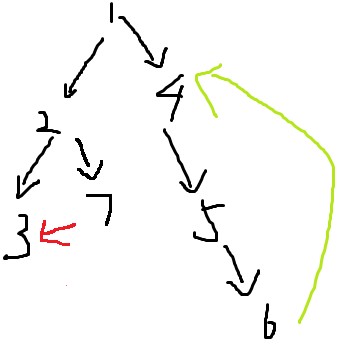
# Tarjan

[**强连通分量及缩点tarjan算法解析**](http://blog.csdn.net/acmmmm/article/details/16361033)



强连通定义：在有向图G<V,E>中，对于点集V'∈V, 点集中的任意两点都可达，则称V'为强连通。

孤立的一个点也是一个强连通分量

在嵌套的多个环时 : {所有环上的点}为一个强连通分量( 最小环就是每个孤立点）注意一定是满足条件的最大点集。

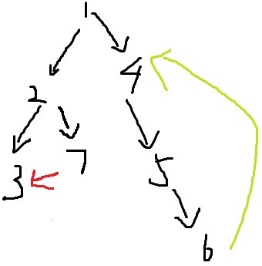
 则上图中强连通分量有 {1},{2},{3},{7},{4,5,6}

---------------------------------------------------------------------------

tarjan的过程就是dfs过程

对图dfs一下，遍历所有未遍历过的点 ，会得到一个[有向树](http://baike.baidu.com/link?url=CkM7EiHiG9MSdQcAKwF4oReg83XHA1pz7BtrS4WWhqBjfhWRvAwCGaqlNIJ_d5ssAS1NgpMIDbwkNx_IPIsYpa)，显然有向树是没有环的。（注意搜过的点不会再搜）

则能产生环的 只有（指向已经遍历过的点）的边

如左图，只有红色与绿色边有可能产生环。

对于深搜过程，我们需要一个栈来保存当前所在路径上的所有点（栈中所有点一定是有父子关系的）

再仔细观察红边与绿边，首先得到结论：红边不产生环，绿边产生环

1、对于红边，连接的两个点3、7没有父子关系，这种边称为横叉边。

横叉边一定不产生环。

2、对于绿边，连接的两个点6、4是父子关系，这种边称为后向边。

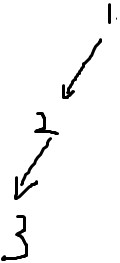
环一定由后向边产生。

3、图中除了黑色的树枝边，一定只有横叉边和后向边（不存在其他种类的边）

-------------------------------------------------------------------------

则以下考虑对于这两种边的处理和判断：

首先深搜会搜到这样的图：

Stack = {1,2,3},3没有多余的其他边，因此3退栈，把3作为一个强连通分量

-------------------------------------------------------------------------

再次深搜：

此时栈 Stack = {1,2,7}

发现红边指向了已经遍历过的点3 => 是上述的2种边之一

而3不在栈中 => 3点与7点无父子关系

=> 该边为横叉边

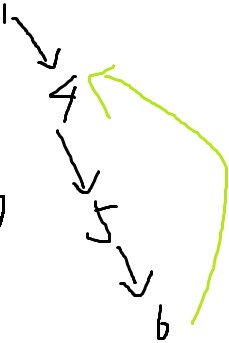
=>采取无视法。

继而7点退栈 产生连通分量{7}

继而2点退栈 产生连通分量{2}

--------------------------------------------------------------------------------------

再次深搜：

此时 Stack = {1,4,5,6}

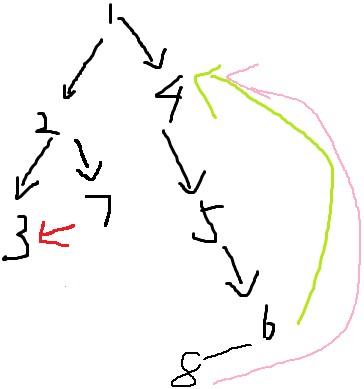
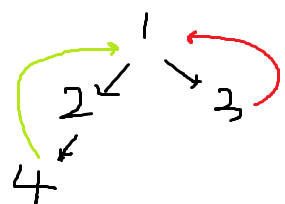
发现绿边指向了已经遍历过的点4 => 是上述的2种边之一

