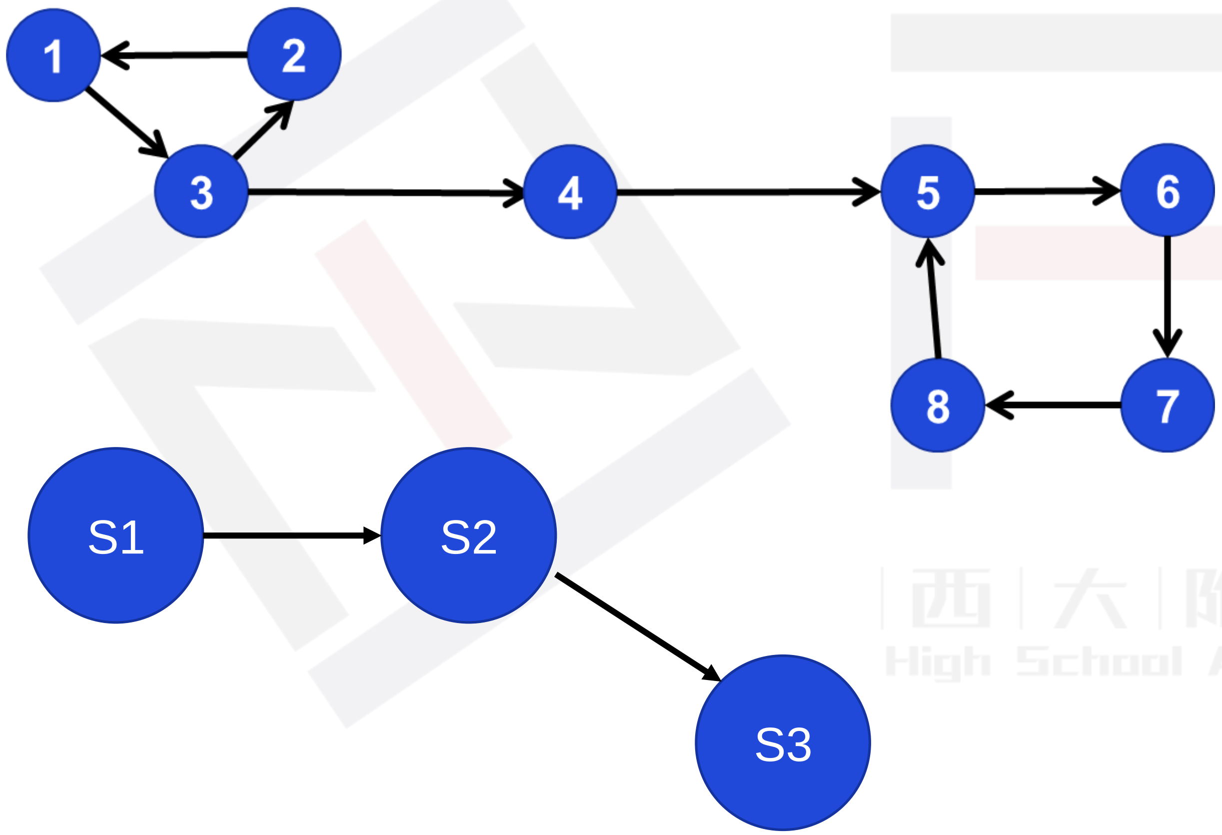
问题

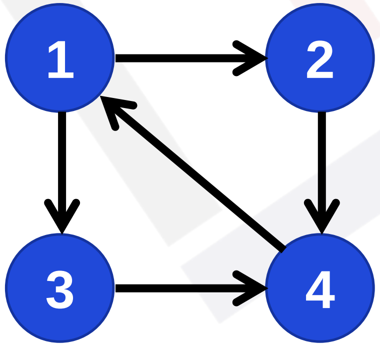


有些时候题目给出的图点非常多且复杂，我们希望能够不改变图整体结构的前提下简化图的模型，从而便于分析、解决问题

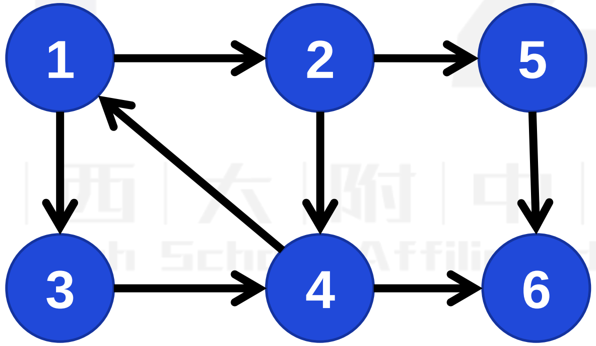
强连通分量(SCC)

- 在**有向图**中，如果两个顶点u，v存在一条u到v的路径且也存在v到u的路径，则称这两个顶点u，v**强连通**。
- 如果有向图的任意两个顶点都强连通，则称为**强连通图**。
- 有向**非**强连通图的极大强连通子图，称为**强连通分量**(Strongly Connected Components，**SCC**)。

