)
(6,8)
(8,11)
(11,10)
(10,7)
(7,5)
(5,6)
(7,2)
(2,3)
(7,11)

9是割点

(9,12)

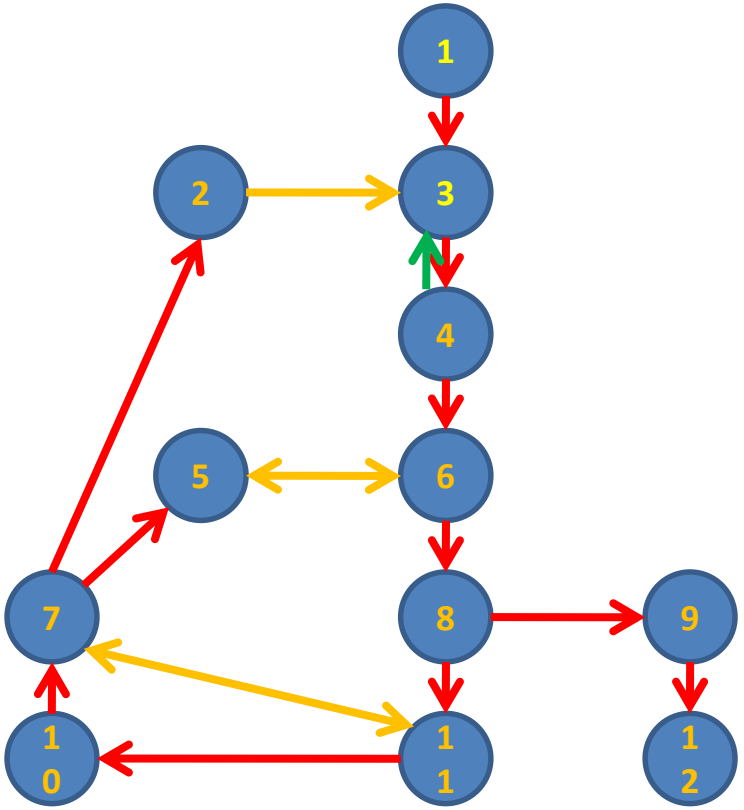
8是割点

(8,9)

6.1、Tarjan’s Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



栈

(1,3)
(3,4)
(4,6)
(6,8)
(8,11)
(11,10)
(10,7)
(7,5)
(5,6)
(7,2)
(2,3)
(7,11)

9是割点

(9,12)

8是割点

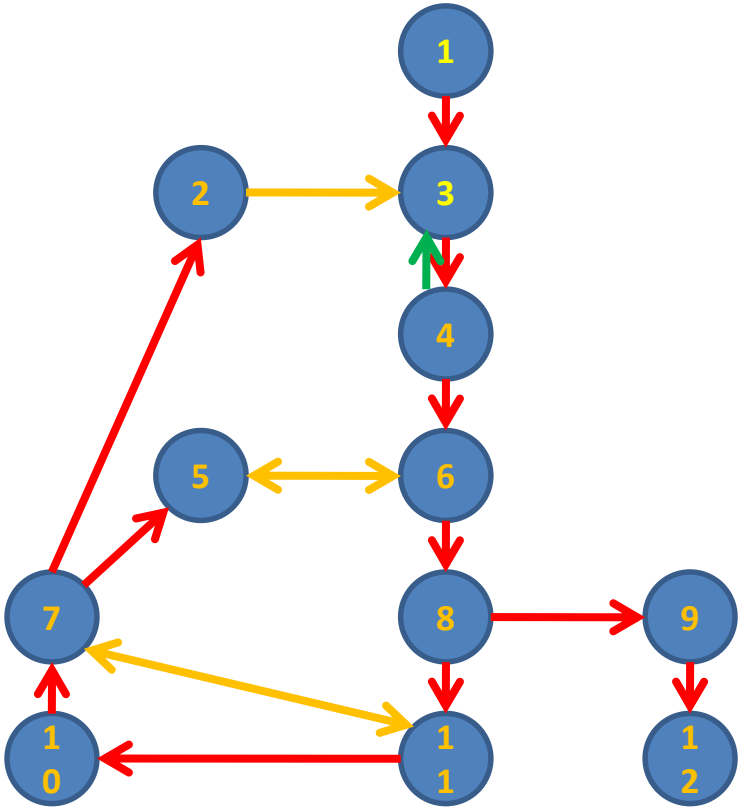
(8,9)

