§6.3 有向图的强连通性

一．定义：

设是一个有向图。我们可以把划分成等价类如下：顶点v和w是等价的当且仅当存在一条从v到w的有向路径(简称路径)和一条从w到v的路径。设是连接中顶点对的边的集合。图称为G的强连通分支。一个图称为是强连通的，是指它只包含一个强连通分支。强连通分支的深度优先搜索生成树的根称为该强连通分支的根。

二．算法的理论依据

引理6.5：设是有向图G的一个强连通分支，设是G的深度优先搜索生成森林。那么的所有顶点和中的边构成一棵树。

证明：设v和w是中的顶点(假设每个顶点以深度优先搜索序号命名)。不失一般性，假设vw。由于v和w在同一强连通分支中，中存在从v到w的路径P。设x是P上标号最小的顶点(可能xv)。当P到达x的某个后代以后，它不可能离开x的后代构成的子树，这因为能离开那棵子树的边是到达一个标号比x小的顶点的横跨边或回边(因为x的后代是从x开始连续地标号的，离开该子树的横跨边或回边必然到达一个标号比x小的顶点)。因此w是x的一个后代。由于顶点按先序遍历的顺序标号，标号在x和w之间的所有顶点都是x的后代。因为xvw，故v是x的后代。

我们已证中任何两个顶点有一个共同的祖先在中。设r是中顶点的公共祖先中标号最小的那个顶点。如果v在中，那么在深度优先生成树中，从r到v的路径上的任何顶点也在中。本引理得证。

\*有向图G的强连通分支可以通过深度优先搜索找到各分支的根，然后根据查找各个根的逆顺序找到各个分支。设是各个分支的根，按照深度优先搜索查找这些结点终止的顺序列出(即，对的查找比对的查找先终止)。那么对每个i j ，或者在的左边，或者是的后代。

设是以为根的强连通分支()。那么由的所有后代组成，因为不存在使得是的后代。

引理6.6：对每个，由以下顶点组成，它们是的后代，但它们不是的顶点。

证明：根不可能是的后代，这因为调用SEARCH()在SEARCH()之后终止。

定义LOWLINK如下：

LOWLINK[v]=MIN({v}{w | 存在横跨边或回边从v的后代到w,并且

包含w的强连通分支的根是v的一个祖先}) （6.3）

图6.3

引理6.7：设是一个有向图。顶点是的一个强连通分支的根当且仅当。

证明：仅当）设v是G的一个强连通分支的根。由LOWLINK的定义，

LOWLINK[v] v。假设LOWLINK[v] v，那么存在顶点w和r使得：

1. w由一条从v的后代发出的横跨边或回边到达，
2. r是包含w的强连通分支的根，
3. r是v的祖先，并且
4. w v。

由条件2，r是w的祖先，因此，r w。因而由条件4，r v和条件3蕴含r是v的真祖先。但r和v必然在同一强连通分支中，这因为G中存在路径从r到v，和路径从v到w再到r。因此，v不是强连通分支的根，与假设矛盾。从而LOWLINK[v] = v。

当）设LOWLINK[v] = v。如果v不是包含v的那个强连通分支的根，那么v的某个真祖先r是根。因此存在路径P从v到r。考虑P中第一条从v的后代到一个不是v的后代的顶点w的边。这条边或者是到v的一个祖先的回边，或者是到一个标号比v小的顶点的横跨边。在两种情况下，都有w v。

剩下的工作要证r和w在的同一强连通分支中。因为r是v的祖先，因而一定存在从r到v的路径。而路径P从v到w到r。从而r和w在同一强连通分支中。故LOWLINK[v] w v，矛盾。

三．算法：

输入：一个有向图；

输出：G的所有强连通分支的表。

方法：

BEGIN

COUNT := 1;

FOR all v in V DO mark v “new”;

Initialize STACK to empty;

WHILE there exists a vertex v marked “new” DO

SEARCHC(v);

END;

PROCEDURE SEARCHC (v);

BEGIN

1. mark v “old”;
2. DFNUMBER[v] := COUNT;（序号）
3. COUNT := COUNT + 1;
4. LOWLINK[v] := DFNUMBER[v];
5. push v on STACK;
6. FOR each vertex w on L[v] DO（v指向的w）
7. IF w is marked “new” THEN

BEGIN

1. SEARCHC(w);
2. LOWLINK[v] := MIN( LOWLINK[v], LOWLINK[w] );

END（w是v的儿子，）

ELSE

1. IF (DFNUMBER[w] < DFNUMBER[v]) AND w is on STACK

THEN（横跨边或回边，w的强连通分支还没有找完，根节点一定是v的祖先）

1. LOWLINK[v] := MIN( DFNUMBER[w], LOWLINK[v] );
2. IF LOWLINK[v] = DFNUMBER[v] THEN

BEGIN

13. REPEAT

14. pop x from top of STACK;