一个强连通分量中的每两个点可以互相到达，且这个强连通分量所包含的的节点数尽可能大
结点数大于2的强联通分量一定存在环；特殊地，一个点也是强连通分量



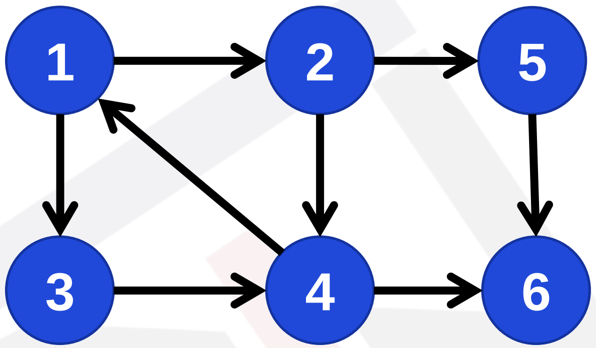
强连通图



非强连通图

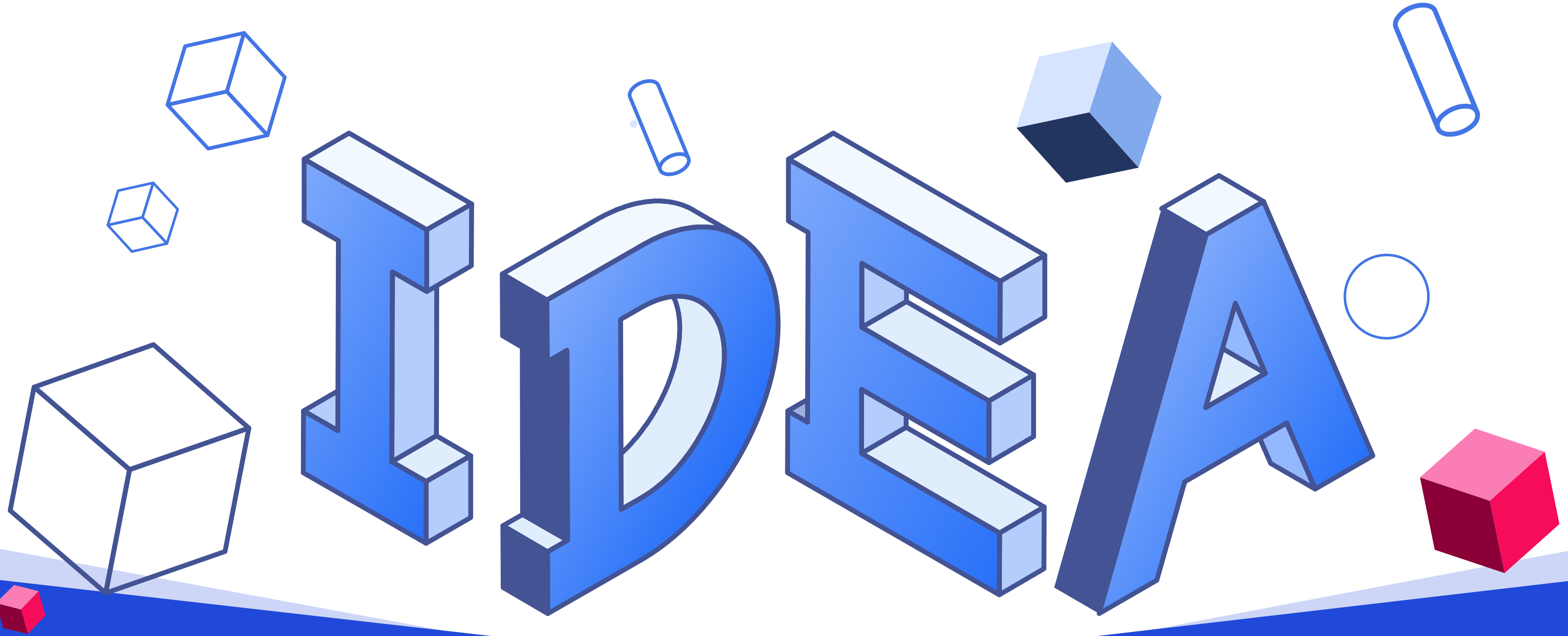
- {1, 2, 3, 4}，{5}，{6} 是强连通分量
- {1, 3, 4} 强连通，但不是极大强连通子图，所以不是强连通分量

求解SCC



问题：对于这样一张图，你猜想下可以怎么求SCC?

DFS?

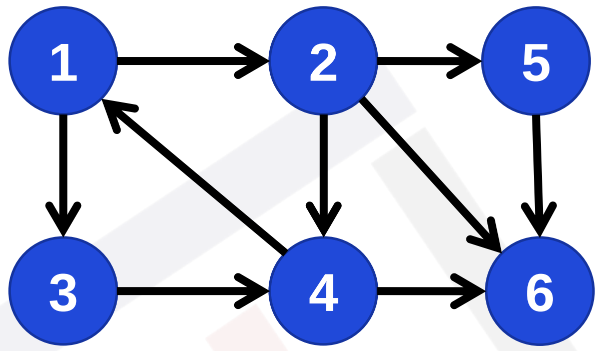


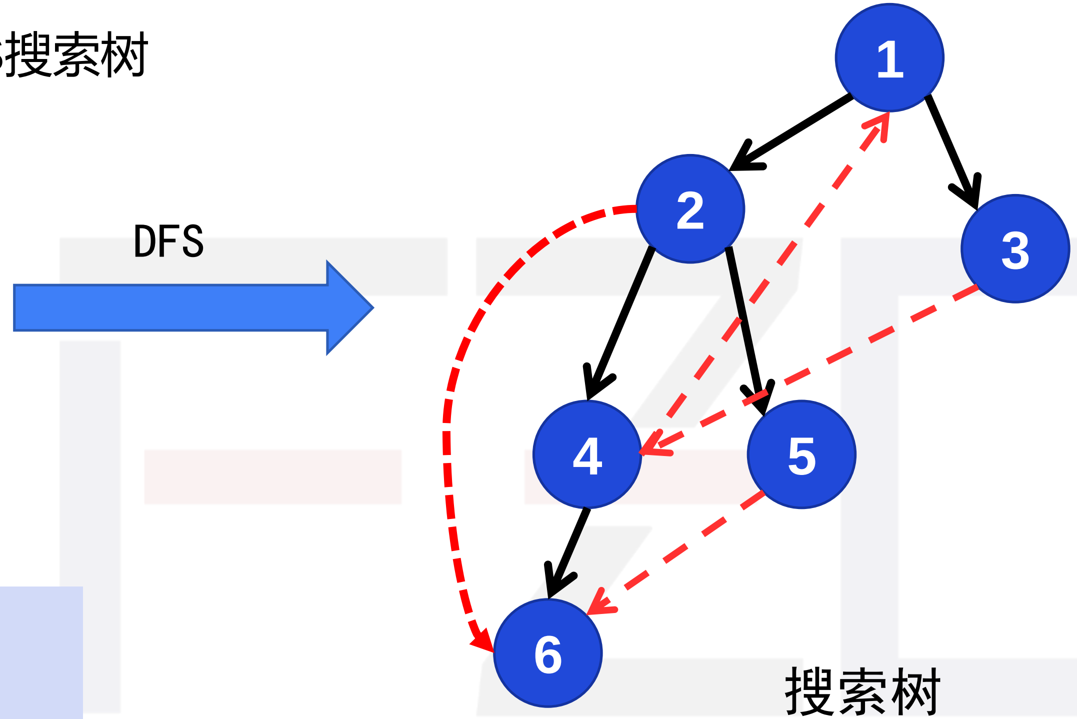
信息学暑期

Tarjan求强连通与缩点(有向图)

DFS搜索树

Tarjan算法求解SCC需要基于DFS搜索树





搜索树

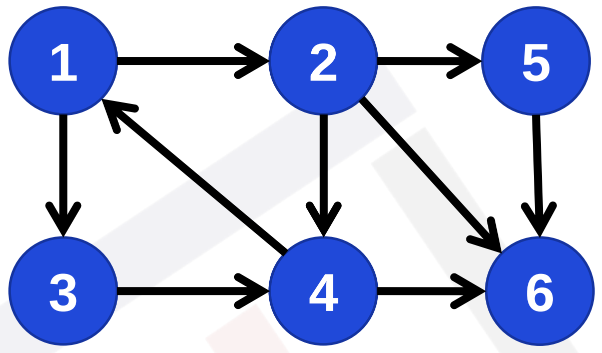
```
bool vis[MAXN];

void dfs(int u)
{
    vis[u] = true;
    for(int e = first[u]; e; e = nxt[e])
    {
        // 遍历连接u的每条边
        int v = go[e];
        if(!vis[v]) dfs(v);
        // 如果没有访问过就往下继续搜
    }
}
```