3是割点

6.1、Tarjan’s Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



栈

(1,3)

9是割点

(9,12)

8是割点

(8,9)

3是割点

(3,4)

(4,6)

(6,8)

(8,11)

(11,10)

(10,7)

(7,5)

(5,6)

(7,2)

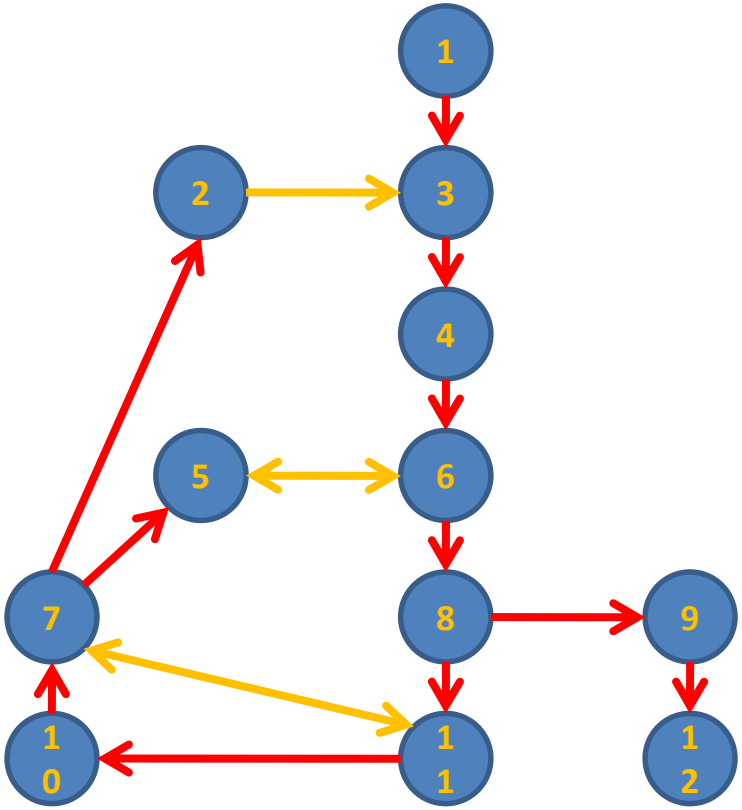
(2,3)

(7,11)

6.1、Tarjan's Algorithm求点双连通分量

实现方式：

在求割点算法的基础上，每次搜索到一条新的边，就将其压入栈中。当发现一个割点时，排出栈顶元素直到排出相应的边。



栈

(1,3)

9是割点

(9,12)

8是割点

(8,9)

3是割点

(3,4)

(4,6)

(6,8)

(8,11)

(11,10)

(10,7)

(7,5)

(5,6)

(7,2)

(2,3)

(7,11)

6.2、Tarjan’s Algorithm求边双连通分量

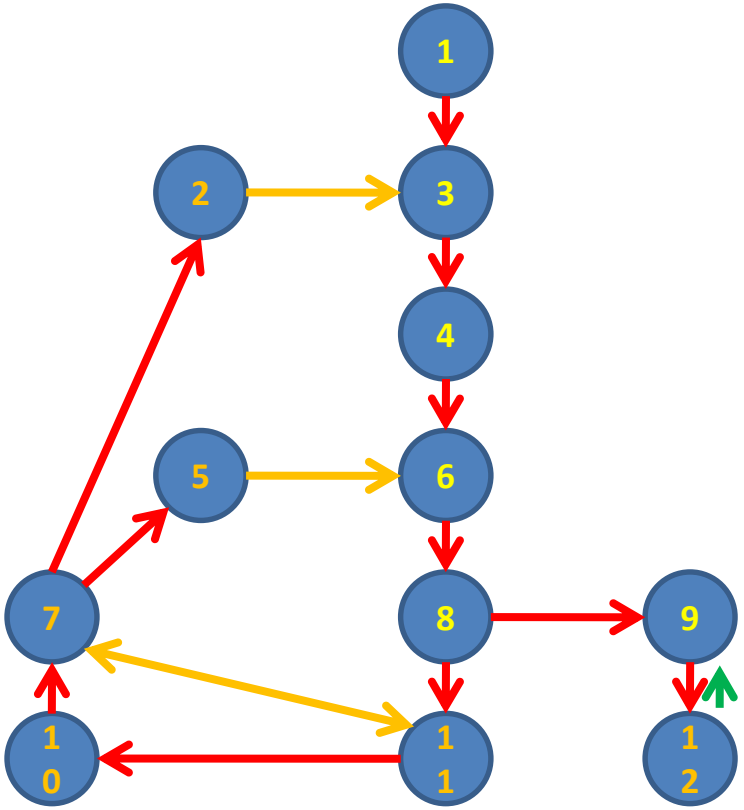
边双联通分量：不含割边的极大联通分量。
边双联通分量是对点集的一个划分。
两个边双联通分量由一条割边连接。

