# 第三章 基本图论算法

\*本章我们介绍一些基本图论算法和常用的算法设计技术在图论算法中的应用。

## §3.1 图及其表示

### 一．图

1.图的定义：一个图，其中V是有穷、非空的顶点集，E为边集。如果边是有序的顶点对，则图G称为有向图。u称为该边的尾，v称为该边的头。如果边是无序的顶点对，则图G称为无向图。

例3.1：(见图3.1)

2. 几个概念

邻接

关联

入度，出度，度

一条有向(或无向)路径是一系列边。其中是从的边，。则称该路径是从到长度为的路径，用表示。

一条路径称为是简单的，是说该路径上所有顶点和边互不相同。一个起点和终点相同的简单路径称为一个圈。

图G中如果任意一对顶点之间有一条路径，则称G是连通的。

3. 图的表示方法

邻接矩阵：图3.1中(a), (b)两个图的邻接矩阵表示如图3.2中(a), (b)。

邻接表：图3.1中(a), (b)两个图的邻接表见图3.3中(a), (b)。

### 二．树

树：一个连通无圈图称为一棵树。

例3.2：见图3.4

有根树，二叉树，m叉树

## §3.2 深度优先搜索和宽度优先搜索

### 一．几个概念

1. 生成树：给定一个图，G的一棵生成树T是一棵树，且及。

2. 深度优先搜索：深度优先搜索的思想是，从任一顶点出发，找它的一个相邻未访问过的顶点作为下一个要访问的顶点，再从下一个顶点出发，找它的一个相邻未访问过的顶点作为下一个要访问的顶点，如此下去。如果当前顶点无相邻未访问过的顶点，则退到上一个顶点，找其它未访问过的相邻顶点作为下一个要访问的顶点。

3. 宽度优先搜索：宽度优先搜索的思想是，从任一顶点出发，先逐个访问完它的所有相邻顶点，再从每一个相邻顶点出发，访问它们的所有相邻点。

4. 深先搜索生成树和深先搜索生成森林：按深先搜索的方法遍历图，所得到的树和森林。

5. 回边和树边：

例3.3：见图3.5

### 二．深度优先遍历图的算法

递归算法：

输入：图和邻接表

输出：把E分成树边集T和回边集B.

PROCEDURE Search (v);

BEGIN

Mark v “old”;

FOR each vertex w on L[v] DO

IF w is marked “new” THEN

BEGIN

add (v, w) to T;

Search (w);

END

END;

主程序：

BEGIN

;

FOR all v in V DO mark v “new”;

WHILE there exists a vertex v in V marked “new” DO

Search (v);

END;

\*算法时间复杂性：O(max (n, e)): 其中 n =|V|, e = |E|。

2. 几个性质：

引理3.1：如果(v, w)是一条回边，那么在深度优先搜索生成森林中，v是w的祖先，或相反。

证明：不失一般性，v比w先被访问。当v被访问时，w仍标记为“new”,由Search(v)访问的所有标记为“new”的顶点在生成森林中都是v的后代，并且Search(v)在w被访问前不可能结束，这因为w在L[v]中，从而有本引理的结论。

\*深度优先搜索编号：DFNUMBER[v]

### 三．宽度优先遍历图的算法

PROCEDURE BFS (i: integer; adjlist: 图的邻接表；VAR visit: 访问标志数组)；/\*非递归宽度优先搜索遍历图；i：起始顶点标号\*/

VAR

queue: ARRAY[1··max] oF integer;

j, hp, tp : integer;

p : 顶点指针；

BEGIN

FOR j := 1 TO max DO

visit[j] := false;

hp:=1; tp:=1;

WRITE(“v”, i:1); /\*输出起始顶点\*/

visit[i] := true; /\*置起始顶点已访问标志\*/

queue[hp] := i; /\*起始顶点进队列\*/

REPEAT

p := adjlist[queue[hp]]; /\*取对列首顶点\*/

WHILE DO /\*当该顶点有相邻的顶点\*/

BEGIN

i := p^.vertex; /\*选取一个\*/

IF NOT visit[i] THEN /\*若未访问过\*/

BEGIN

WRITE (“v”, i:1);

visit[i] := true; /\*置已访问标志\*/

tp := tp+1;

queue[tp] := i; /\*该顶点进队列\*/

END;

p := p^.link;

END;

hp := hp+1;

UNTIL hp > tp;

END;