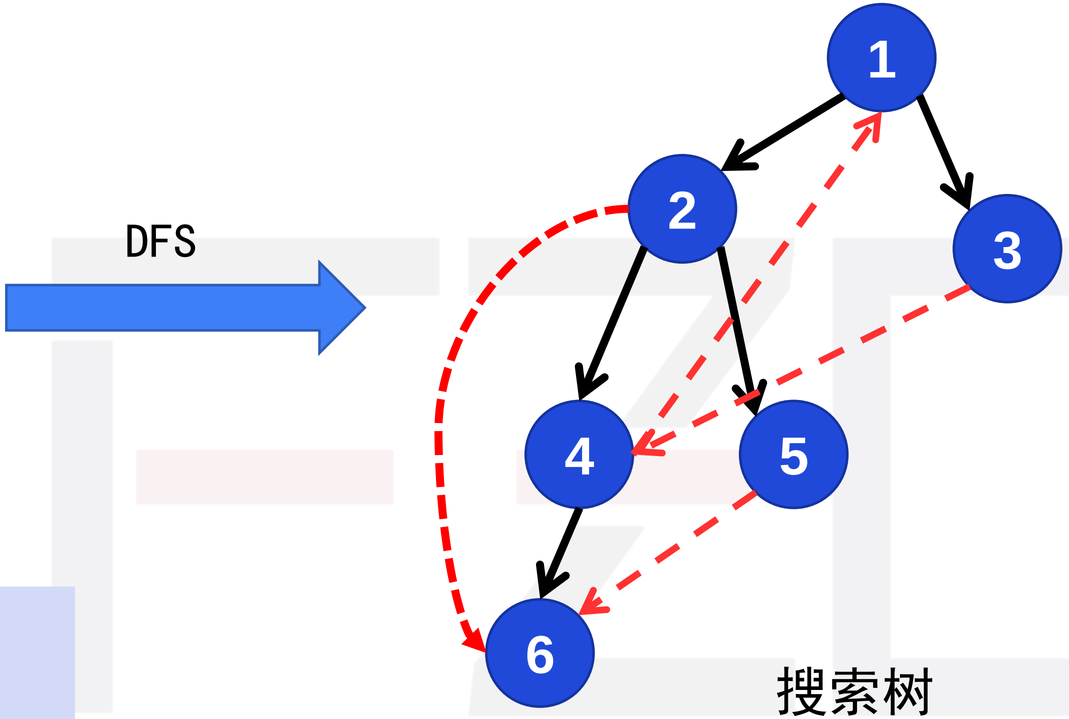
为了表示每个节点第一次访问的时间，我们先引入一个时间戳tot 其实也是dfs序

越先访问的节点时间戳越小，越后访问的节点时间戳越大

使用数组dfn来存储每个节点的时间戳，同时也起到标记的作用

DFS搜索树






```
bool vis[MAXN];

void dfs(int u)
{
    dfn[u] = ++tot;
    for(int e = first[u]; e; e = nxt[e])
    {
        // 遍历连接u的每条边
        int v = go[e];
        if(!dfn[v]) dfs(v);
        // 如果没有访问过就往下继续搜
    }
}
```

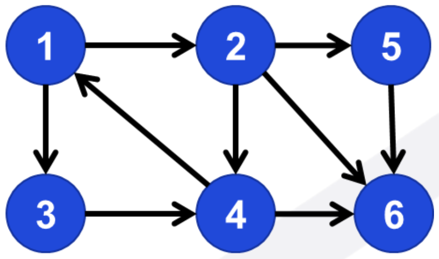
为了表示每个节点第一次访问的时间，我们先引入一个时间戳tot
其实也是dfs序

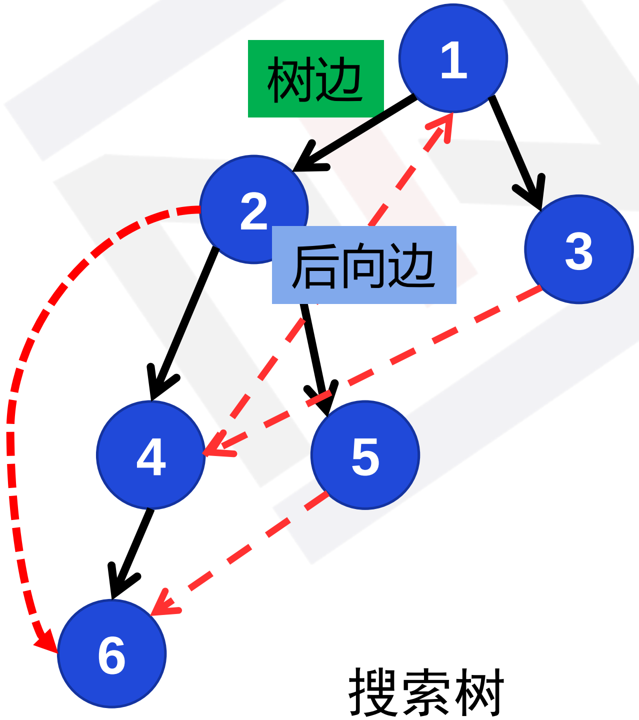
越先访问的节点时间戳越小，越后访问的节点时间戳越大

使用数组dfn来存储每个节点的时间戳，同时也起到标记是否访问的作用

搜索树的几种边

寻找强连通分量之前，需要dfs树上的了解几种边





搜索树

1.树边

指dfs树上的边，dfs正常访问时产生的

```
if(!dfn[v]) dfs(v);
```

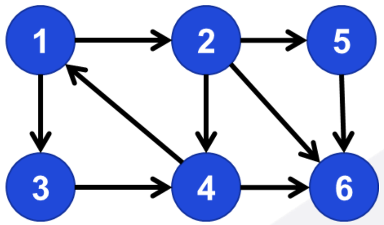
2.后向边

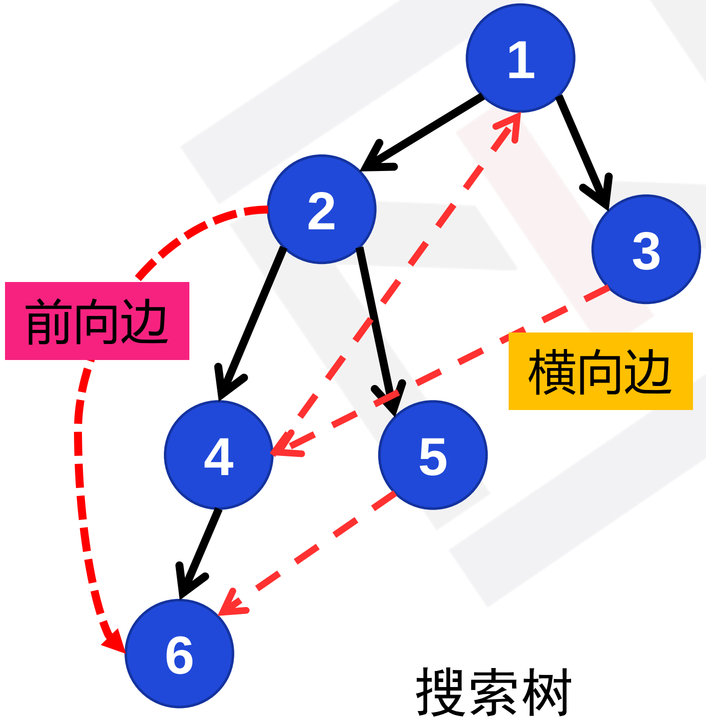
是指将节点u连接到其在深度优先搜索树中的祖先节点v的边<u,v>

```
dfn[v] != 0
&&
dfn[v] <= dfn[u]
```