实现方式：
在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。

6.2、Tarjan’s Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



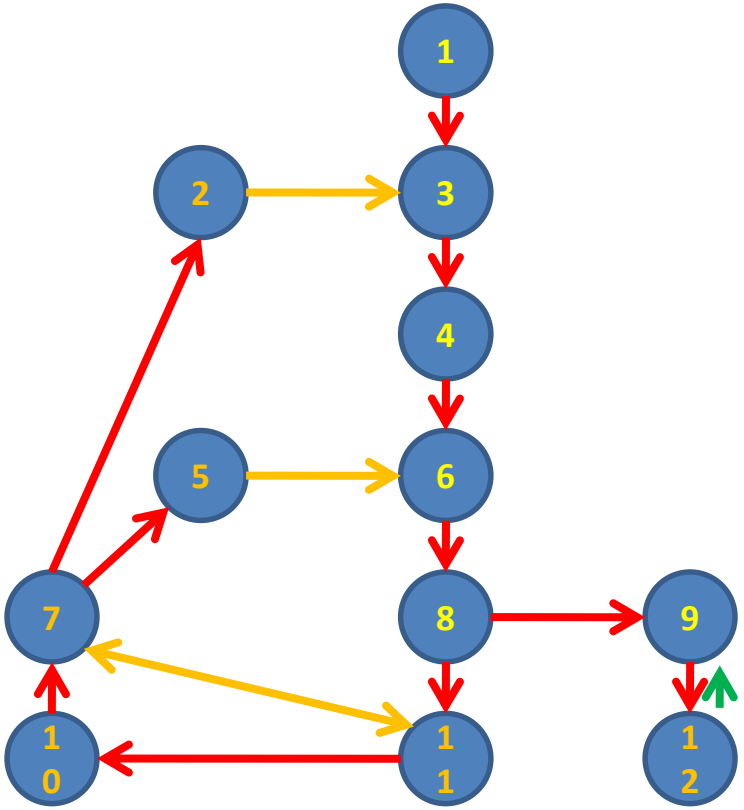
栈

1
3
4
6
8
11
10
7
5
2
8
9
12

6.2、Tarjan’s Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



栈

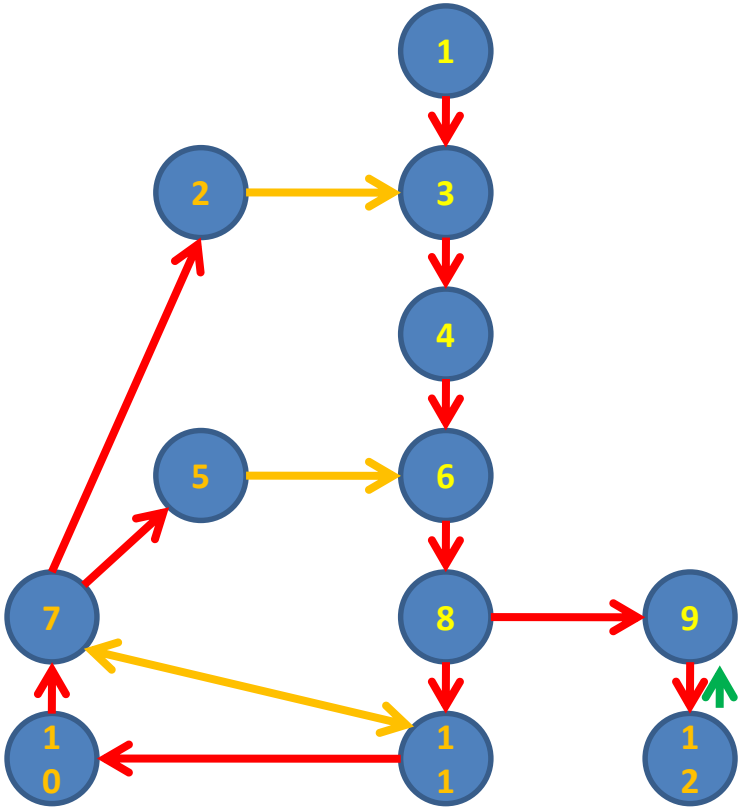
(9,12)是割边

1
3
4
6
8
11
10
7
5
2
8
9
12

