强连通分支、桥和割点

北京大学信息学院 郭炜

本讲义部分内容参考北京大学信息学院实验班袁洋、陈科吉同学讲义，特此致谢

定 义

- 在有向图G中，如果任意两个不同的顶点相互可达，则称该有向图是强连通的。有向图G的极大强连通子图称为G的强连通分支。

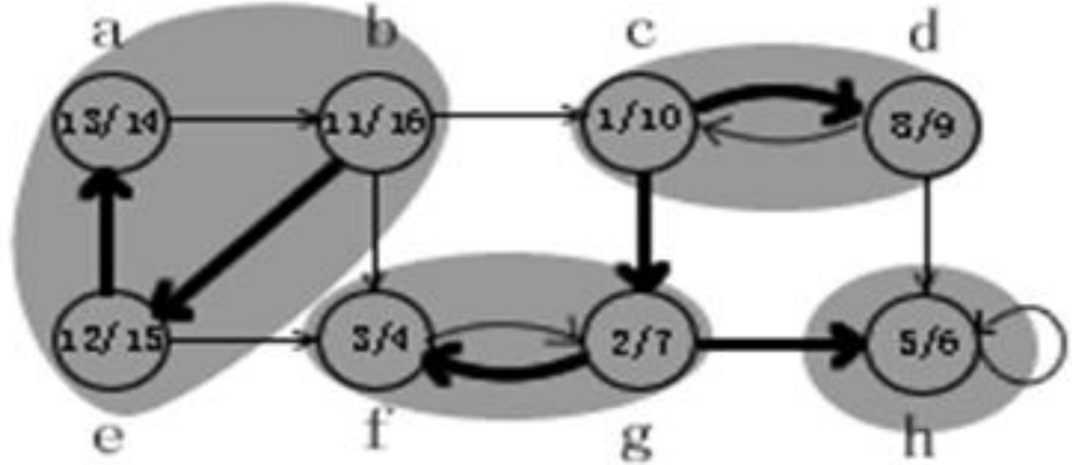
- 转置图的定义：将有向图G中的每一条边反向形成的图称为G的转置G^T。（注意到原图和G^T的强连通分支是一样的）

Korasaju算法求有向图强连通分支

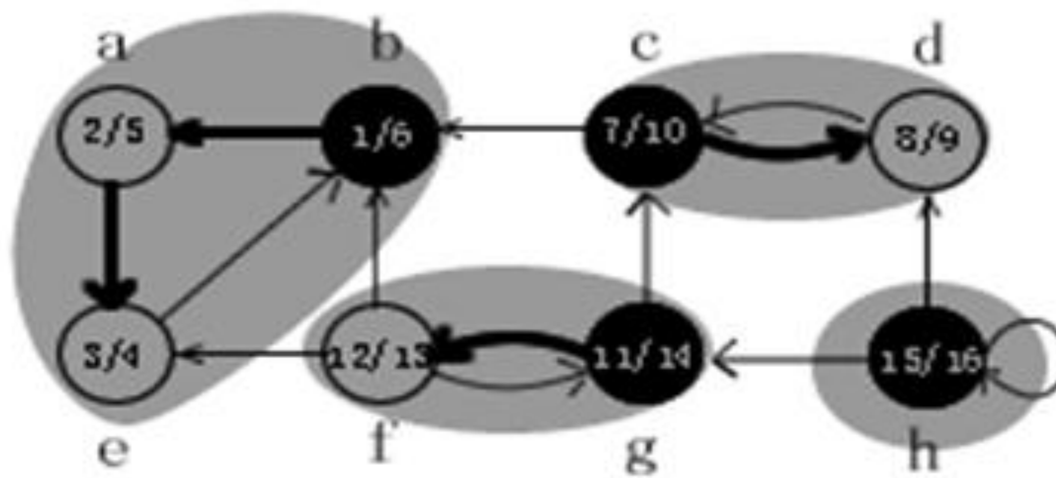
- procedure Strongly_Connected_Components(G);
- begin
- 1.深度优先遍历G，算出每个结点u的结束时间f[u],起点如何选择无所谓。
  - 2.深度优先遍历G的转置图G^T，选择遍历的起点时，按照结点的结束时间从大到小进行。遍历的过程中，一边遍历，一边给结点做分类标记，每找到一个新的起点，分类标记值就加1。
- 3. 第2步中产生的标记值相同的结点构成深度优先森林中的一棵树，也即一个强连通分量
- end;
- 证明参考：http://www.bioisland.com/Algorithm/ShowArticle.asp?ArticleID=58

(a)为有向图G，其中的阴影部分是G的强连通分支，对每个顶点都标出了其发现时刻与完成时刻，黑色边为深度优先搜索的树枝；

(b)G的转置图G^T依次以b, c, g, h为起点做DFS，得到4个强连通分量



(a)



(b)

算法复杂度分析

- 深度优先搜索的复杂度:Θ(V + E)
- 计算G^T的复杂度:0或者Θ(V + E)(临接表)
- 所以总的复杂度为:Θ(V + E)
- 非常好的算法!

POJ2186:Popular Cows

- 给定一个有向图，求有多少个顶点是由任何顶点出发都可达的。
- 顶点数<= 10,000,边数 <= 50,000

有用的定理:

有向无环图中唯一出度为0的点，一定可以由任何点出发均可达(由于无环，所以从任何点出发往前走，必然终止于一个出度为0的点)

POJ2186: 解题思路

- 1. 求出所有强连通分量
- 2. 每个强连通分量缩成一点，则形成一个有向无环图DAG。
- 3. DAG上面如果有唯一的出度为0的点,则该点能被所有的点可达。那么该点所代表的连通分量上的所有的原图中的点，都能被原图中的所有点可达，则该连通分量的点数，就是答案。
- 4. DAG上面如果有不止一个出度为0的点，则这些点互相不可达，原问题无解，答案为0

POJ2186: 解题思路

- 缩点的时候不一定要构造新图，只要把不同强连通分量的点染不同颜色，然后考察各种颜色的点有没有连到别的颜色的边即可(即其对应的缩点后的DAG图上的点是否有出边）。

POJ1236: Network of Schools

- 题目大意：N(2<N<100)各学校之间有单向的网络，每个学校得到一套软件后，可以通过单向网络向周边的学校传输，问题1：初始至少需要向多少个学校发放软件，使得网络内所有的学校最终都能得到软件。2，至少需要添加几条传输线路(边)，使任意向一个学校发放软件后，经过若干次传送，网络内所有的学校最终都能得到软件。

ACM1236: Network of Schools

- 给定一个有向图，求：

1) 至少要选几个顶点，才能做到从这些顶点出发，可以到达全部顶点

2) 至少要加多少条边，才能使得从任何一个顶点出发，都能到达全部顶点

- 顶点数<= 100

有用的定理:

有向无环图中所有入度不为0的点，一定可以由某个入度为0的点出发可达。(由于无环，所以从任何入度不为0的点往回走，必然终止于一个入度为0的点)

ACM1236: 解题思路

- 1. 求出所有强连通分量
- 2. 每个强连通分量缩成一点，则形成一个有向无环图DAG。
- 3. DAG上面有多少个入度为0的顶点，问题1的答案就是多少

ACM1236: 解题思路

- 在DAG上要加几条边，才能使得DAG变成强连通的，问题2的答案就是多少
- 加边的方法：
- 要为每个入度为0的点添加入边，为每个出度为0的点添加出边
- 假定有 n 个入度为0的点，m个出度为0的点，max(m,n)就是第二个问题的解(证明难，略)

有向图强连通分支的Tarjan算法

- 做一遍DFS，用dfn[i]表示编号为i的节点在DFS过程中的访问序号(也可以叫做开始时间）。在DFS过程中会形成一搜索树。在搜索树上越先遍历到的节点，显然dfn的值就越小。dfn值越小的节点，就称为越“早”。
- 用low[i]表示从i节点出发DFS过程中i下方节点(开始时间大于dfn[i]，且由i可达的节点）所能到达的最早的节点的开始时间。初始时low[i]=dfn[i]
- DFS过程中，碰到哪个节点，就将哪个节点入栈。栈中节点只有在其所属的强连通分量已经全部求出时，才会出栈。
- 如果发现某节点u有边连到栈里的节点v，则更新u的low 值为min(low[u],dfn[v])，若low[u]被更新为dfn[v],则表明目前发现u可达的最早的节点是v.

有向图强连通分支的Tarjan算法

- 对于u的子节点v，从v出发进行的DFS结束回到u时，使得 low[u] = min(low[u],low[v])。因为u可达v,所以v可达的最早的节点，也是u可达的。

- 如果一个节点u，从其出发进行的DFS已经全部完成并回到u，而且此时其low值等于dfn值，则说明u可达的所有节点，都不能到达任何比u早的节点 ---- 那么该节点u就是一个强连通分量在DFS搜索树中的根。

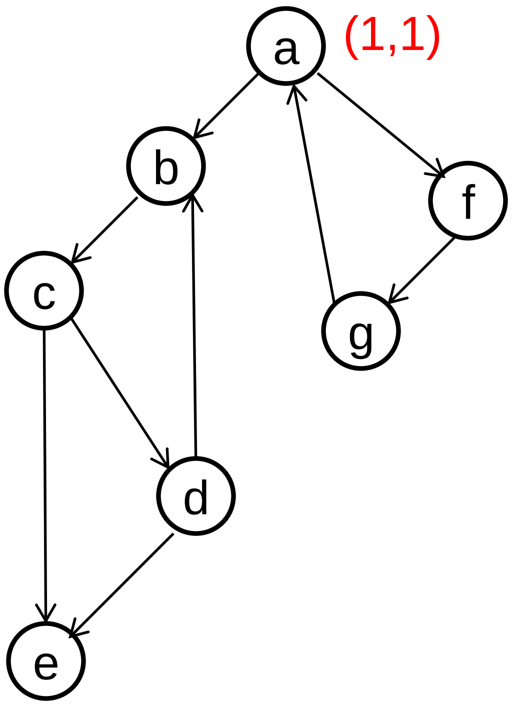
- 此时，显然栈中u上方的节点，都是不能到达比u早的节点的。将栈中节点弹出，一直弹到u(包括u),弹出的节点就构成了一个强连通分量.