搜索树的几种边

寻找强连通分量之前，需要dfs树上的了解几种边





搜索树

3.前向边

节点u连接到其在深度优先搜索树中的后代节点v的边<u,v>

```
dfn[v] != 0
&&
dfn[v] > dfn[u]
```

即：v被访问过，且v比u后被访问。

4.横向边

对于<u,v>，u不是v的祖先。

```
dfn[v] < dfn[u]
```

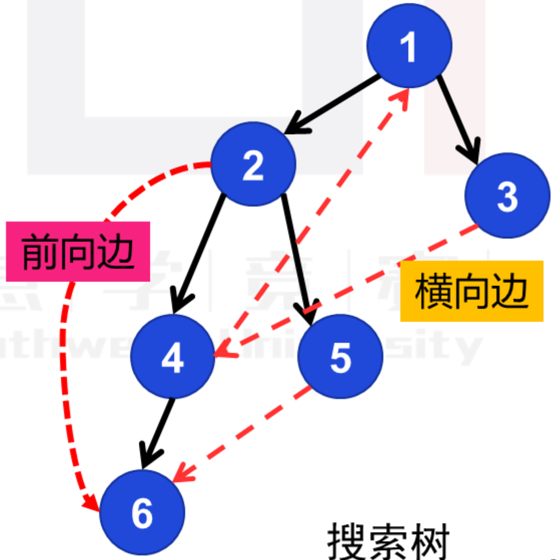
树边的几个性质

性质1： 横向边u→v，满足dfn[u]>dfn[v]。

证明：

根据深度优先搜索的策略，访问到结点u之后，接下来会访问它所有邻接的未被访问的结点，u到所有这些结点的边都是树边。

因为此处u→v不是树边，而是横向边，所以在访问u时v一定已被访问过。根据dfn[ ]随访问顺序严格单调递增，显然有dfn[u]>dfn[v]。



搜索树

树边的几个性质

性质2： 若有横向边u→v，则必不存在从v到u的路径。

证明：

反证法。

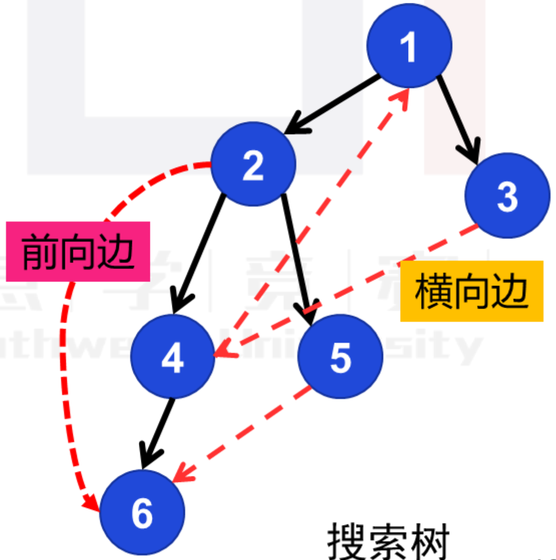
假设存在从v到u的路径，则该路径必是深度优先搜索树上的一条链。

因为在访问节点v时，该路径上的点都没有被访问过。因此根据前面的代码，访问v时必会继续搜索这条路径上的点，这条路径上的每条边必是树边。这样，v就是u的祖先了，u→v就是后向边，与u→v是横向边矛盾。因此假设不成立，不存在从v到u的路径。

证毕。

意义：

若有横向边u→v，则u、v必不在同一个强连通分量中。
所以tarjan算法求解强连通时可以忽略横向边



搜索树

性质3 ：若存在后向边u→v，则u、v在同一个强连通分量中。

证明：

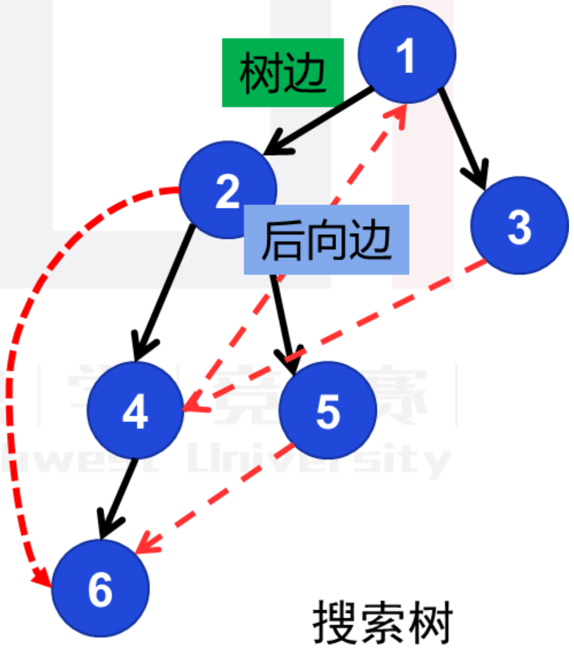
由u→v知道：v是u的祖先节点，所以路径v→u存在， 且是深度优先搜索树上的一条链。

故v→u→v构成一个环。

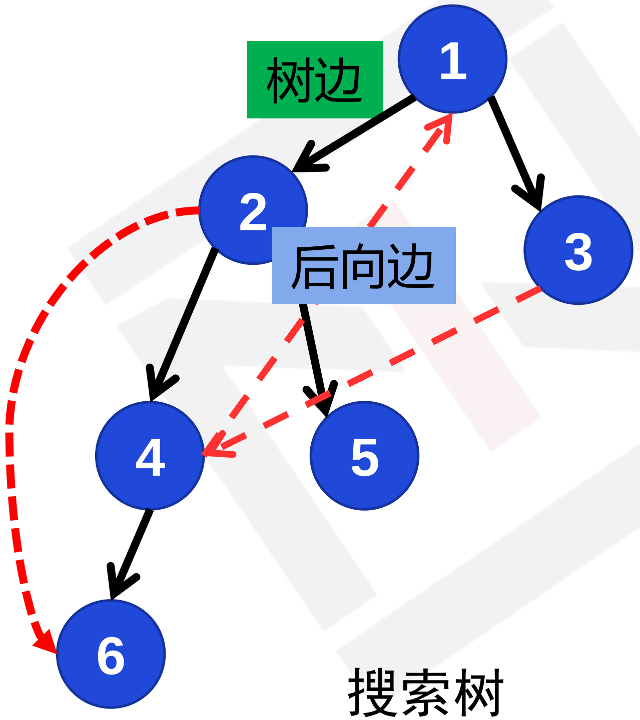
因此u、v在同一个强连通分量中。

意义：

表明后向边是构成强连通分量的关键因素。
但对于前向边u→v而言，其发挥的作用和树边是相同的
不管走树边还是前向边，都可以从u到v



搜索树



搜索树

1.树边

指dfs树上的边，dfs正常访问时产生的

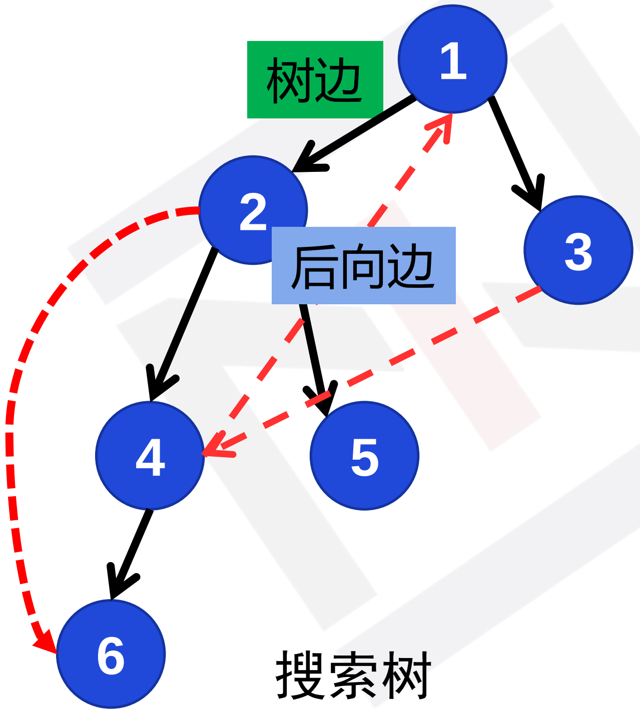
```
if(!dfn[v]) dfs(v);
```

2.后向边

是指将节点u连接到其在深度优先搜索树中的祖先节点v的边u→v

```
dfn[v] != 0 && dfn[v] <= dfn[u]
```

Tarjan算法求解时，只考虑树边和后向边



搜索树

1.树边

指dfs树上的边，dfs正常访问时产生的

```
if(!dfn[v]) dfs(v);
```

2.后向边

是指将节点u连接到其在深度优先搜索树中的祖先节点v的边u→v

```
dfn[v] != 0 && dfn[v] <= dfn[u]
```

但有个问题：
如何判断是后向边?