6.2、Tarjan's Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



栈

1
3
4
6
8
11
10
7
5
2
8
9

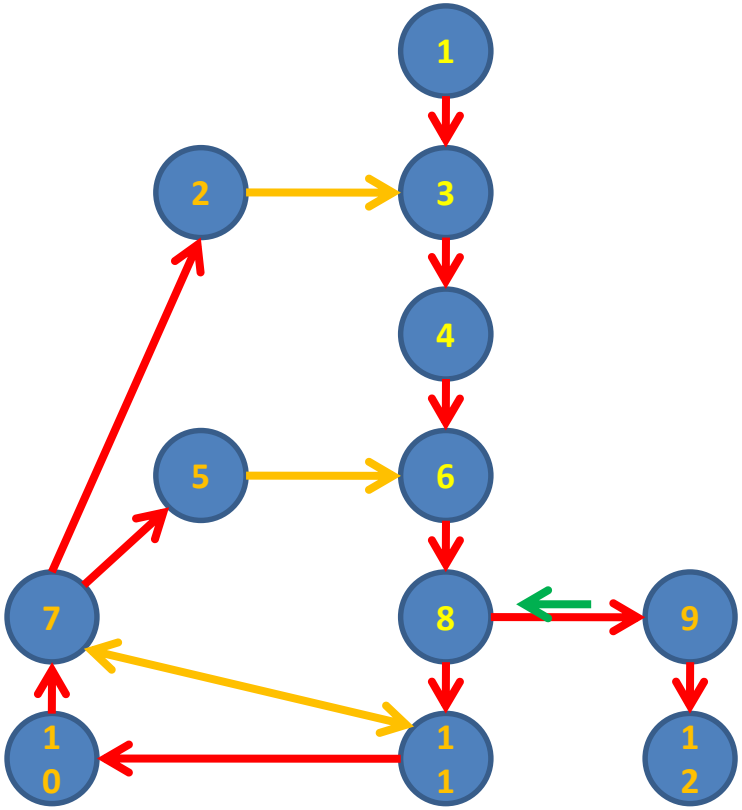
(9,12)是割边

12

6.2、Tarjan's Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



栈

1
3
4
6
8
11
10
7
5
2
8
9

(9,12)是割边

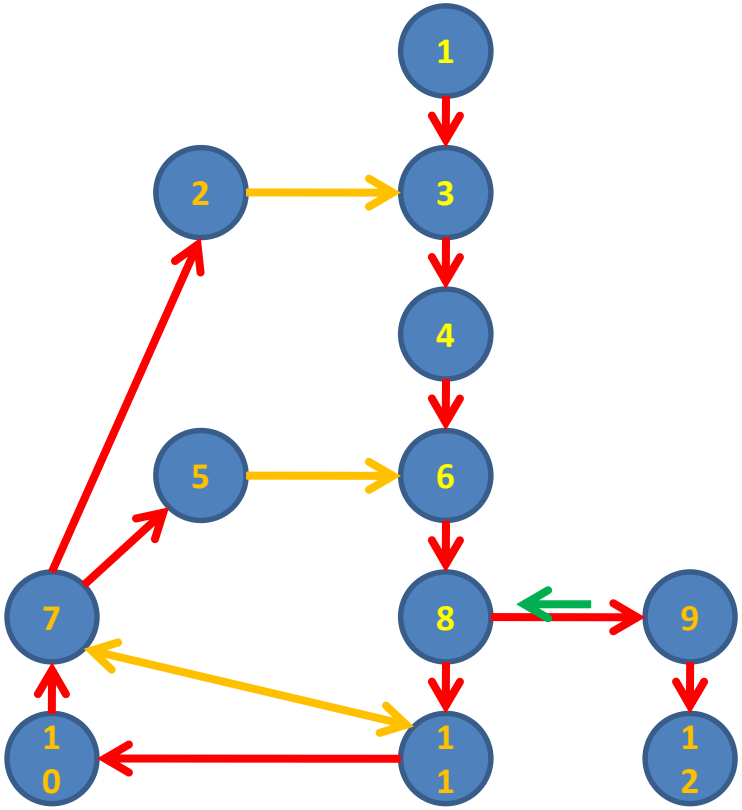
12

(8,9)是割边

6.2、Tarjan's Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