有向图强连通分支的Tarjan算法

```
void Tarjan(u) {
    dfn[u]=low[u]=++index
    stack.push(u)
    for each (u, v) in E {
            if (v is not visted) {
                        tarjan(v)
                        low[u] = min(low[u], low[v])
            }
            else if (v in stack) {
                        low[u] = min(low[u], dfn[v])
            }
    }
    if (dfn[u] == low[u]) { //u是一个强连通分量的根
            repeat
                        v = stack.pop
                        print v
            until (u== v)
    } //退栈，把整个强连通分量都弹出来
} //复杂度是O(E+V)的
```

(dfn,low)

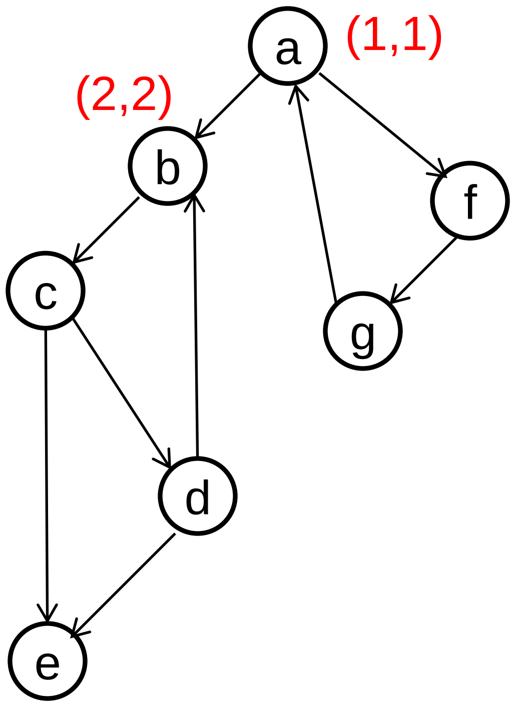
栈



a

(dfn,low)

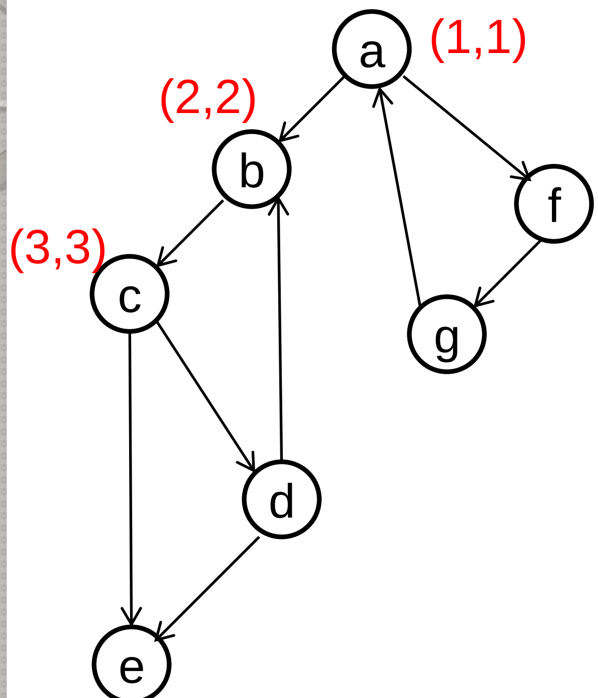
栈



b
a

(dfn,low)

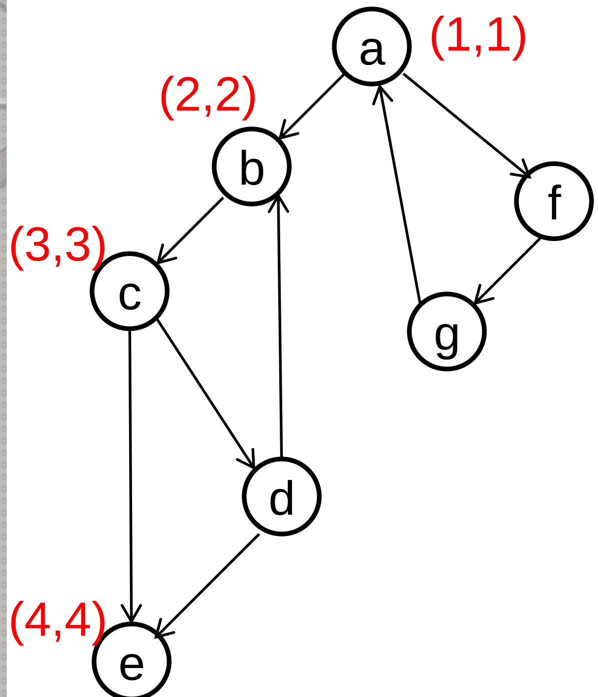
栈



c
b
a

(dfn,low)

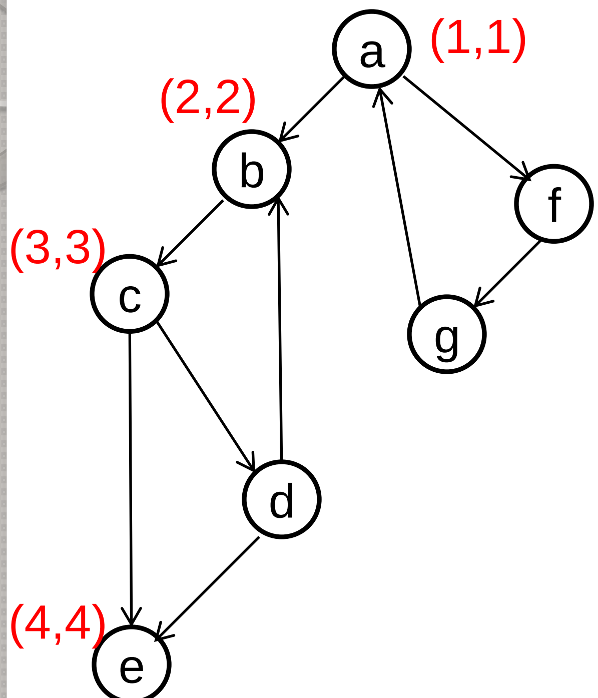
栈

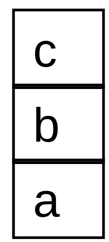


e
c
b
a

(dfn,low)

栈



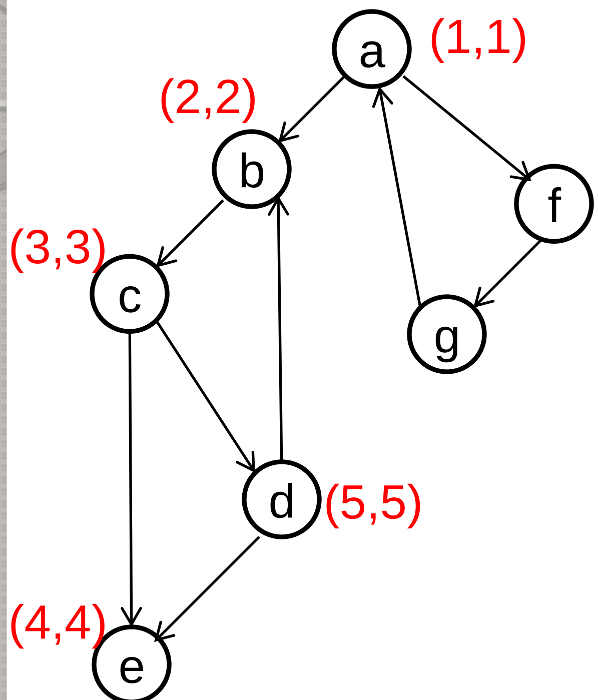


强连通分量：

{e}

(dfn,low)

栈

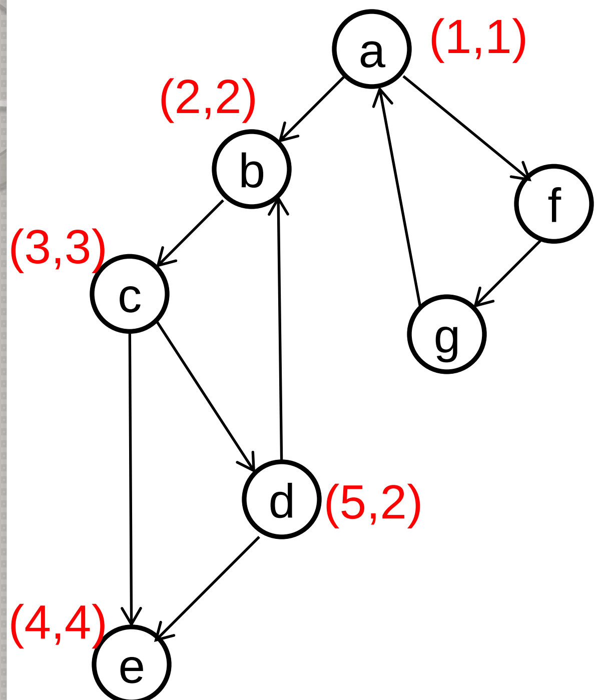


d
c
b
a

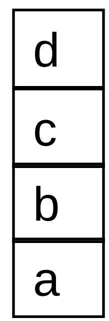
强连通分量：

{e}

(dfn,low)



栈

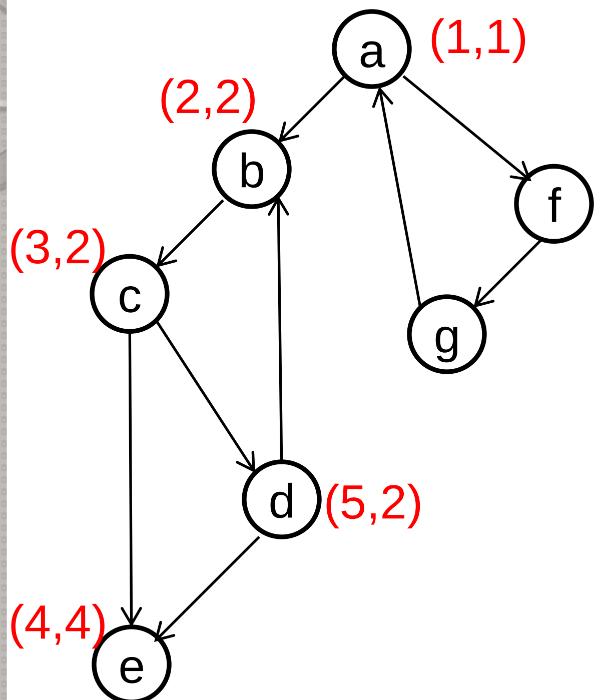


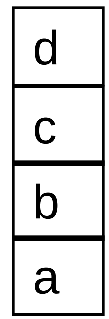
强连通分量：

{e}

(dfn,low)