Tarjan-栈stk

我们考虑维护一个栈stk：栈中的元素是当前搜索树上的点。
显然，如果一条边u→v是后向边，那么当我们在访问u时会发现v已经在栈中。
如果dfn[v]<dfn[u]，则u→v是后向边。
如何判断一个数是否在栈中?
我们定义instack[]数组，节点u入栈时instack[u]=true，出栈时instack[u]=false

```
int dfn[MAXN], tot = 0;
bool instack[MAXN]; // 可以考虑用bitset<MAXN>
stack<int> stk;

void dfs(int u){
    dfn[u] = ++tot;
    stk.push(u);
    instack[u] = true;
    for(int e = first[u]; e; e = nxt[e]){
        int v = go[e];
        if(!dfn[v]) dfs(v);
        else if(instack[v]) // v访问过且在栈中，意味着u→v是后向边
        {
            //处理
        }
    }
    stk.pop();
    instack[u] = false;
}
```

知道u→v是后向边之后，我们要做什么呢?

根据刚才的性质3，我们知道
后向边的两个点同属于一个SCC，
那么我们希望能把SCC的所有点都能找出来

此时，我们希望用一种方法标明，栈中的元素，从v到u，都属于同一个SCC。

核心：引入low[]数组

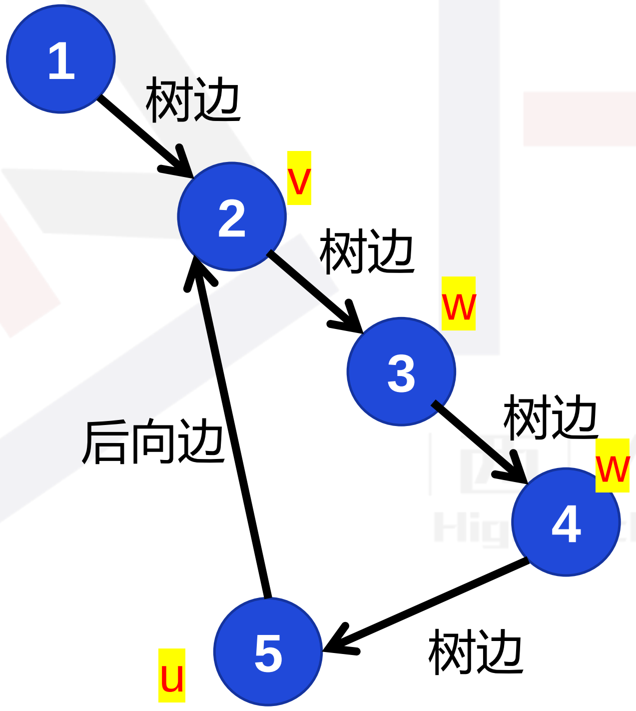
Tarjan-low数组

引入low[]数组

low[u]：包含u的SCC中第一个被搜索到的节点的dfn值，也可以理解为从u出发能回溯到的dfn最小的节点的dfn值。

显然，若u→v是一个后向边，那么v是u的祖先，v是v、u所在的SCC中最先被访问到的节点，low[u]=dfn[v]。而且，对于v→u路径（都是树边）上的每一个节点w，有low[w]=dfn[v]，因为w和v、u属于同一个SCC。

举个简单粒子：

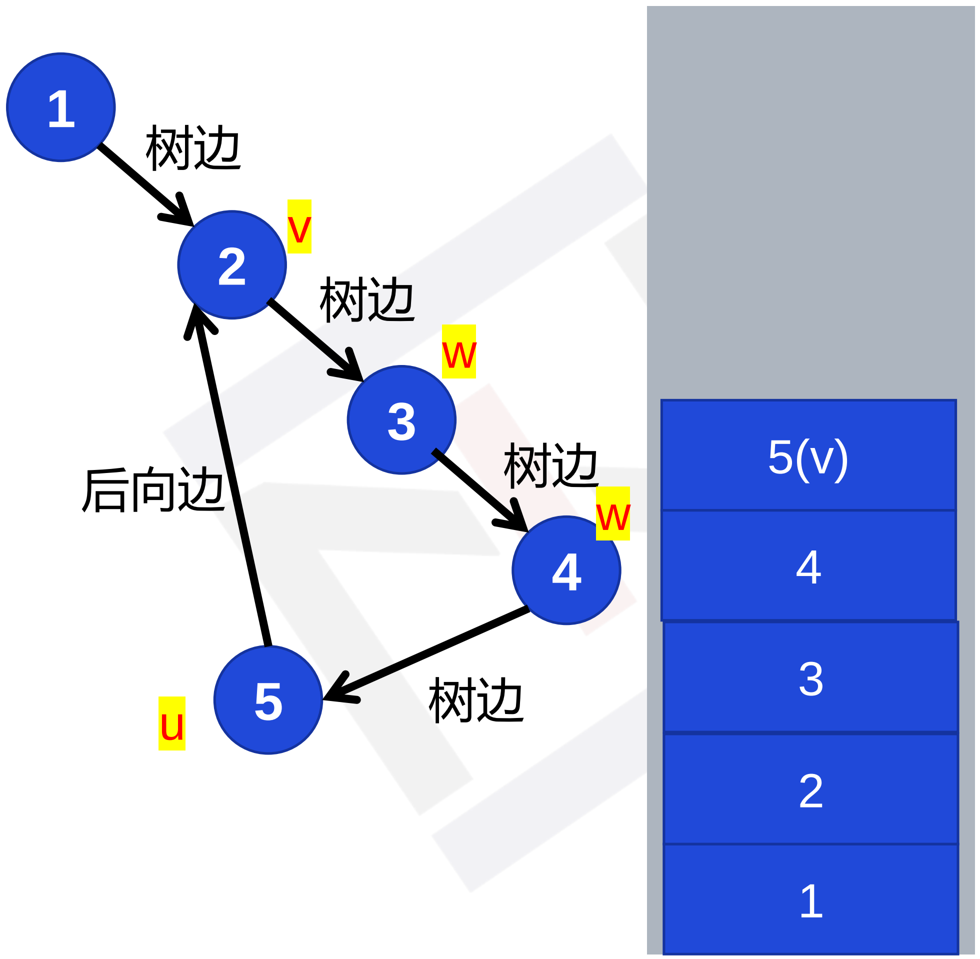


{2，3，4，5}属于一个SCC

它们每个点的low值都应该是dfn[v]， dfn[v] =dfn[2]=2

问题来了：以何种方式更新low数组呢?

Tarjan-low数组



可不可以把栈中压在v以上的元素的low值全部改为dfn[v]?
其实在回溯的时候，设当前节点为u，子节点为v，则执行
low[u] = min(low[u], low[v])。

为什么要用low[u]=min(low[u], low[v])，而不是直接
low[u]=low[v]呢?

