§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。

第七章 算法设计技术

\*经过多年的实践和总结，我们找到几种为解决问题设计有效算法的通用的技术：分治策略，动态规划，贪心算法，回溯技术，分支定界法和局部查找。

通常我们设计算法时，首先考虑这几种技术能否设计出解决问题的有效算法。

§7.1 分治策略

一．分治策略的思想：

1. 思想：把大问题化成若干个较小的子问题，把子问题再化成更小的子问题，依此下去，直到每个子问题可以方便地求出解。而我们从较小的子问题的解可以方便地求出大问题的解。

2. 例子：

例1：河内塔问题：设有三个柱子A, B和C。C柱子上有n个盘子，从下往上依次从大到小。现在要把C柱子上的盘子全部搬到A柱子上，每次只允许把一个盘子从一个柱子搬到另一个柱子上，任何时候不允许把大盘子放到小盘子上。以上问题称为河内塔问题。

求解思想：先按某种方法将C柱子上个盘子搬到B柱子上，再将C柱子上第个盘子搬到A柱子上，再用上述方法将B柱子上的个盘子搬到A柱子上。

算法：

PROCEDURE Hanoi\_Tower (A, B, C, n);

/\* 把C柱子上的n个盘子移到A柱子上 \*/

BEGIN

IF n 1 THEN

BEGIN

Hanoi\_Tower (B, A, C, n1);

Move (A, B, C, n);

Hanoi\_Tower (A, C, B, n1);

END

END;

其中，Move(A, B, C, n)表示把盘子n从C移到A, 中间杆为B。

算法分析：设为上述算法的计算时间:

解得：为。

例2：求集合S的最大元和最小元。

算法：

PROCEDURE MAXMIN (S);

BEGIN

IF |S| = 2 THEN

BEGIN

设S；

RETURN(MAX(), MIN())

END

ELSE BEGIN

把S划分成两个子集和，和各含S的一半元素；

(max1, min1) := MAXMIN();

(max2, min2) := MAXMIN();

RETURN( MAX(max1, max2), MIN(min1, min2) );

END;

END;

算法分析：设为上述算法的元素比较次数

解得, 为2的幂。

即 为。

二．求两个长整数的乘积

1. 小学算术的解法：

时间复杂度为，为每个长整数的位数。

2. 分治策略解法

X: A B

Y: C D

这时，设为计算两个位数为n的整数的乘法所需的时间。则

解出是。

3. 改进方法：

上式可以改成

从而

解得 。

4. 算法：

FUNCTION mult (X, Y, n : integer) : integer;

/\* X和Y是小于的有符号数，n是2的幂 \*/

VAR

s : integer; /\* s中存放XY的符号 \*/

: integer;

A, B, C, D : integer;

BEGIN

s := sign(X) \* sign(Y);

X := abs(X);

Y := abs(Y);

IF n=1 THEN

IF (X=1) AND (Y=1) THEN

RETURN (s)

ELSE RETURN(0)

ELSE BEGIN

A := left n/2 bits of X;

B := right n/2 bits of X;

C := left n/2 bits of Y;

D := right n/2 bits of Y;

:= mult(A, C, n/2);

:= mult(AB, DC, n/2);

:= mult(B, D, n/2);

RETURN(s\*())；

END

END;

\*虽然是，但常数系数很大。当时，普通算术方法更快。

三．平衡子问题

1. 原则：用分治策略划分子问题时，尽可能将子问题化成等体积的。

2. 例子：

插入排序第i遍，可看作将一个长为i的文件与一个长为1的文件归并，因此，

解得 。

而归并排序则是将两个长为的文件归并，

解得，优于插入排序。

四．循环赛日程表

分治法不仅可以用来设计算法，而且在其它方面也有广泛的应用。例如：可以用分治思想来设计电路，构造数学证明等。下面举一个例子说明。

设有个运动员，要进行网球循环赛。现在要设计一个满足以下要求的比赛日程表：

1. 每个选手必须与其他个选手各赛一次；
2. 每个选手一天只能赛一次；
3. 循环赛一共进行天。

按分治策略，可以将所有选手分成两半，n个选手的比赛日程表可以通过n/2个选手设计的比赛日程表来决定。递归地用这种一分为二的策略对选手进行分割，直到只剩下2个选手时，只要让这两个选手进行比赛就可以了。

例如：安排8个选手的比赛日程表。

图7.1

其中，左上角与左下角的2个小块分别为选手1至4以及选手5至8前3天的比赛日程。据此，将左上角的小块中所有数字按其相对位置抄到右下角，将左下角小块中所有数字按其相对位置抄到右上角，这样就分别安排好选手1至4及选手5至8在后4天的比赛日程。

五．已学过的分治策略算法的例子：

1. 归并排序：

解得。

2. 快速排序：

最好情况：

解得: 。

3. 找第k个最小元

解得：。

4. 二分查找：

解得：。

作业13：

1. 设n个不同的整数从小到大排序后存于中。若存在下标，，使得 ，设计一个有效算法找到这个下标。要求算法在最坏情况下的计算时间为 。

二分搜索

1. 设是个元素的数组。对任一元素，设

。当时，称为T的主元素。试设计一个有效算法，确定T是否有一个主元素。其中算法的计算复杂性为。

S(x)是数组中值为x的下标集合

数组中存在一半以上的数相同，则该数是主元素

如果一个数列有主元素,那么必然是其中位数。求一个数列有没有主元素，只要看中位数是不是主元素。

先找出中位数（寻找第n/2小元），再计数