栈



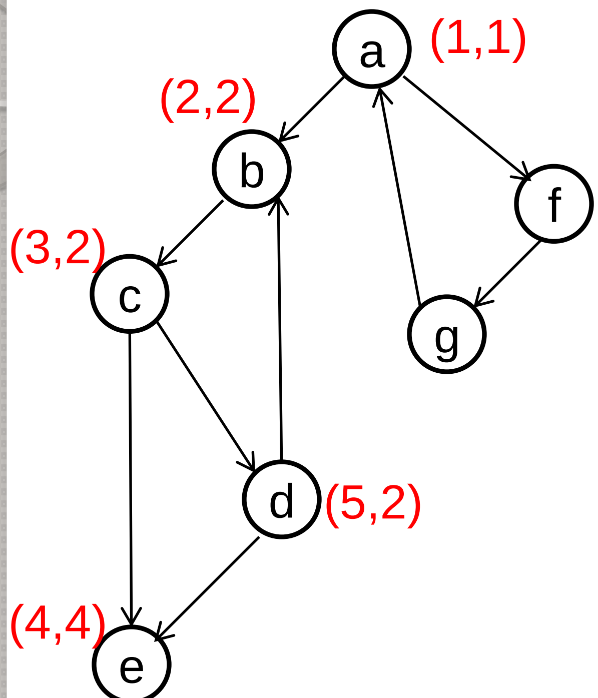


强连通分量：

{e}

(dfn,low)

栈



强连通分量：

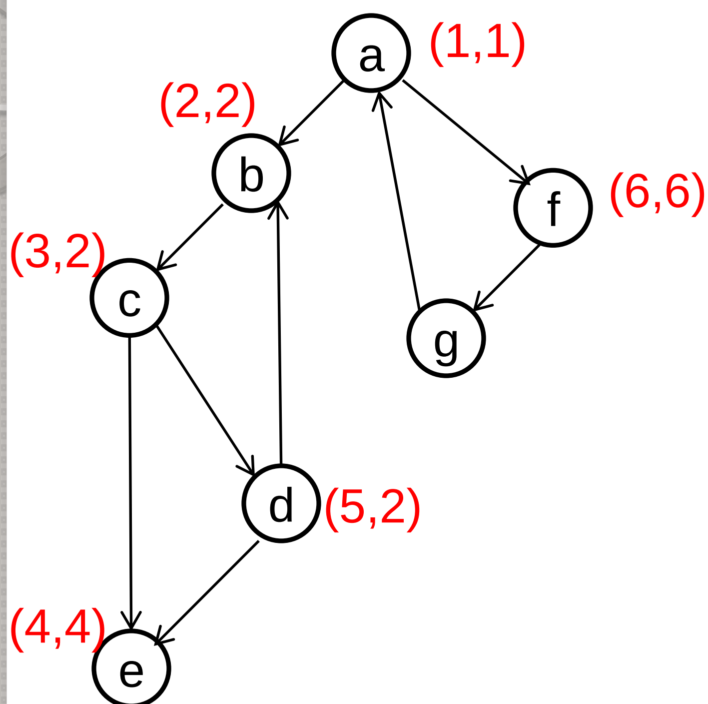
{e}

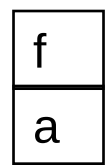
{b c d}

a

(dfn,low)

栈





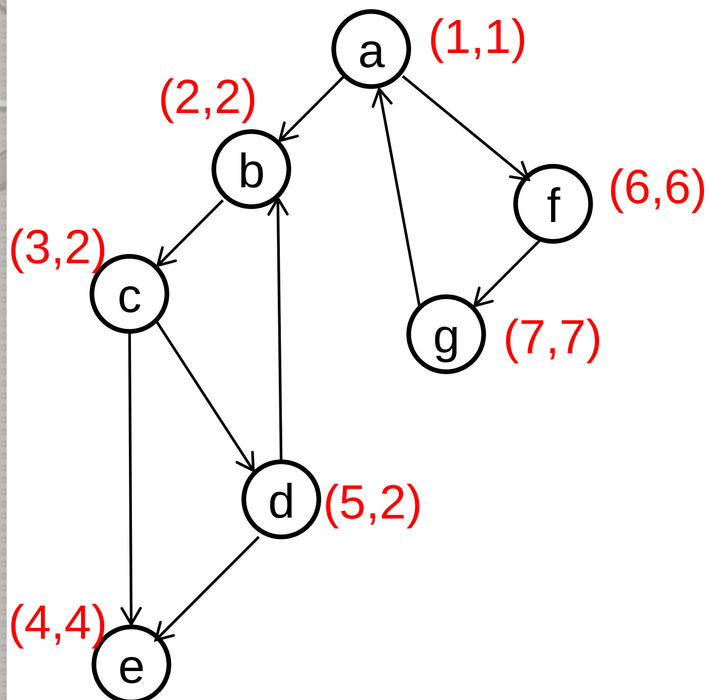
强连通分量：

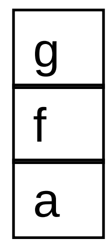
{e}

{b c d}

(dfn,low)

栈





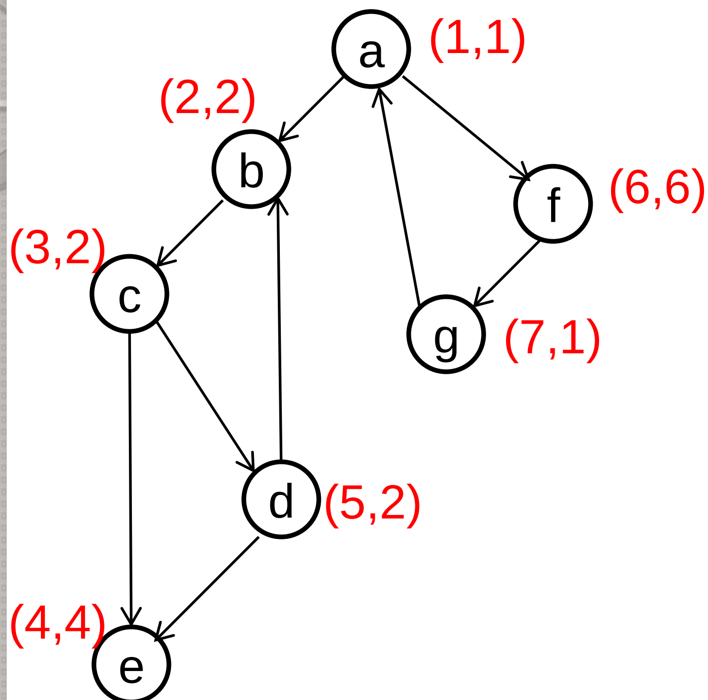
强连通分量：

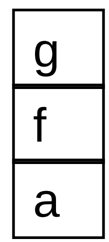
{e}

{b c d}

(dfn,low)

栈





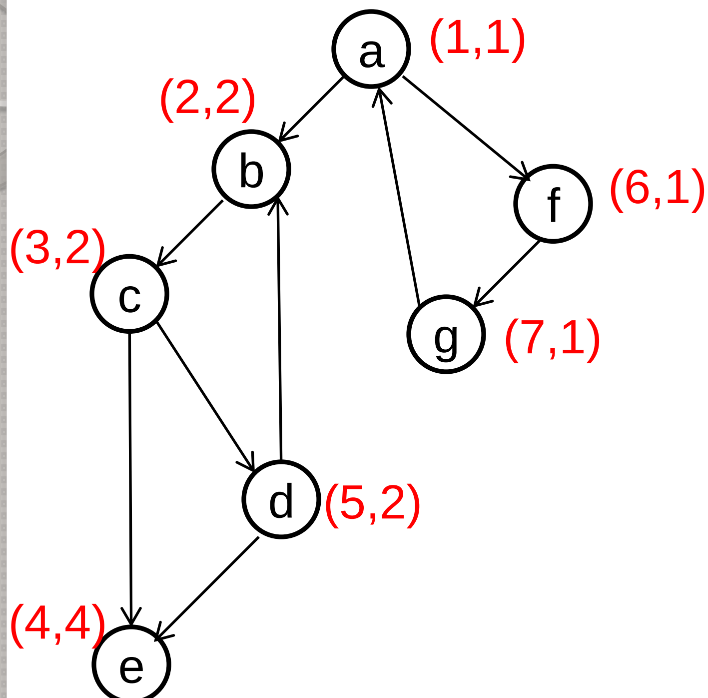
强连通分量：

{e}

{b c d}

(dfn,low)

栈



强连通分量：

{e}

{b c d}

{a f g}

有向图强连通分支的Tarjan算法

为什么从u出发的DFS全部结束回到u时，若dfn[u]=low[u], 此时将栈中u及其上方的节点弹出，就找到了一个强连通分量?

此时所有节点分成以下几类：

1）还没被访问过的节点
2) 栈中比u早的节点
3) 栈中比u晚的节点
4) 栈中的u
5) 曾经入栈（访问过），又出了栈的节点

要证明1) 2) 5) 三类节点，都是要么从u不可达，要么不可达u，且 3) 4）类节点互相可达

有向图强连通分支的Tarjan算法

此时所有节点分成以下几类：

1）还没被访问过的节点(显然由u不可达）

2) 栈中比u早的节点 (由u不可达，因为low[u]=dfn[u])

3) 栈中比u晚的节点 (由u可达）

4) 栈中的u

5) 曾经入栈（访问过），又出了栈的节点（不可达u)

要证明：

1. 第5)类节点不可达u
2. 2. 第3)类节点可达u

有向图强连通分支的Tarjan算法

1. 证第5)类节点不可达u
此类节点分成两部分：

1） 早于u的
2） 晚于u的

早于u的不可达u。若可达的话，它此时应该还在栈里面，u的下面 ---导致矛盾。

第2类中，任取节点x,假设x可达的最早节点是y，则y一定晚于u, 即 x不可达u.