因为若low[v]=dfn[v]，low[u]=dfn[u]，则可能
low[u]<low[v]（u比v先被访问），所以取二者较小的

Tarjan-low数组更新代码


```
int dfn[MAXN], tot = 0;
bool instack[MAXN];
int low[MAXN];
stack<int> stk;

void dfs(int u){
    dfn[u] = ++tot;
    low[u] = dfn[u]; // 一开始low[u]是自己，有后向边再更新
    stk.push(u);
    instack[u] = true;
    for(int e = first[u]; e; e = nxt[e]){
        int v = go[e];
        if(!dfn[v])
        {
            dfs(v);
            low[u] = min(low[u], low[v]); // 子节点更新了，父节点也要更新
            // 若子节点没更新，则min能够保证low[u] == dfn[u]
        }
        else if(instack[v]) // v访问过且在栈中，意味着u→v是后向边
        {
            low[u] = min(low[u], dfn[v]);
            // 此处用min的原因是u→v可能是前向边，此时dfn[v]>dfn[u]
        }
    }
    stk.pop();
    instack[u] = false;
}
```

问题：什么时候能得到强连通分量?

如果结点u的子树全部搜索完，有：

dfn[u]==low[u]

Tarjan算法过程

（1）依次搜索结点u，low[u]=dfn[u],dfn为时间戳，即搜索次序，将结点u入栈；

（2）搜索u出发的每条边(u,v)：

如果v未访问，说明(u,v)是树枝边，继续访问v：

tarjan(v); low[u]=min(low[u],low[v]);

如果v访问过且在栈中，说明(u,v)是后向边，则：

low[u]=min(low[u],dfn[v]);

（3）如果结点u的子树全部搜索完，dfn[u]==low[u]，则u到栈顶的所以结点构成一个强连通分量，出栈。

（4）继续搜索，直到所以结点遍历结束。 时间复杂度：O(N+M)，每个点出栈一次，进栈一次，每条边被访问一次。

Tarjan完整代码


```
void Tarjan(int u)
{
    dfn[u] = ++tot;
    low[u] = dfn[u]; // 一开始low[u]是自己，有后向边再更新
    stk.push(u);
    instack[u] = true;
    for(int e = first[u]; e; e = nxt[e])
    {
        int v = go[e];
        if(!dfn[v])
        {
            Tarjan(v);
            low[u] = min(low[u], low[v]); // 子节点更新了，父节点也要更新
            // 若子节点没更新，则min能够保证low[u] == dfn[u]
        }
        else if(instack[v]) // v访问过且在栈中，意味着u→v是后向边
        {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if(low[u] == dfn[u]) // 是SCC中的第一个被访问的节点
    {
        color[u] = ++col;
        while(stk.top() != u) {color[stk.top()] = col, instack[stk.top()]
= false, stk.pop();}// 染色，弹栈
        instack[u] = false;
        stk.pop(); // 最后把u弹出去
    }
}
```

注意更换搜索起点，这个有向图可能分为多个不连通的部分：

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

问题：怎么标记每一个强连通分量呢?

给每个节点“染色”，在同一个SCC中的节点拥有相同的颜色。

`color[]/scc[]`

当我们发现low[u] == dfn[u]时，代表u是其所在的SCC的最先访问到的节点。
此时，栈中压在u以上的所有元素，包括u，构成一个SCC。
然后将u即压在它上面的所有元素的颜色标记为++col，并弹出。

小结

Tarjan算法求解SCC需要四个数组：

dfn[u]: dfs number，为结点u的搜索的时间戳(搜索次序)