15. print x;

UNTIL x = v;（x到v即是一个强连通分支）

16. print “end of strongly connected component”;

END

END;

四．算法分析

最坏情况下算法的时间复杂度为O(MAX(n, e))，其中n和e分别为有向图的顶点数和边数。

对SEARCHC(v)的一次调用，除递归调用外，所花时间与v发出的边数成正比。由于SEARCHC对每个顶点只调用一次，所有SEARCHC调用所花的时间与边数成正比。主程序中将所有顶点标记为“new”所花时间与n成正比。故得以上结论。

五．算法的正确性

定理6.8：上述算法正确地找出有向图G的所有强连通分支。

证明：以下对已经终止的对SEARCHC的调用的次数归纳证明：当SEARCHC(v)终止时，LOWLINK被正确地计算。由SEARCHC的12-16行，v成为一个强连通分支的根，当且仅当LOWLINK[v] = v。而且打印出来的顶点恰好是v的那些后代，它们不属于根在v之前已发现的那些强连通分支。即栈中v以上的顶点是v的后代，它们的根在v之前还没有被发现，这因为它们还在栈中，正如引理6.6所要求的那样。

要证明LOWLINK正确地计算，注意在SEARCHC中有两个地方，LOWLINK[v]取得小于v的值，即第9行和第11行。在第1个情况下，w是v的儿子，且LOWLINK[w] < v。那么存在顶点x = LOWLINK[w],它由w的后代y发出的一条横跨边或回边到达。而且包含x的强连通分支的根r是w的祖先。因为x < v，我们有r < v，因此，r是v的真祖先，故LOWLINK[v] LOWLINK[w]。

在第2种情况下(即第11行)，有一条横跨边或回边从v到w < v，包含w的强连通分支C还没有被发现。对C的根r调用SEARCHC还没有终止。因而r必然是v的一个祖先。(因为r w < v，或者r在v的左边，或者r是v的祖先。但如果r在v的左边，SEARCHC(r)已经终止了)。因此，LOWLINK[v] w。

我们还要证SEARCHC计算的LOWLINK[v]取到它应该取的最小值。假设存在v的后代x，x发出一条横跨边或回边到y，并且包含y的强连通分支的根r是v的一个祖先。我们证明LOWLINK[v] y。

情形1：x = v。我们由归纳假设和引理6.7，假设目前已找到的强连通分支都是正确的。那么y必然仍在STACK中，因为SEARCHC(r)还未终止。故11行置LOWLINK[v]为y或更小。

情形2：x v。设z是v的一个儿子，使得x是z的后代。由归纳假设，当SEARCHC(z)终止时，LOWLINK[z]已被置成y或更小。第9行, LOWLINK[v]至少被置成这么小，或更小。

作业12：

1. 用深度优先搜索策略设计一个O(e)时间的算法，判定一个有向图是否无圈(无有向圈)。

在图的深度遍历中，我们将图中每个节点设置为三个状态：-1,0,1，分别表示还没发现的节点；已经发现但是还没处理完的节点；已经处理完的节点。对应上述思路，假设我们正在处理v，那么v的状态为0，其余正在处理节点的状态也为0，如果从状态为0的节点遍历到状态为0的节点就存在环。

法二：本方法与法一非常相似，方法一中存在三种情况，还未入栈时表示未被访问过的点；在栈中时表示已经被访问过但是还没有递归结束；从栈中出栈时表示递归结束，即后代也全部被访问过了。上述三种情况分别用 -1，0，1三种状态来标记点。  
 针对上述思路，假设正在处理点v，那么v的状态是0，其余正在处理的结点的状态也是0，如果从状态0的结点遍历到状态为0的结点，那么就存在环。此时所有状态为0的结点构成了一个环！发现存在环时遍历输出state数组即可，不过该方法输出时不是按照环路的顺序输出；如果需要顺序输出环路，可增加一个cycle数组，每次记录环路的起点位置i。用cycle[i]记录结点i的下一个结点编号。利用递归的方式输出cycle数组即可得到回路顺序。

判断有向图是否有圈的rule—— 一个有向图是无圈图当且仅当它没有背向边。对边的处理（引入背向边的定义）： 如果当我们处理（v, w）时 发现w是未被标记的， 或当我们处理（w, v）时发现v是未标记的，那么我们就用树的一条边表示它； 如果当我们处理 （v, w）时发现w 是已被标记的， 并且当我们处理（w, v）时发现v 也是已标记的， 那么我们就画一条虚线， 并称之为背向边，表示这条边实际上不是树的一部分；