有向图强连通分支的Tarjan算法

第2类中，任取节点x。X之所以已经被弹出栈，一定是因为最终 low[x] = dfn[x]，或x位于某个y节点上方，由y可达,且y满足条件:最终的 low[y] = dfn[y]。 因为y曾经出现在u的上方，所以y一定晚于u。因为low[x]不可能小于等于dfn[u](否则low[y]就也会小于等于dfn[u],这和low[y]=dfn[y]矛盾），所以x到达不了u及比u早的节点。

1. 证第5)类节点不可达u　证毕

有向图强连通分支的Tarjan算法

2. 第3)类节点可达u

   若有此类节点x不可达u，则考虑最终的low[x]：

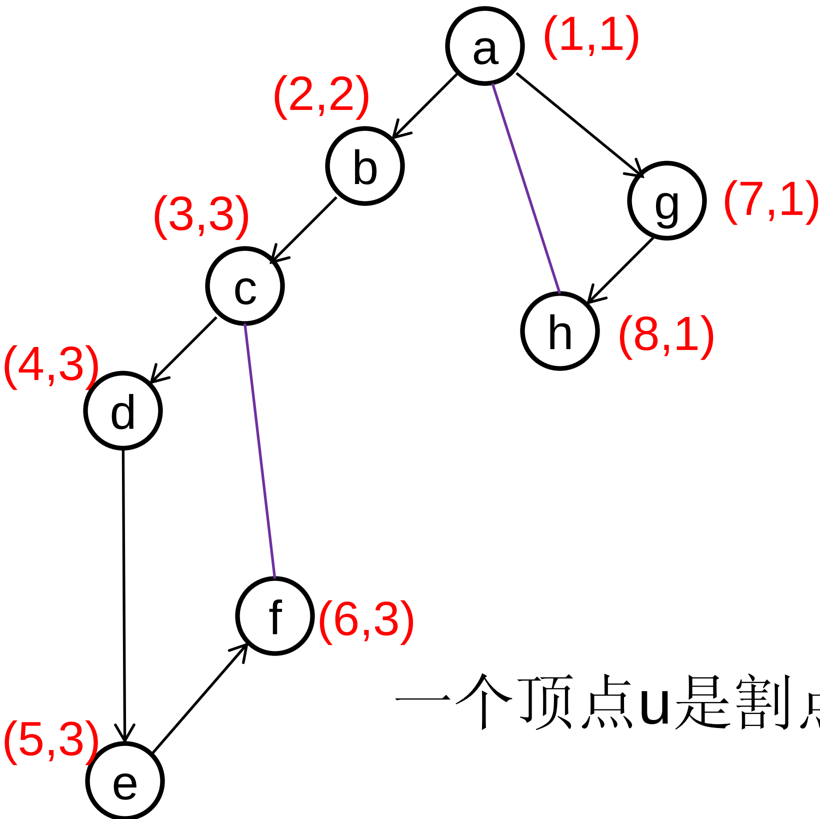
low[x] < dfn[u]　　不可能，否则有 low[u]<dfn[u]
low[x] = dfn[u]　　x可达u
其他：
寻找x所能到达的最早的，栈里面的节点y,则y必然比u晚。而且有 low[y] = dfn[y]（若此条不成立，则x还能到达比y更早的节点，矛盾）。而若 low[y] = dfn[y]，则y应该已经被弹出栈了，y上方的x当然也已经不再栈中，这和x是第3类节点矛盾。

无向连通图求割点和桥

- 无向连通图中，如果删除某点后，图变成不连通，则称该点为割点。
- 无向连通图中，如果删除某边后，图变成不连通，则称该边为桥。

求桥和割点的Tarjan算法

- 思路和有向图求强连通分量类似
- 在深度优先遍历整个图过程中形成的一棵搜索树
- dfn[u]定义和前面类似，但是low[u]定义为u或者u的子树中能够通过非父子边追溯到的最早的节点的DFS开始时间



割点：

c b a

桥

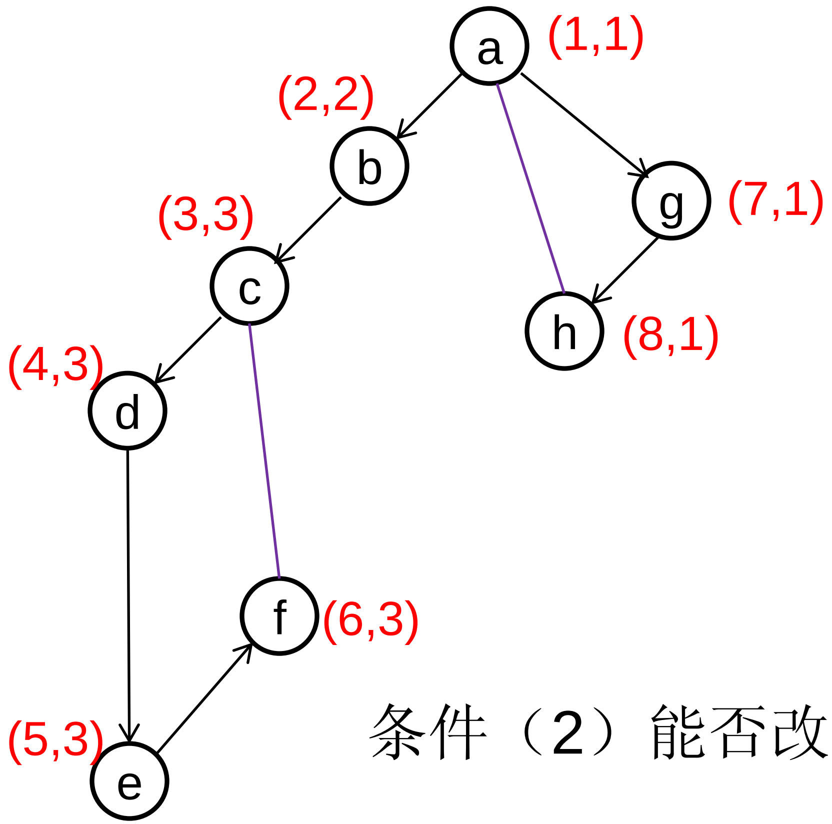
ab
bc

一个顶点u是割点，当且仅当满足(1)或(2)

(1) u为树根，且u有多于一个子树。
(2) u不为树根，且满足存在(u,v)为树枝边(或称父子边，即u为v在搜索树中的父亲)，使得dfn(u)<=low(v)。

非树枝边不可能是桥

一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足dfn(u)<low(v)（前提是其没有重边）。



割点：

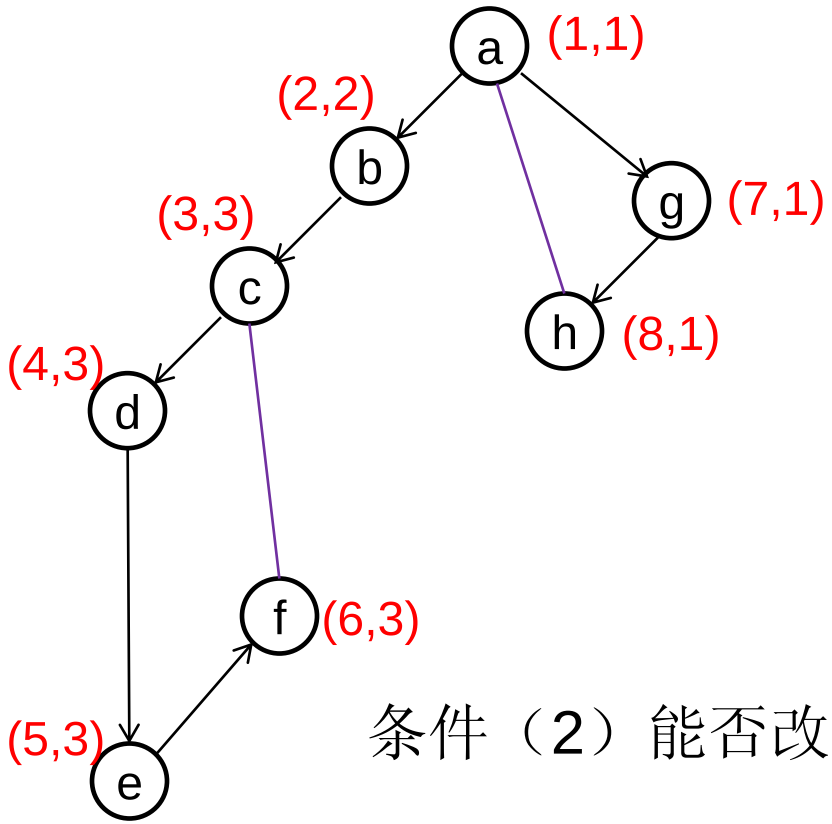
c b a

桥

ab
bc

条件（2）能否改成：

u不为树根，且满足dfn(u) = low(u)？



割点：

c b a

桥

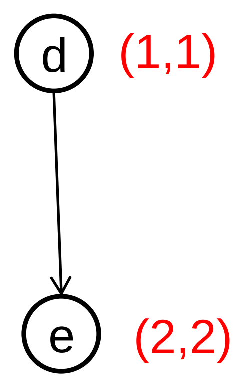
ab
bc

条件（2）能否改成：

u不为树根，且满足dfn(u) = low(u)？

不行。考虑

d,e都不是割点



求桥和割点的Tarjan算法

- low[u]定义为u或者u的子树中能够通过非父子边追溯到的最早的节点的DFS开始时间
- 如果下面程序没有：

if(v 不是u 的父节点)

则求不出桥了

求桥和割点和桥的Tarjan算法

```
Tarjan(u) {
        d[u]=low[u]=++index
        for each (u, v) in E {
                if (v is not visted)
                        tarjan(v)
                        low[u] = min(low[u], low[v])
                         d[u]<low[v] ⇔  (u, v) 是桥
                }
                else {
                        if(v 不是u 的父节点)
                                low[u] = min(low[u], d[v])
                }
        }
        if (u is root)
           u 是割点 <=> u 有至少两个子节点
        else
          u 是割点 <=> u 有一个子节点v，满足d[u]<= low[v]
}
```