low[u]: 结点 u 或者 u 的子树在栈中最多能够回溯（到达）到的结点 x，使得dfn[u]最小

stk: 每访问一个点就把它压入栈中，这是为了之后找一个强连通分量里所有的子节点。

color[]:染色，即标记每个点所属的SCC编号

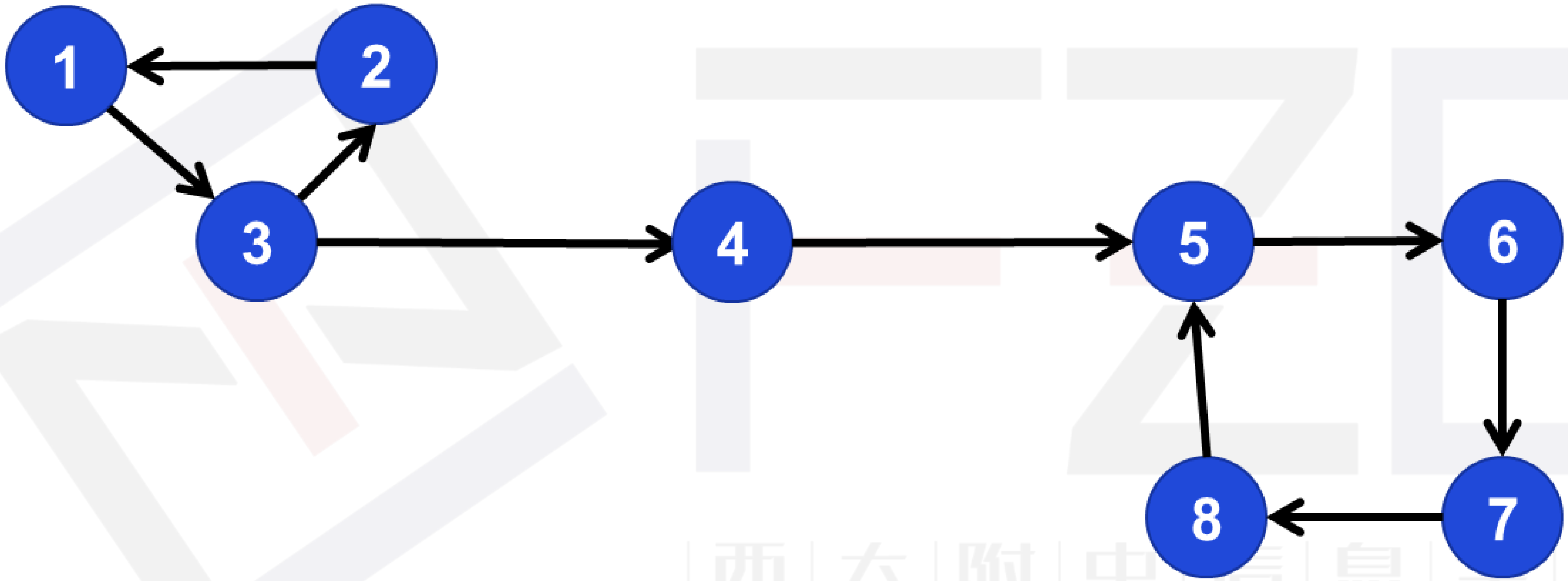
Tarjan算法求解需要关注两种边：

树边：指dfs树上的边

后向边：指将节点u连接到其在深度优先搜索树中的祖先节点v的边u→v

练一练

从1开始求解SCC，假设4比2先访问
拿出草稿纸，请标记出每个点最终的dfn，low



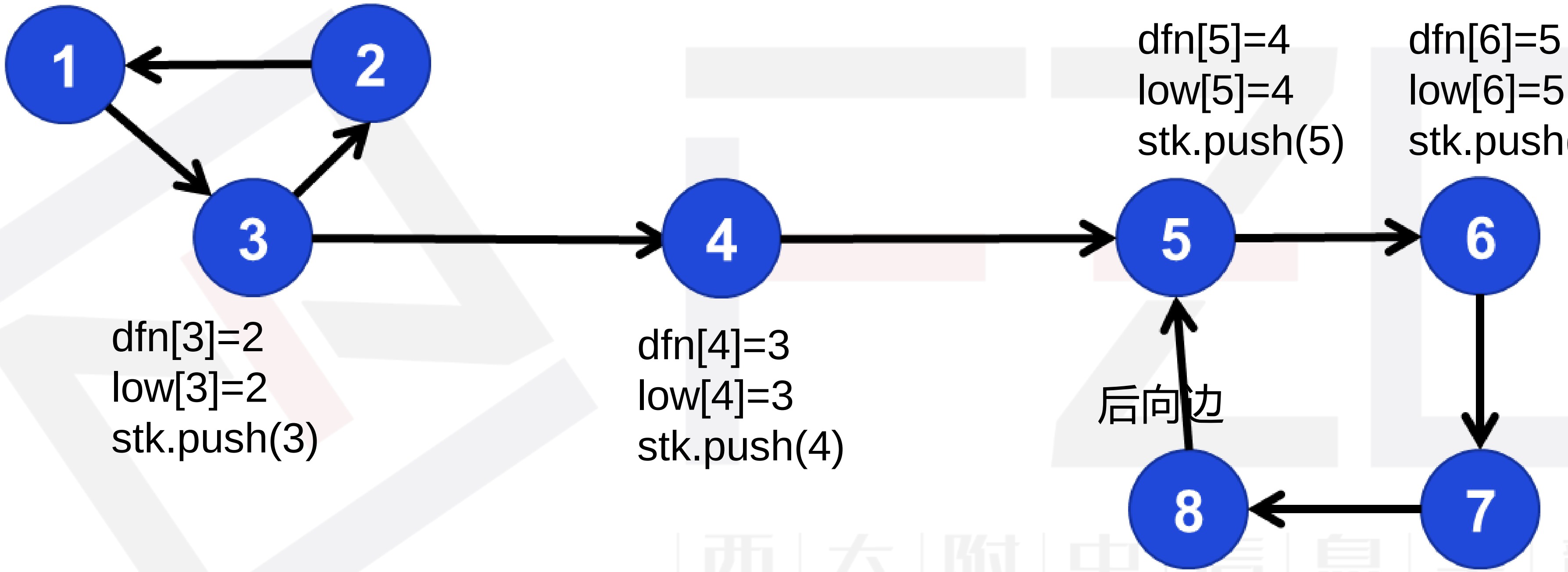
练一练

回溯，使得
low[6]=low[7]=4

dfn[1]=1
low[1]=1
stk.push(1)

dfn[5]=4
low[5]=4
stk.push(5)

dfn[6]=5
low[6]=5
stk.push(6)



dfn[3]=2
low[3]=2
stk.push(3)

dfn[4]=3
low[4]=3
stk.push(4)

更新low[8]=dfn[5]=4

dfn[7]=6
low[7]=6
stk.push(6)

执行第一次出栈
{8，7，6，5}，为第一个SCC

回溯，使得low[7]=low[8]=4

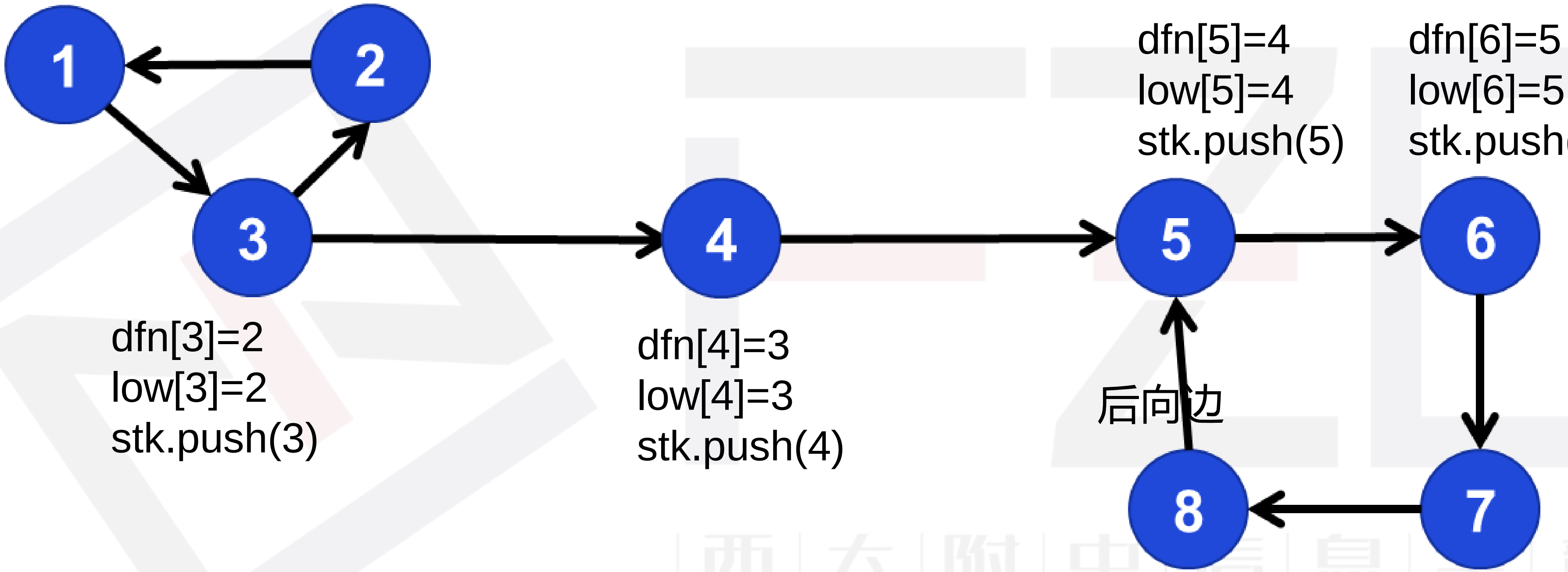
练一练

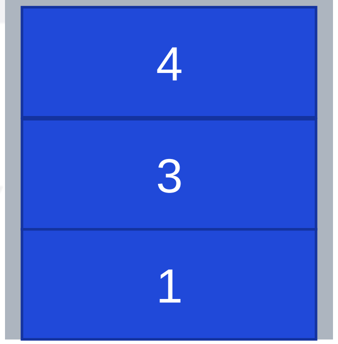
回溯，使得
low[6]=low[7]=4

dfn[1]=1
low[1]=1
stk.push(1)

dfn[5]=4
low[5]=4
stk.push(5)

dfn[6]=5
low[6]=5
stk.push(6)





dfn[3]=2
low[3]=2
stk.push(3)

dfn[4]=3
low[4]=3
stk.push(4)