栈

1
3
4
6
8
11
10
7
5
2
8

(9,12)是割边

12

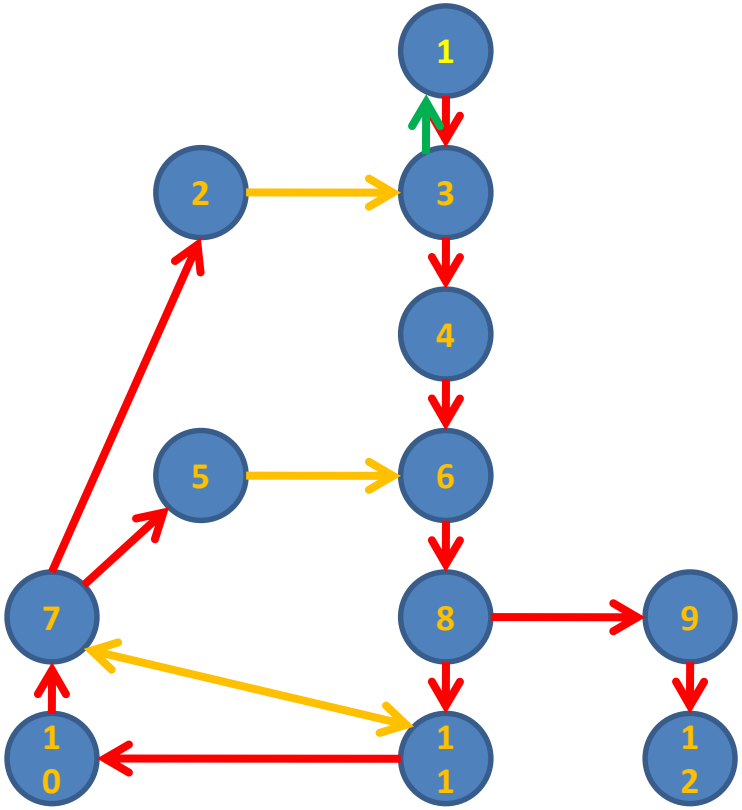
(8,9)是割边

9

6.2、Tarjan’s Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



栈

1
3
4
6
8
11
10
7
5
2
8

(9,12)是割边

12

(8,9)是割边

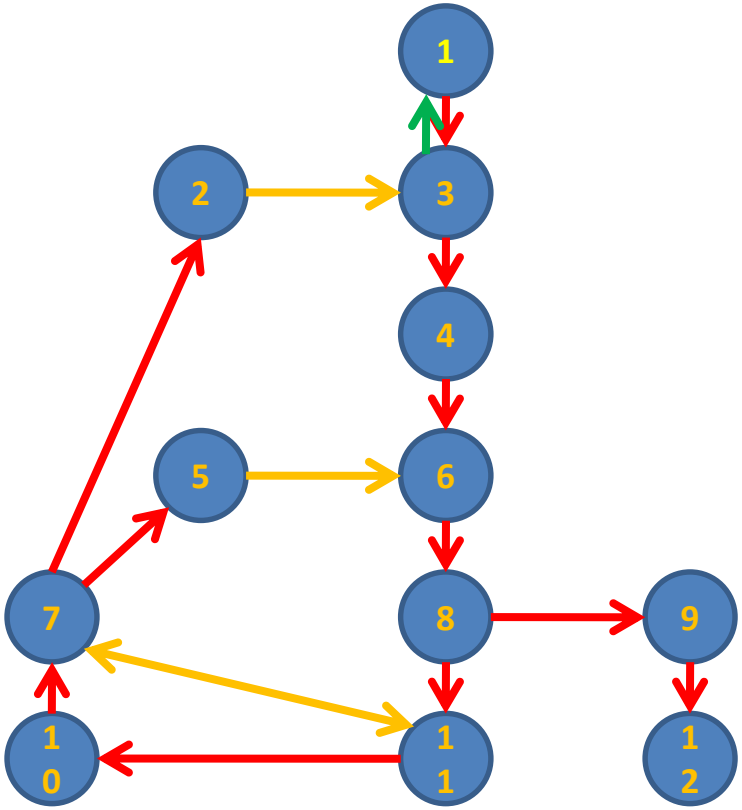
9

(1,3)是割边

6.2、Tarjan's Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。

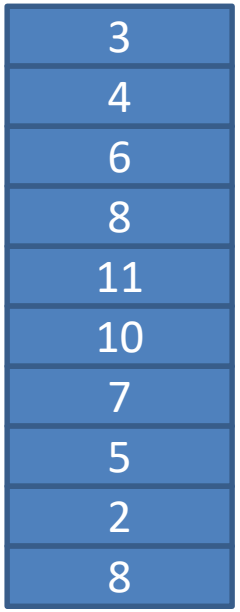


