**Tarjan三大算法之双连通分量（双连通分量）**

**定义：**   
对于一个连通图，如果任意两点至少存在两条点不重复路径，则称这个图为**点双连通**的（简称双连通）；如果任意两点至少存在两条边不重复路径，则称该图为**边双连通**的。点双连通图的定义等价于任意两条边都同在一个简单环中，而边双连通图的定义等价于任意一条边至少在一个简单环中。对一个无向图，点双连通的极大子图称为**点双连通分量**（简称双连通分量），边双连通的极大子图称为**边双连通分量**。这篇博客就是总结一下求解无向图点双连通分量与边双连通分量的方法。

**算法：**   
求解点双连通分量与边双连通分量其实和求解割点与桥密切相关。不同双连通分量最多只有一个公共点，即某一个割顶，任意一个割顶都是至少两个点双连通的公共点。不同边双连通分量没有公共点，而桥不在任何一个边双连通分量中，点双连通分量一定是一个边双连通分量。   
下面首先介绍**点双连通分量的Tarjan算法**：   
在之前的博客中，我们已经知道如何求解[割顶](http://blog.csdn.net/fuyukai/article/details/51039788" \t "_blank)了，很容易可以发现，当我们找到割顶的时候，就已经完成了一次对某个极大点双连通子图的访问，那么我们如果在进行DFS的过程中将遍历过的点保存起来，是不是就可以得到点双连通分量了？为了实现算法，我们可以在求解割顶的过程中用一个栈保存遍历过的**边**（注意不是点！**因为不同的双连通分量存在公共点即割顶**），之后每当找到一个点双连通分量，即子结点v与父节点u满足关系low[v]>=dfn[u]，我们就将栈里的东西拿出来直到遇到当前边。   
这里注意放入栈中的不是点，而是边，这是因为点双连通分量是存在重复点的，如果我们放入栈中的是点，那么对于某些点双连通分量，就会少掉一些点（这些点都是割顶）。   
**代码：**

struct Edge{

int u,v;

Edge(int u=0,int v=0):u(u),v(v){}

}e[maxm];

int n,m,stamp,dfn[maxn],low[maxn],iscut[maxn],bccno[maxn];

int scnt,stack[maxm],bcc\_cnt;

vector<int> vec[maxn],bcc[maxn];

void tarjan(int index,int fa)

{

int child=0,tmp;

dfn[index]=low[index]=++stamp;

for(int i=0;i<vec[index].size();i++)

{

tmp=e[vec[index][i]].v;

if(!dfn[tmp])

{

stack[++scnt]=vec[index][i],child++;

tarjan(tmp,index);

low[index]=min(low[index],low[tmp]);

if(low[tmp]>=dfn[index])

{

iscut[index]=1;

bcc[++bcc\_cnt].clear();

while(1)

{

int num=stack[scnt--];

if(bccno[e[num].u]!=bcc\_cnt)

{

bcc[bcc\_cnt].push\_back(e[num].u);

bccno[e[num].u]=bcc\_cnt;

}

if(bccno[e[num].v]!=bcc\_cnt)

{

bcc[bcc\_cnt].push\_back(e[num].v);

bccno[e[num].v]=bcc\_cnt;

}

if(e[num].u==index && e[num].v==tmp)

break;

}

}

}

else if(dfn[tmp]<dfn[index] && tmp!=fa)

{

stack[++scnt]=vec[index][i];

low[index]=min(low[index], dfn[tmp]);

}

}

if(fa<0 && child==1)

iscut[index]=0;

}

void find\_bcc()

{

// 割顶的bccno值无意义

memset(dfn,0,sizeof(dfn));

memset(low,0,sizeof(low));

memset(iscut,0,sizeof(iscut));

memset(bccno,0,sizeof(bccno));

memset(bcc,0,sizeof(bcc));

stamp=scnt=bcc\_cnt=0;

for(int i=1;i<=n;i++)

if(!dfn[i])

tarjan(i,-1);

}

这里需要十分注意的是，算法结束之后，每个结点会有一个编号，代表它属于哪一个点双连通分量，但是，**割顶的编号是完全没有意义的**！这个算法灵活使用了两个时间戳和栈，完成了点双连通分量的发现。   
**例题：**[UVALIVE 5135](http://acm.hust.edu.cn/vjudge/problem/viewProblem.action?id=19845" \t "_blank)

之后介绍**边双连通分量**的求解算法：   
边双连通分量的求解非常简单，因为边双连通分量之间没有公共边，而且桥不在任意一个边双连通分量中，所以算法十分简单，即先一次DFS找到所有桥，再一次DFS（排除了桥）找到边双连通分量。   
PS：当然可以用一次DFS实现。   
**代码：**

struct Edge{

int u,v;

Edge(int u=0,int v=0):u(u),v(v){}

}e[maxm];

int n,m,stamp,dfn[maxn],low[maxn],bccno[maxn],bcc\_cnt;

vector<int> vec[maxn],bcc[maxn];

bool g[maxn][maxn],isbridge[maxm];

void tarjan(int index,int fa)

{

int tmp;

dfn[index]=low[index]=++stamp;

for(int i=0;i<vec[index].size();i++)

{

tmp=e[vec[index][i]].v;

if(!dfn[tmp])

{

tarjan(tmp,index);

low[index]=min(low[index],low[tmp]);

if(low[tmp]>dfn[index])

isbridge[vec[index][i]]=isbridge[vec[index][i]^1]=1;

}

else if(dfn[tmp]<dfn[index] && tmp!=fa)

{

low[index]=min(low[index], dfn[tmp]);

}

}

}

void dfs(int index)

{

dfn[index]=1;

bccno[index]=bcc\_cnt;

for(int i=0;i<vec[index].size();i++)

{

int tmp=vec[index][i];

if(isbridge[tmp])

continue;

if(!dfn[e[tmp].v])

{

dfs(e[tmp].v);

}

}

}

void find\_ebcc(){

bcc\_cnt=stamp=0;

memset(dfn,0,sizeof(dfn));

memset(low,0,sizeof(low));

memset(isbridge,0,sizeof(isbridge));

memset(bccno,0,sizeof(bccno));

memset(bcc,0,sizeof(bcc));

for(int i=1;i<=n;i++)

if(!dfn[i])

tarjan(i, -1);

memset(dfn,0,sizeof(dfn));

for(int i=1;i<=n;i++)

{

if(!dfn[i])

{

bcc\_cnt++;

dfs(i);

}

}

}

**例题：**[POJ 3352](http://poj.org/problem?id=3352)