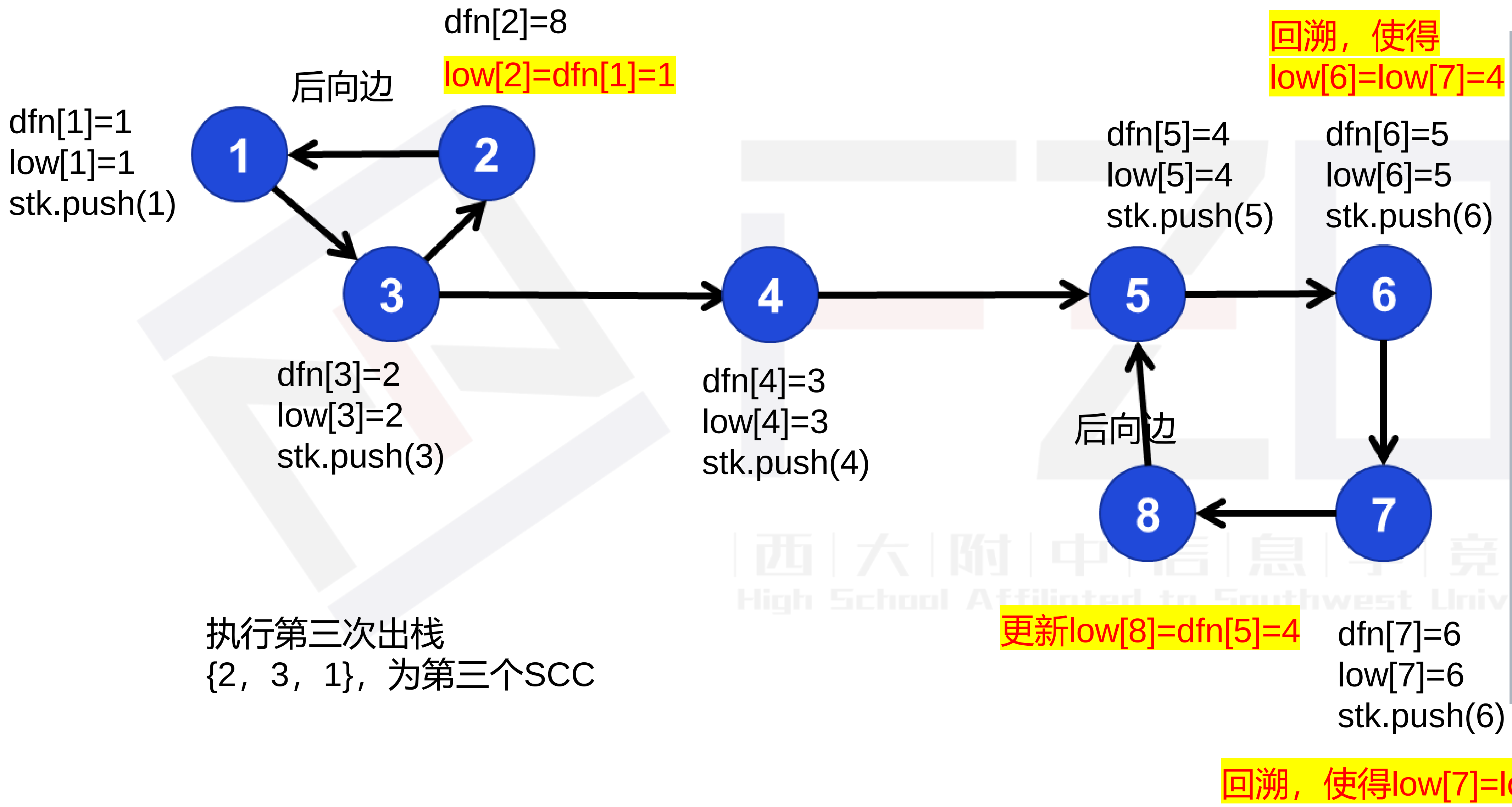
执行第二次出栈
{4}，为第二个SCC

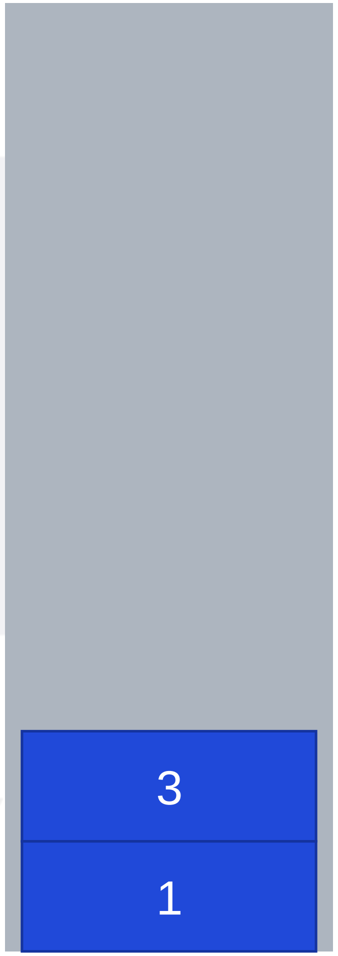
更新low[8]=dfn[5]=4

dfn[7]=6
low[7]=6
stk.push(6)

回溯，使得low[7]=low[8]=4

练一练



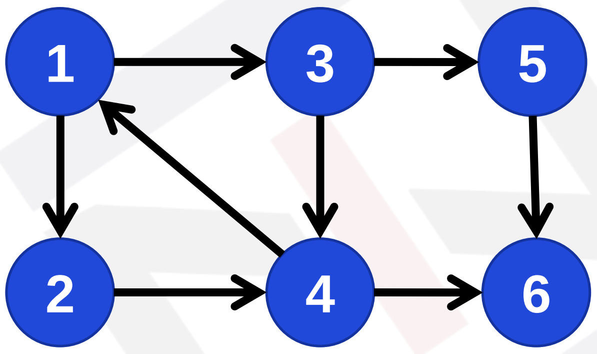


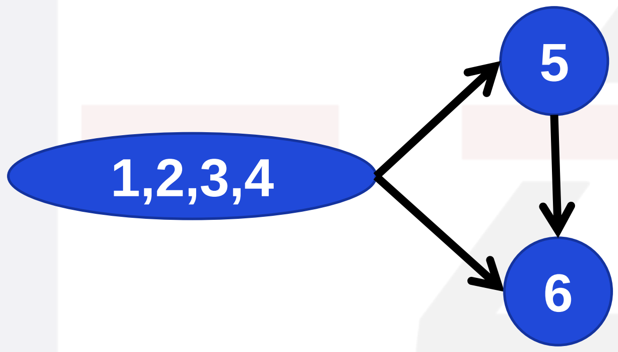
执行第三次出栈
{2，3，1}，为第三个SCC

缩点

将**每一个强连通分量看作一个点**，可以极大的简化图，即有向无环图(DAG图)。

对于DAG图，我们可以进行拓扑排序、DAG上DP等。





缩点


```
void Tarjan(int u)
{
    dfn[u] = ++tot;
    low[u] = dfn[u]; // 一开始low[u]是自己，有后向边再更新
    stk.push(u);
    instack[u] = true;
    for(int e = first[u]; e; e = nxt[e])
    {
        int v = go[e];
        if(!dfn[v])
        {
            Tarjan(v);
            low[u] = min(low[u], low[v]); // 子节点更新了，我也要更新
            // 若子节点没更新，则min能够保证low[u] == dfn[u]
        }
        else if(instack[v]) // v访问过且在栈中，意味着u→v是后向边
        {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if(low[u] == dfn[u]) // 是SCC中的第一个被访问的节点
    {
        scc[u] = ++col;
        while(stk.top() != u) {scc[stk.top()] = col, instack[stk.top()] =
false, stk.pop();}// 染色，弹栈
        instack[u] = false;
        stk.pop(); // 最后把u弹出去
    }
}
```

原点的编号：
i
原来的边：
i->j

新点的编号：
scc[i]
新的边：
scc[i]->scc[j]

Thanks

For Your Watching

FZOI

西大附中信息学竞赛
High School Affiliated to Southwest University