Tarjan’s algorithm

- 也可以先用Tajan()进行dfs算出所有点的low和dfn值，并记录dfs过程中每个点的父节点，然后再把所有点看一遍，看其low和dfn,以找出割点和桥。

- 找桥的时候，要注意看有没有重边。有重边，则不是桥。

无重边连通无向图求割点和桥的程序
给出点数和所有的边，求割点和桥

Input: （11点13边）

11 13
1 2
1 4
1 5
1 6
2 11
2 3
4 3
4 9
5 8
5 7
6 7
7 10
11 3

output:
1
4
5
7
5,8
4,9
7,10

```
//无重边连通无向图求割点和桥的程序
#include <iostream>
#include <vector>
using namespace std;

#define MyMax 200
typedef vector<int> Edge;
vector<Edge> G(MyMax);
bool Visited[MyMax] ;
int dfn[MyMax] ;
int low[MyMax] ;
int Father[MyMax]; //DFS树中每个点的父节点
bool bIsCutVetext[MyMax]; //每个点是不是割点
int nTime; //Dfs时间戳
int n,m; //n是点数，m是边数
```

```
void Tarjan(int u, int father) //father 是u的父节点
{
        Father[u] = father;
        int i,j,k;
        low[u] = dfn[u] = nTime ++;
        for( i = 0;i < G[u].size() ;i ++ ) {
                int v = G[u][i];
                if( ! dfn[v]) {
                        Tarjan(v,u);
                        low[u] = min(low[u],low[v]);
                }
                else if( father != v ) //连到父节点的回边不考虑，否则求不出桥
                        low[u] = min(low[u],dfn[v]);
        }
}
```

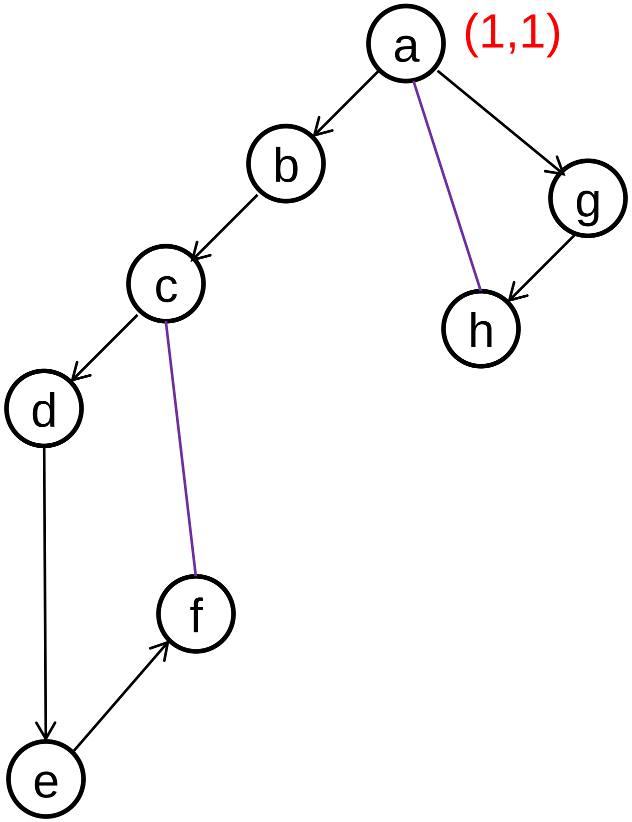
```
void Count()
{ //计算割点和桥
        int nRootSons = 0;          int i;
        Tarjan(1,0);
        for( i = 2;i <= n;i ++ ) {
                int v = Father[i];
                if( v == 1 )
                        nRootSons ++; //DFS树中根节点有几个子树
                else {
                        if( dfn[v] <= low[i])
                                bIsCutVetext[v] = true;
                }
        }
        if( nRootSons > 1)
                bIsCutVetext[1] = true;
        for( i = 1;i <= n;i ++ )
                if( bIsCutVetext[i] )
                        cout << i << endl;
        for( i = 1;i <= n;i ++) {
                int v = Father[i];
                if(v >0 &&  dfn[v] < low[i])
                        cout << v << "," << i <<endl;
        }
}
```

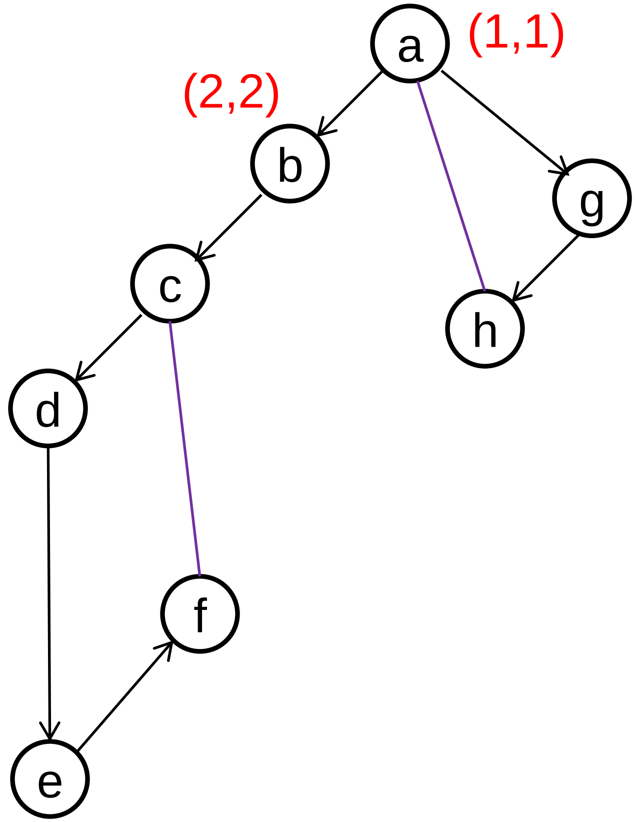
```
int main()
{
	int u,v;
	int i;

	nTime = 1;
	cin >> n >> m ; //n是点数，m是边数
	for(  i = 1;i <= m;i ++ ) {
		cin >> u >> v; //点编号从1开始
		G[v].push_back(u);
		G[u].push_back(v);
	}
	memset( dfn,0,sizeof(dfn));
	memset( Father,0,sizeof(Father));
	memset( bIsCutVetext,0,sizeof(bIsCutVetext));
	Count();
	return 0;
}
```

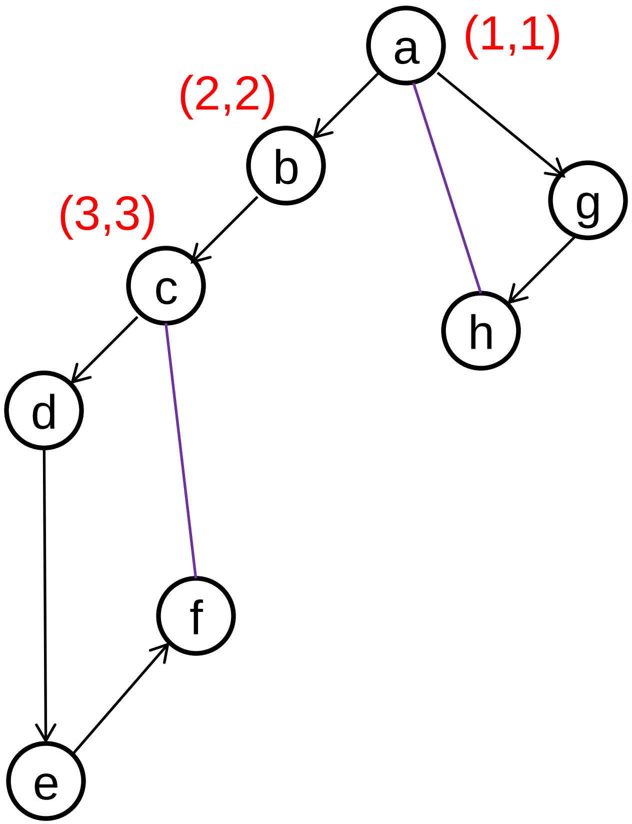
求无向图连通图点双连通分支(不包含割点的极大连通子图）：

- 对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或反向边(连到树中祖先的边），就把这条边加入栈中。如果遇到某边树枝边（u,v) 满足dfn(u)<=low(v)，说明u是一个割点，此时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支。

