## §3.4 有向图的强连通性

### 一、定义

设是一个有向图。我们可以把V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3.4：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),见图3.7。不失一般性，假设。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)，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，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是中的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。证毕。

\*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，或者的左边，或者的后代。

设是以为根的强连通分支。那么由的所有后代组成，因为不存在使得的后代。

**引理3.5：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。证毕。

**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}) (3.3)

**引理3.6：设G是一个有向图。顶点v是G的一个强连通分支的根当且仅当LOWLINK[v] = v。**

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]。假设。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。

由条件2，r是w的祖先，因此，。因而由条件4，，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的一个真祖先的回边，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。

剩下的工作要证r和w在G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]，矛盾。 证毕。

### 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

mark v “old”;

DFNUMBER[v] := COUNT;

COUNT := COUNT+1;

LOWLINK[v] := DFNUMBER[v];

push v on STACK;

FOR each vertex w on L[v] DO

IF w is marked “new” THEN

BEGIN

SEARCHC (w);

LOWLINK[v] := MIN(LOWLINK[v], LOWLINK[w]);

END

ELSE

IF DFNUMBER[w]<DFNUMBER[v] and w is on STACK THEN

LOWLINK[v] := MIN(DFNUMBER[w],LOWLINK[v]);

IF LOWLINK[v] = DFNUMBER[v] THEN

BEGIN

REPEAT

pop x from top of STACK;

print x;

UNTIL x = v;

print “end of strongly connected component”;

END

END;

## §3.5 找图中最短圈

\*用宽度优先搜索的方法

PROCEDURE BFS1 (i : integer; adjlist : 图的邻接表；VAR Mincyclelength : integer);

VAR

queue, FATHER, BRANCH : ARRAY [1max] OF integer;

visit : ARRAY [1max] OF boolean;

j, hp, tp, x, x’, y, y’, , branch, cyclelength : integer;

p : 顶点指针；new : boolean;

BEGIN

FOR j := 1 TO max DO visit[j] := false;

mincyclelength := ; new := false;

hp := 1; tp := 1;

visit[i] := true; queue[hp] := i; /\*访问顶点i\*/

p := adjlist[i]; branch:= 1;

WHILE pNIL DO /\*访问根结点i的所有相邻点\*/

BEGIN

x := p^.vertex; FATHER[x] := i;

BRANCH[x] := branch; branch := branch+1;

/\*每个儿子一个分支\*/

visit[x] := true;

tp := tp+1; queue[tp] := x; p := p^.link;

END;

hp := hp+1;

REPEAT /\*访问i的所有儿子的后代\*/

p := adjlist[queue[hp]];

WHILE pNIL DO

BEGIN

x := p^.vertex;

IF NOT visit[x] THEN

BEGIN

FATHER[x] := queue[hp];

BRANCH[x] := BRANCH[FATHER[x]];

visit[x] := true; tp := tp+1; queue[tp] := x;

END

ELSE IF BRANCH[x]BRANCH[queue[hp]] THEN

BEGIN

y := queue[hp]; x’ := x; y’ := y;

cyclelength := 1;

WHILE x DO

BEGIN

cyclelength := cyclelength+1;

x := FATHER[x];

END;

WHILE y DO

BEGIN

cyclelength := cyclelength+1;

y := FATHER[y];

END;

END; /\* WHILE \*/

IF cyclelength < mincyclelength THEN

BEGIN

new := true;

mincyclelength := cyclelength;

END;

p := p^.link;

END;

IF new THEN

BEGIN /\*找到包含i的最短圈，打印该圈\*/

WRITE(“(”); new := false;

WHILE () OR () DO

BEGIN

IF THEN BEGIN WRITE(“(”);

END;

IF THEN BEGIN WRITE(“()”);

END;

END;

END;

hp := hp+1;

UNTIL hp > tp;

END;

主程序：

BEGIN

minlength := ;

FOR i := 1 TO |V| DO

BEGIN

BFS1(i, adjlist, mincyclelength);

IF mincyclelength < minlength THEN

minlength := mincyclelength;

END;

WRITE(“The length of minimum cycle is”, minlength);

END;

## §3.6 贪心算法与连线问题

问题：假设要建造一个连接若干城镇的铁路网络。已知城镇和之间直通线路的造价为，试设计一个总造价最小的铁路网络。这个问题名为连线问题。

把每个城镇看作是具有权的赋权图G的顶点，问题转化为：在赋权图G中，找出具有最小权的连通生成子图。由于权是造价，当然是非负的，所以最小权生成子图是G的一棵生成树T。赋权图的最小权生成树称为最优树。

例3.5：最小生成树的例子：(见图3.8)

Kruskal算法：

选择连杆，使得尽可能小。

若已选定边则从中选择，使

(i)为无圈图；

(ii)是满足(i)的尽可能小的权。

当第2步不能继续执行时，则停止。

\*以例3.5的图为例，讲解算法。

\*Kruskal算法本质上是贪心算法，贪心算法不一定总能求出最优解，必须证明算法的正确性。

算法正确性证明：

##### 定理3.7：由Kruskal算法构作的任何生成树都是最优树。

证：用反证法。对G的任何异于的生成树T，用f(T)记使不在T中的最小i值。现在假设不是最优树，T是一棵使f(T)尽可能大的最优树。

假设；也就是说，同时在T和中，但不在T中。则包含唯一的圈C。设是C的一条边，它在T中而不在中，的割边，因此，是具有

条边的连通图，所以它是G的另一棵生成树。显然

在Kruskal算法中选出的边，是使为无圈图的权最小的边。由于是T的子图，它也是无圈的。于是得到：

结合(3.4)式和(3.5)式，有

所以T’也是一棵最优树，然而

，

与T的选法矛盾。因此，，从而确实是一棵最优树。证毕。