栈

(9,12)是割边

(8,9)是割边

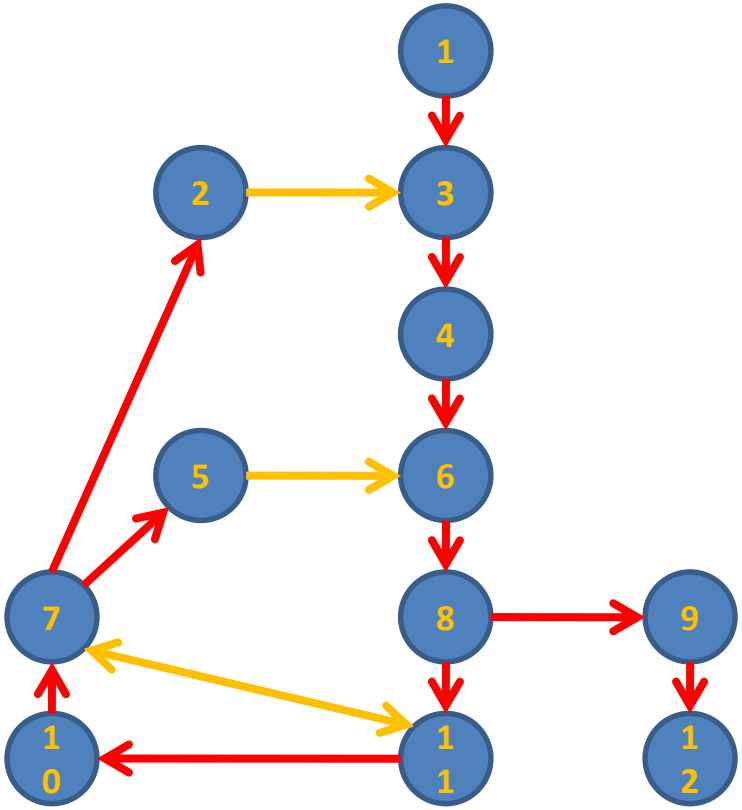
(1,3)是割边



6.2、Tarjan's Algorithm求边双连通分量

实现方式：

在求割边算法的基础上，每次搜索到一个点，就将其压入栈中。当发现一条割边时，排出栈顶元素直到排出相应的顶点。



栈

1

(9,12)是割边

12

(8,9)是割边

9

(1,3)是割边

3
4
6
8
11
10
7
5
2
8

6.2、Tarjan’s Algorithm求双连通分量算法复杂度

相比求割点、割边的算法，求双连通分量的算法只是增加了一个栈，每个元素至多进栈一次出栈一次。

因此
算法的时间复杂度依旧是**O(V+E)**
算法的空间复杂度依旧是**O(V+E)**