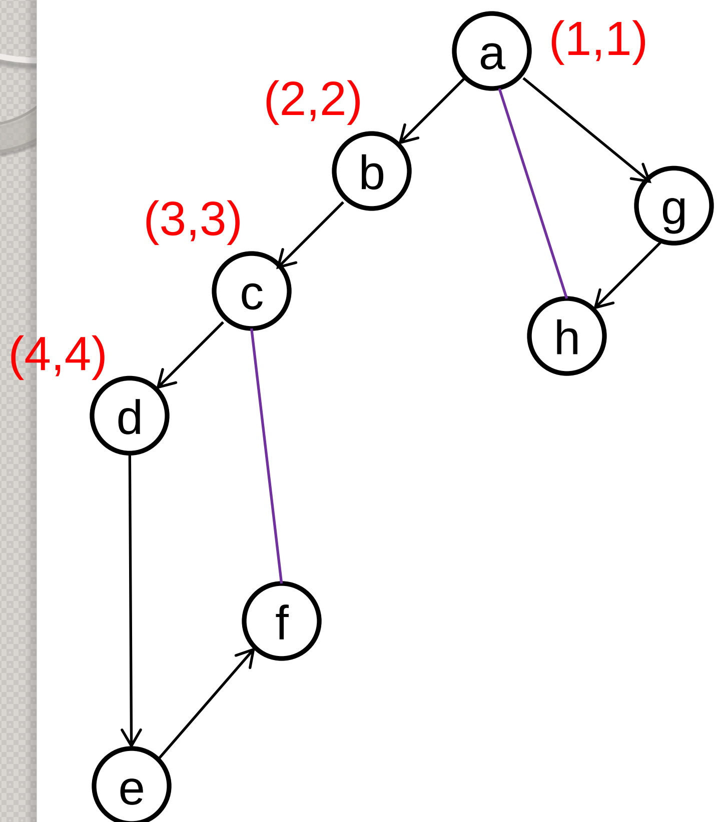

a
(1,1)
b
g
c
h
d
f
e



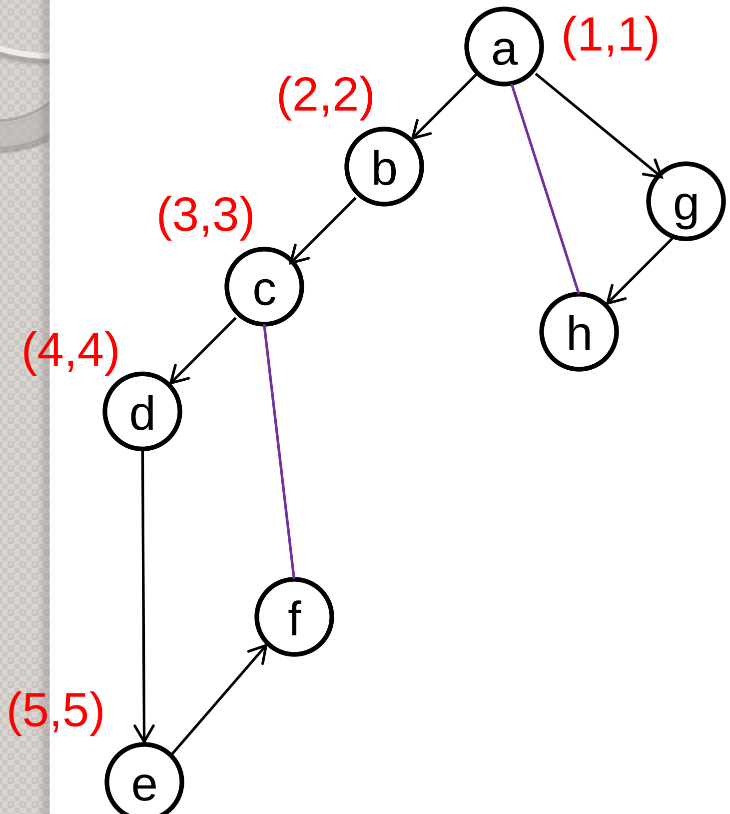
ab



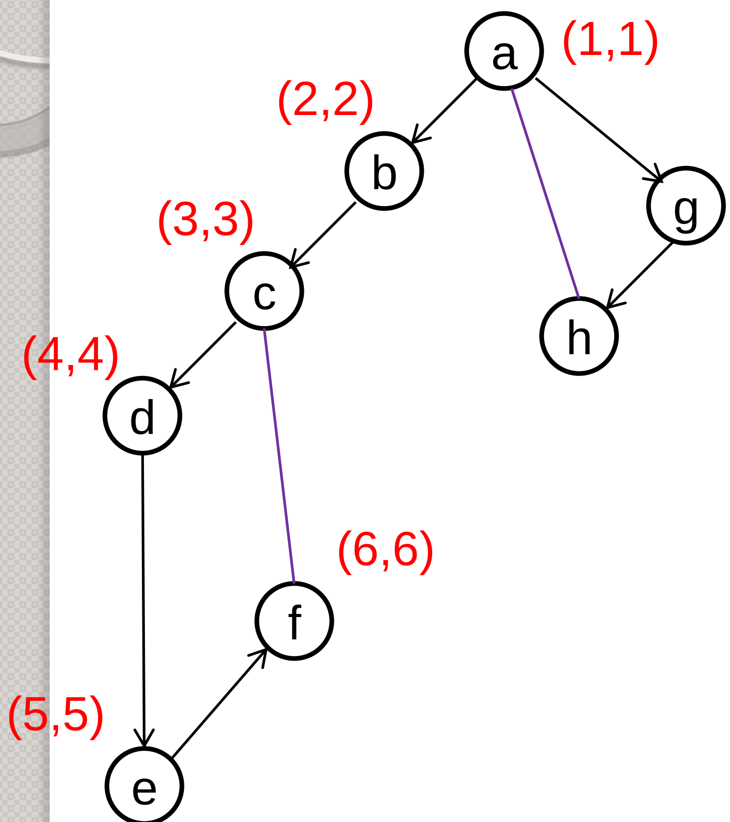
bc
ab



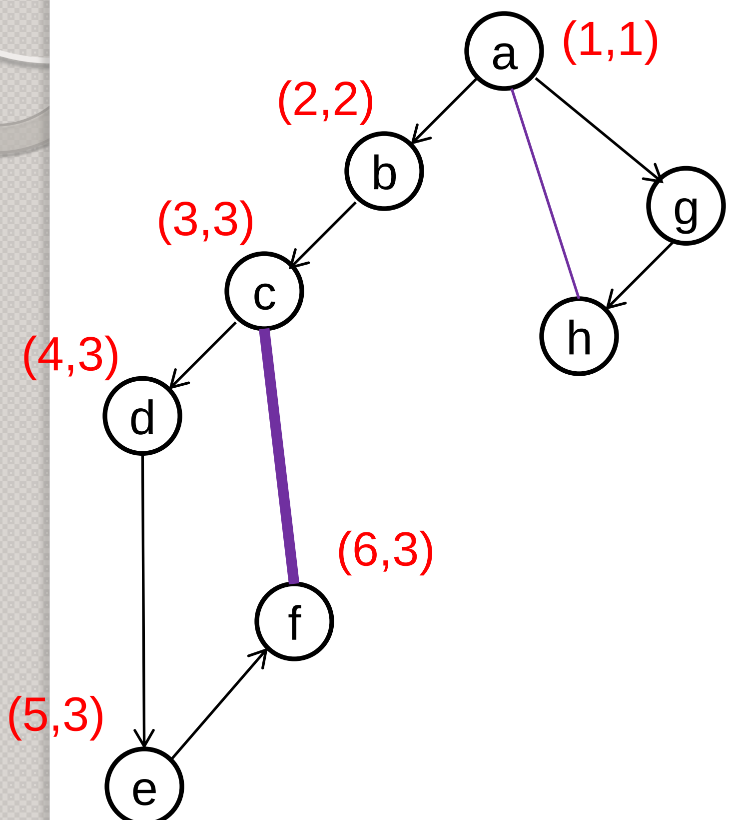
cd
bc
ab



de
cd
bc
ab

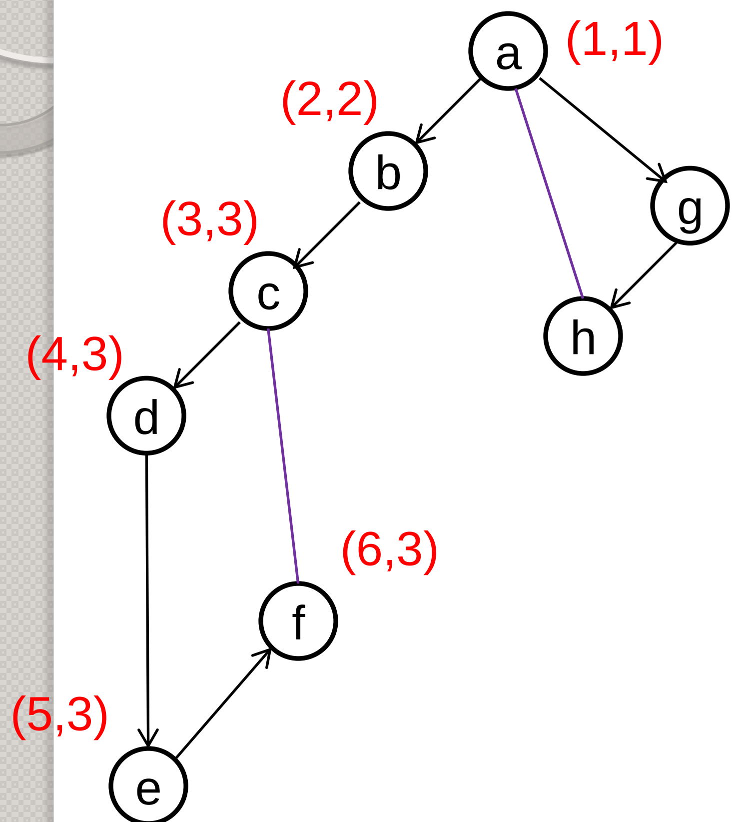


ef
de
cd
bc
ab



fc
ef
de
cd
bc
ab

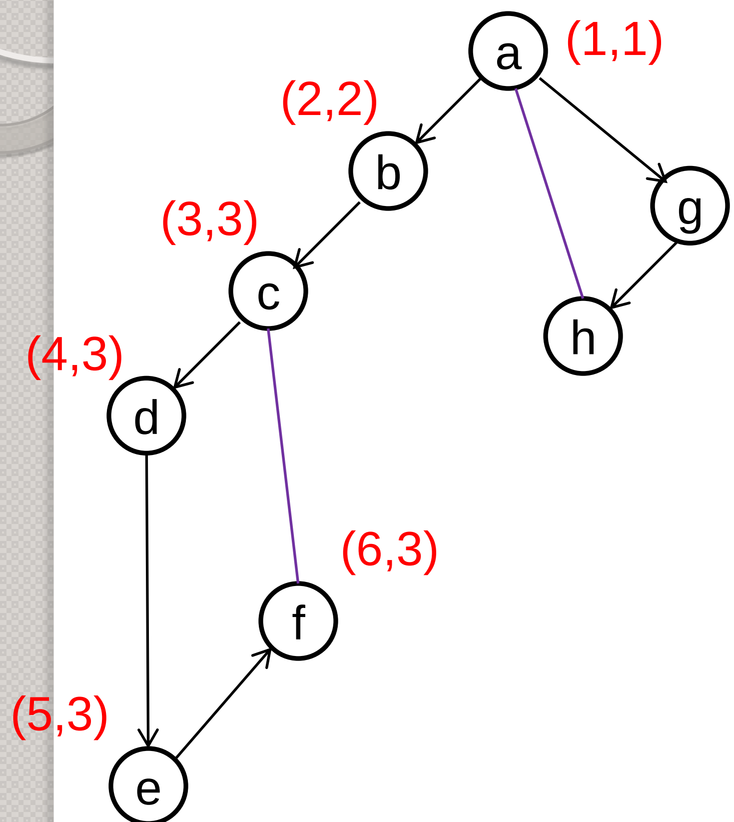
不要让 fc 入栈两次！



点双连通分量：

{ fc,ef,de,cd }

bc
ab

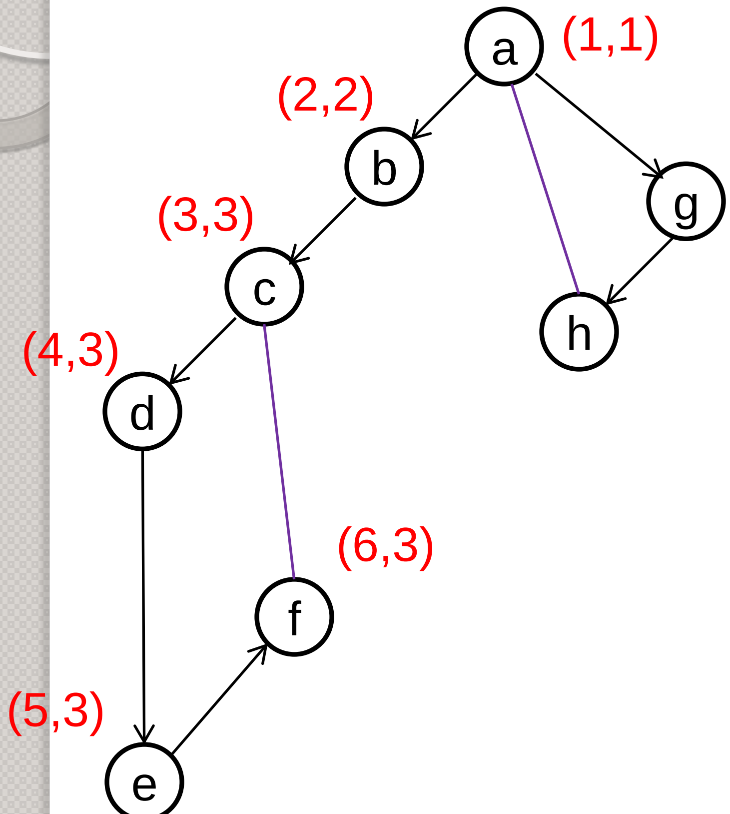


点双连通分量：

{ fc,ef,de,cd }

{bc }

ab

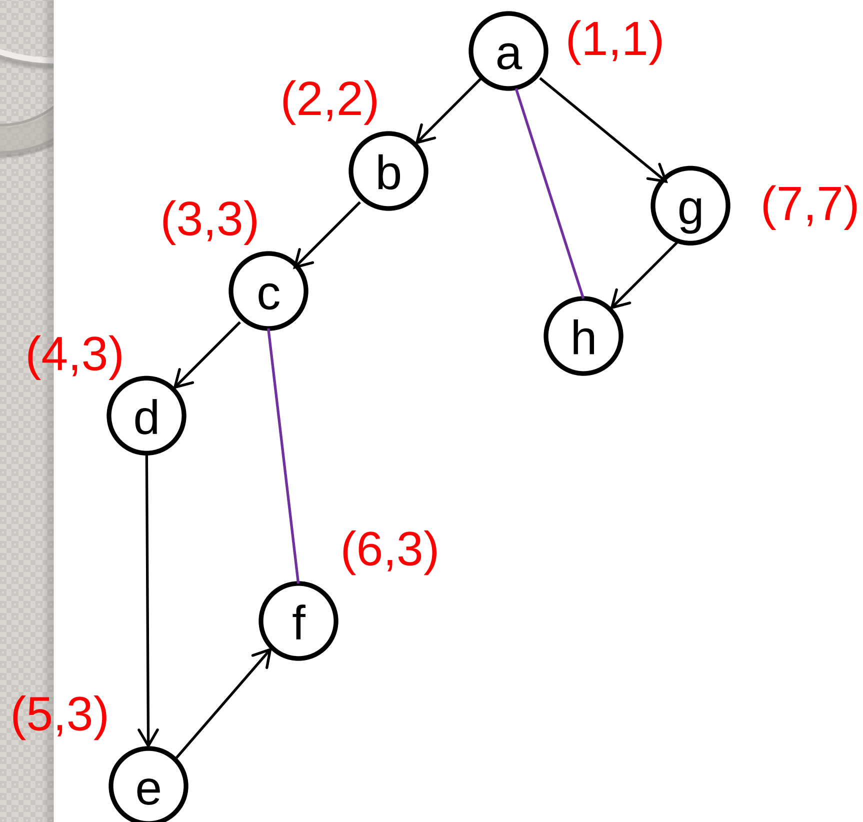