而4在栈中 => 4点与6点是父子关系

=> 该边为后向边

=>4->6的路径上的点都是环。

实际情况可能更加复杂（如上图），它有可能大环套小环，也有可能双环

我们对每一个点保存dfn[u]和low[u]

**定义dfn[u]**为u在dfs搜索树（以下简称为树）中被遍历到的次序号。

**定义low[u]**为u或u的子树中能通过非父子边追溯到的最早的节点，即dfn[]最小的节点。

我们只有当dfn[u]==low[u]的时候才说明它是一个一个环的根结点，也可以说是环中离根最近的点（有可能这个环就它自己一个点）

所以左图会忽略的4 5 6这个小环

右图会把1 2 3 4看作一个8字型大环

/\*

时间复杂度为O(n+m)

1、先最初调用init()

2、把图用add 存下来，注意图点标为1-n，若是[0,n-1]则给所有点++;

3、调用tarjan\_init(); 再调用tarjan\_cnt();

4、新图就是vector<int>G[]; 新图点标从1-tar ;

5、对于原图中的每个点u，都属于新图中的一个新点Belong[u];

新图一定是森林。

6、新图中的点u 所表示的环对应原图中的vector<int> bcc[u];

7、旧图中u在新图中所属的点是Belong[u];

8. G存的是新图，点的标号与原图一致

\*/

#define N 30100

//N为最大点数

#define M 150100

//M为最大边数

int n, m;//n m 为点数和边数

struct Edge{

int from, to, nex;

bool sign;//是否为桥

}edge[M<<1];

int head[N], edgenum;

void add(int u, int v){//边的起点和终点

Edge E={u, v, head[u], false};

edge[edgenum] = E;

head[u] = edgenum++;

}

int DFN[N], Low[N], Stack[N], top, Time; //Low[u]是点集{u点及以u点为根的子树} 中(所有反向弧)能指向的(离根最近的祖先v) 的DFN[v]值(即v点时间戳)

int taj;//连通分支标号，从1开始

int Belong[N];//Belong[i] 表示i点属于的连通分支

bool Instack[N];

vector<int> bcc[N]; //标号从1开始

void tarjan(int u ,int fa){

/// printf("abc%d %d\n",u,fa);

DFN[u] = Low[u] = ++ Time ;

Stack[top ++ ] = u ;

Instack[u] = 1 ;

for (int i = head[u] ; ~i ; i = edge[i].nex ){

int v = edge[i].to ;

if(DFN[v] == -1)

{

tarjan(v , u) ;

Low[u] = min(Low[u] ,Low[v]) ;

if(DFN[u] < Low[v])

{

edge[i].sign = 1;//为割桥

}

}

else if(Instack[v]) Low[u] = min(Low[u] ,DFN[v]) ;

}

if(Low[u] == DFN[u]){

int now;

taj ++ ; bcc[taj].clear();

do{

now = Stack[-- top] ;

Instack[now] = 0 ;

Belong [now] = taj ;

bcc[taj].push\_back(now);

}while(now != u) ;

}

}

void tarjan\_init(){

memset(DFN, -1, sizeof(DFN));

memset(Instack, 0, sizeof(Instack));

top = Time = taj = 0;

for(int i=1;i<=n;i++)if(DFN[i]==-1 )tarjan(i, i); //注意开始点标！！！

}

vector<int>G[N];//新图

int du[N];

void tarjan\_cnt(){

memset(du, 0, sizeof(du));

for(int i = 1; i <= taj; i++)G[i].clear();

for(int i = 0; i < edgenum; i++){

int u = Belong[edge[i].from], v = Belong[edge[i].to];

if(u!=v)G[u].push\_back(v), du[v]++;

}

}

void init(){memset(head, -1, sizeof(head)); edgenum=0;}