点双连通分量：

{ fc,ef,de,cd }

{ bc }

{ab}



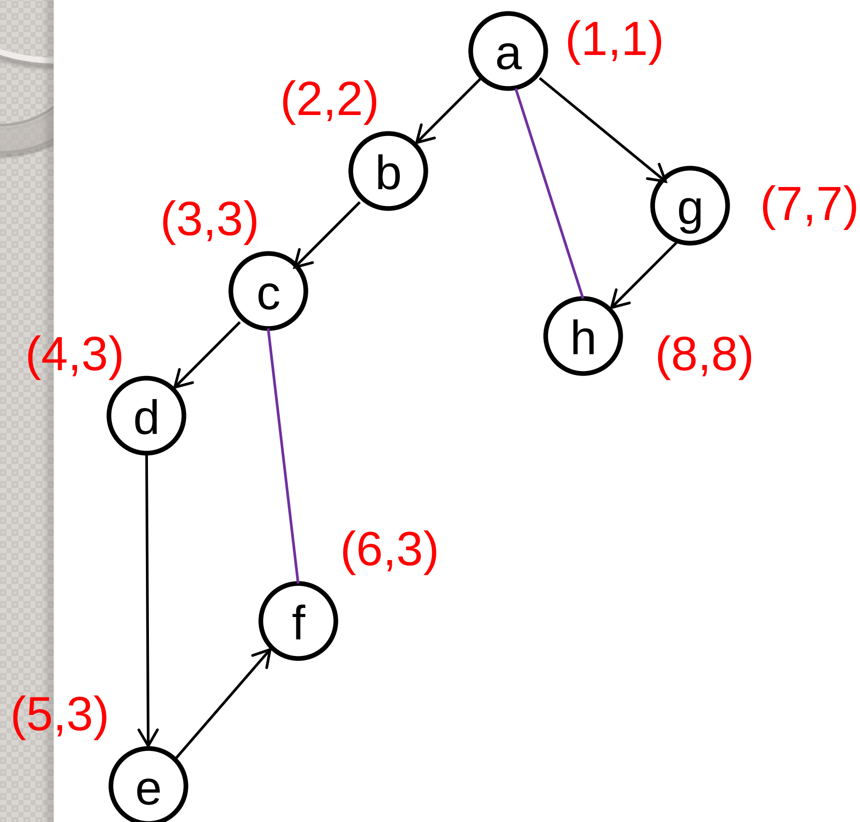
点双连通分量：

{ fc,ef,de,cd }

{ bc }

{ab}

ag



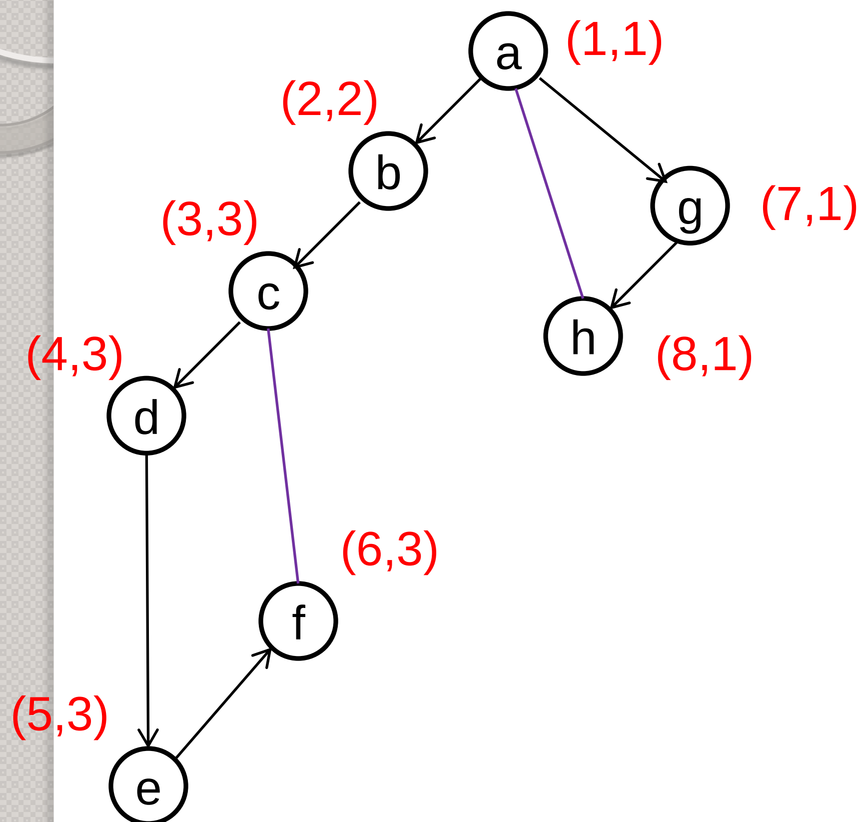
点双连通分量：

{ fc,ef,de,cd }

{ bc }

{ab}

gh
ag



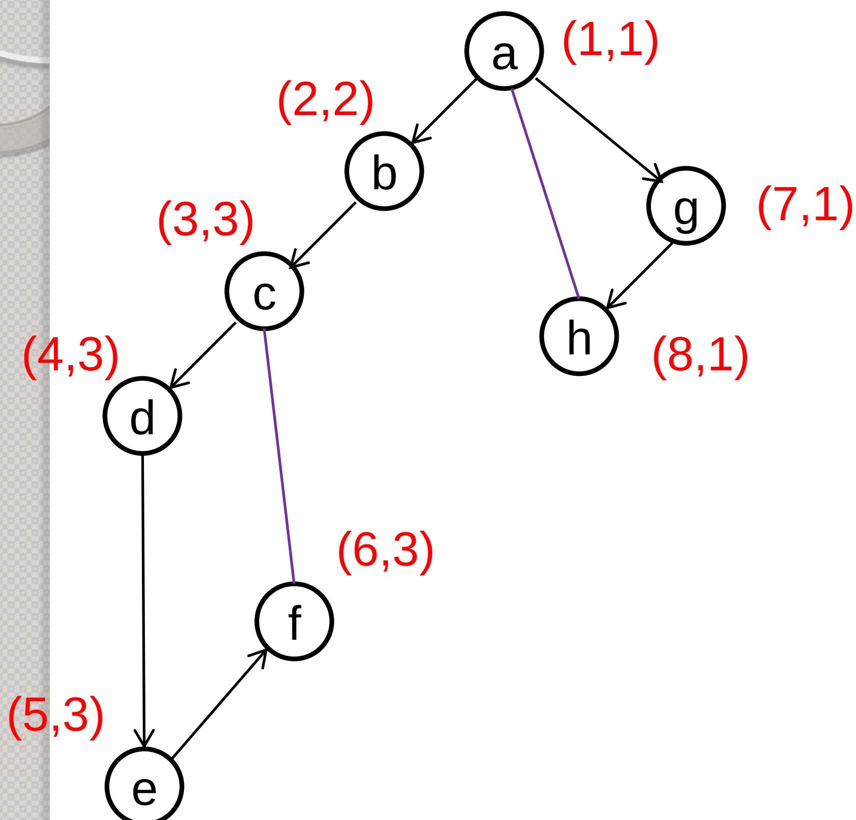
ha
gh
ag

点双连通分量：

{ fc,ef,de,cd }

{ bc }

{ab}



点双连通分量：

{ fc,ef,de,cd }

{ bc }

{ab}

{ha,gh,ag}

求无向连通图点双连通分量（没有割点的连通分量），假定没有重边

Input: （11点13边）

11 13
1 2
1 4
1 5
1 6
2 11
2 3
4 3
4 9
5 8
5 7
6 7
7 10
11 3

output:

Block No: 1
4,9
Block No: 2
4,1
3,4
3,2
11,3
2,11
1,2
Block No: 3
5,8
Block No: 4
7,10
Block No: 5
6,1
7,6
5,7
1,5

```
//求无向连通图点双连通分量（没有割点的连通分量）,假定没有重边
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
#define MyMax 200
typedef vector<int> Edge;
vector<Edge> G(MyMax);
int dfn[MyMax] ;
int low[MyMax] ;
int nTime;
int n,m;  //n是点数，m是边数
struct Edge2
{
		int u;
		int v;
		Edge2(int u_,int v_):u(u_),v(v_) { }
};
deque<Edge2> Edges;
int nBlockNo = 0;
```

```
void Tarjan(int u, int father)
{
        int i,j,k;
        low[u] = dfn[u] = nTime ++;
        for( i = 0;i < G[u].size() ;i ++ ) {
                int v = G[u][i];
                if( ! dfn[v]) { //v没有访问过
                        //树边要入栈
                        Edges.push_back(Edge2(u,v));
                        Tarjan(v,u);
                        low[u] = min(low[u],low[v]);
                        Edge2 tmp(0,0);
                        if(dfn[u] <= low[v]) {
                        //从一条边往下走，走完后发现自己是割
点，则栈中的边一定全是和自己在一个双连通分量里面
                //根节点总是和其下的某些点在同一个双连通分量
里面
                            cout << "Block No: " << ++ nBlockNo
                                << endl;
```

```
                        do {
                                tmp = Edges.back();
                                Edges.pop_back ();
                                cout << tmp.u << "," <<
                                          tmp.v << endl;
                        }while ( !(tmp.u == u &&
                                          tmp.v == v) );
                }
        }  // 对应if( ! dfn[v]) {
        else {
                if( v != father ) {//u连到父节点的回边不考虑
                    low[u] = min(low[u],dfn[v]);
                    if( dfn[u] > dfn[v])
                                //连接到祖先的回边要入栈，
但是连接到儿子的边，此处肯定已经入过栈了，不能再入栈
                        Edges.push_back(Edge2(u,v));
                }
        }
    } //对应  for( i = 0;i < G[u].size() ;i ++ ) {
}
```

```
int main()
{

	int u,v;
	int i;
	nTime = 1;
	cin >> n >> m ;  //n是点数，m是边数
	nBlockNo = 0;
	for(  i = 1;i <= m;i ++ ) {
		cin >> u >> v; //点编号从1开始
		G[v].push_back(u);
		G[u].push_back(v);
	}
	memset( dfn,0,sizeof(dfn));
	Tarjan(1,0);
	return 0;
}
```

求无向连通图边双连通分支(不包含桥的极大连通子图）：

只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支。

POJ 3352 Road Construction

- 给你一个图，要求你加入最少的边，使得最后得到的图为一个边双连通分支。所谓的边双连通分支，即不存在桥的连通分支。
- 可以求出所有的桥，把桥删掉。然后把所有的连通分支求出来，显然这些连通分支就是原图中的双连通分支。把它们缩成点，然后添上刚才删去的桥，就构成了一棵树。在树上添边使得树变成一个双连通分支即可。

POJ 3352 Road Construction

- 本题只要求输出一共需要添加多少条边，而不需要求具体的方案。其实可以统计度为1的叶子节点（设共有x个），然后直接输出(x+1)/2即可

命题：一棵有n(n>=2)个叶子结点的树，至少(只需）要添加ceil(n/2)条边，才(就)能转变为一个没有桥的图。或者说，使得图中每条边，都至少在一个环上。

证明：

这里只证明n为偶数的情况。n为奇数的证明类似。

先证明添加n/2条边一定可以达成目标。

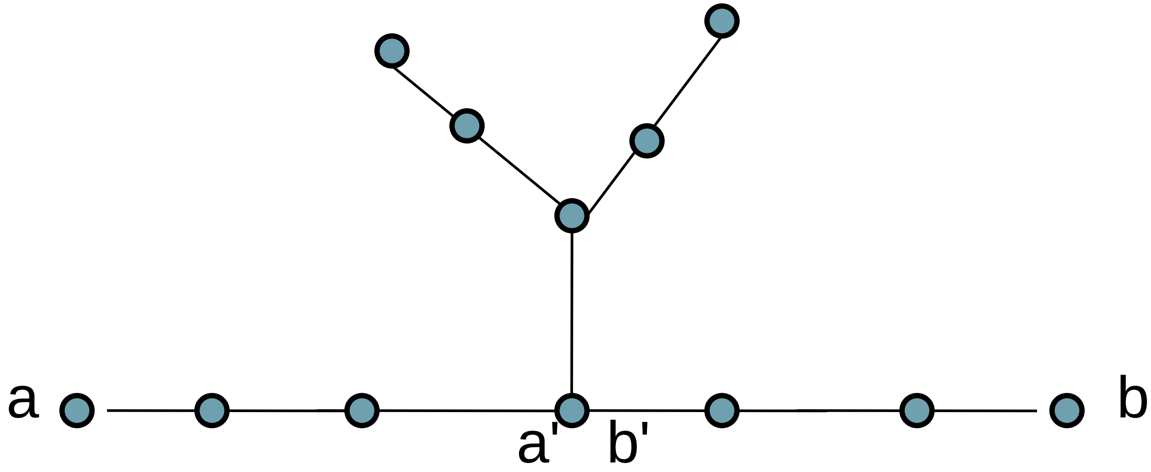
n=2时，显然只需将这两个叶子间连一条边即可。命题成立。
设n=2k(k>=1)时命题成立，即AddNum(2k)=k。下面将推出n=2(k+1)时命题亦成立

n=2k+2时，选取树中一条迹(无重复点的路径），设其端点为a,b；并设离a最近的度>=3的点为a',同理设b'。
（关于a‘和b’的存在性问题：由于a和b的度都为1,因此树中其它的树枝必然从迹<a,b>之间的某些点引出。否则整棵树就是迹<a,b>，n=2<2k+2，不可能。）

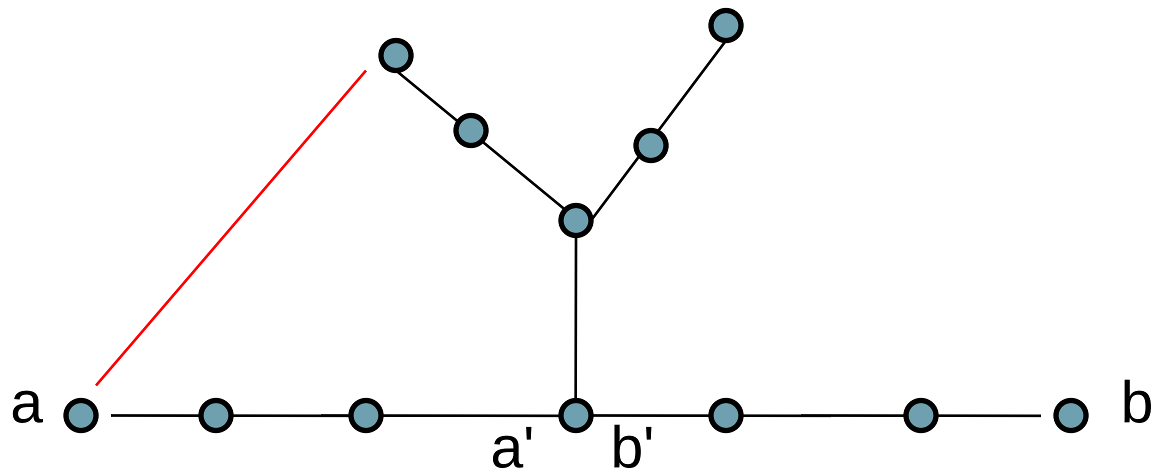
a’ b’不重合时：

在a,b间添一条边，则迹<a,b>上的所有边都已不再是桥。这时，将刚才添加的边，以及aa‘之间，bb’之间的边都删去，得到一棵新的树。因为删去的那些边都已经符合条件了，所以在之后的构造中不需要考虑它们。由于之前a‘和b’的度>=3，所以删除操作不会使他们变成叶子。因此新的树必然比原树少了两个叶子a,b，共有2k个叶子。由归纳知需要再加k条边。因此对n=2k+2的树，一共要添加k+1条边。

a’ b’ 重合时：



a’ b’ 重合时：



将a和一个非b的叶子节点x连上，然后将环缩点至 a’。
因为叶子节点是偶数，所以必然还存在一个非b非x的叶子节点不在环上， 因此a’不会变成叶子节点，于是新图比原图少2个叶子节点。

再证明n/2是最小的解。

显然，只有一个叶子结点被新加的边覆盖到，才有可能使与它相接的那条边进入一个环中。而一次加边至多覆盖2个叶子。因此n个叶子至少要加n/2条边。
证毕。

其他题目

- acm1236， acm3180， acm2762（强连通+拓扑排序），acm2553，acm3114（强连通+dijkstra)，acm3160(强连通+DP）