\*算法的时间复杂性：O(max (n, e)): 其中 n =|V|, e = |E|。

## §3.3 图的2-连通性

### 一．几个概念：

1. 割点：一个顶点a称为G的割点，是说：存在G的两个顶点v和w，a, v, w互不相同，并且G中任意一条从v到w的路径都包含顶点a。换句话说，从G中删除a及与a相关联的边，G将被分割成两个或更多的连通分支。

2. 2-连通性：图G称为是2连通的，是说：对G中任意3个不同的顶点v, w, a，存在一条从v到w的路径不包含a。无向连通图G是2连通的，当且仅当G中不含割点。

3. 2-连通分支：

\*即G中的极大子图，该子图是2-连通的。

定义图G的边集E的等价关系R。当且仅当或存在G中的圈包含。

等价关系R将E划分成等价类，任意两条边属于同一等价类当且仅当这两条边同在某一圈上。对令为中边的端点的集合，则称为G的2-连通分支。见图3.6

### 二．2-连通分支和割点的性质

引理3.2：对为连通无向图的2连通分支。那么

对每个，是2连通的；

对任意至多含一个顶点；

a是G的割点当且仅当对某两个。

证明：1.假设中存在3个不同顶点v, w, a，使得中所有从v到w的路径都经过a，则不是中的边。因此存在中的边和，并且存在中的圈包含这两条边。由2-连通分支的定义，这个圈上的所有边和顶点都分别属于。故在中，v到w有两条内部不相交的路径，而a不可能同时包含在这两条路径中，矛盾。

2. 假设中有两个不同的顶点v和w。那么中存在圈包含v和w，中存在圈也包含v和w。且的边不相交。从而我们可以用的边构成一个圈C，C中既有的边，又有的边。因而中至少有一条边与中一条边等价，这就证明了不是两个等价类，矛盾。

3. 设a是G的一个割点。那么存在G中两个顶点v和w，v, w, a互不相同且每一条从v到w的路径包含a。因为G是连通的，故至少存在一条这样的路径。设是这条路径上两条与a关联的边。如果存在圈包含这两条边，则存在从v到w的路径不包含a。故和属于不同的2连通分支。故a属于它们的顶点集的交。

反过来，如果，那么存在边和分别属于。因为这两边不包含在任何圈中，故从x到y的每一条路包含a。故a是割点。

引理3.3：设是连通无向图，设是G的深度优先搜索生成树。顶点a是G的割点当且仅当或者

1. a是根结点且a有多于一个儿子；或者
2. a不是根，并且对于a的某个儿子s，不存在回边从s的任一后代(包括s本身)到a的任一真祖先。

证明：易证，一个根结点是割点当且仅当它有多于一个儿子。

假设条件2为真。设f是a的父亲。由引理3.1，每一条回边从一个顶点到它的祖先。因此，任意回边从s的后代v到v的祖先。由本引理的假设，这些回边不能到达a的真祖先。因此它们到达a或s的后代。从而每一条从s到f的路径包含a，即a是割点。

假设a是割点，但不是根。设x和y是不同于a的两个不同顶点，并且G中从x到y的每条路径包含a。则x和y中至少有一个(例如x)是a在S中的真后代。假若不然，S中存在从x到y的路径，该路径不包含a。设s是a的儿子使得x是s的后代(可能)。这时，或者不存在从s的后代r到a的真祖先w的回边，从而条件2成立；或者存在一条这样的回边。在后一个情形下，我们考虑两个情形：

情形1：设y不是a的后代。那么存在从x到v到w再到y的路径，该路径不含a，矛盾。

情形2：设y是a的后代。则y不是s的后代，否则存在从x到y的路径不含a。设s’是a的儿子，使得y是s’的后代。那么或者不存在s’的后代v’到a的真祖先w’的回边，从而条件2成立；或者存在这样一条回边。在后一种情形下，存在一条路径从x到v到w到w’到v’再到y，该路径不包含a,矛盾。从而条件2为真。 证毕。

### 三．几个定义：

设连通无向图的深度优先搜索生成树的树边集为T，回边集为B,并设V中顶点以深度优先搜索的序号命名。对任意，定义

LOW[v]=MIN({v}{w | 存在回边，使得x是v的后代，并且w是v的祖先}) (3.1)

\*由于深度优先搜索生成树中顶点标号是按前序遍历的顺序标号的，如果x是v的后代，并且是回边，满足，则w是v的真祖先。

由引理3.3，如果顶点v不是根，那么v是割点当且仅当v有一个儿子s，使得LOW[s]v。

LOW[v]可以通过确定以下顶点w的最小值来计算：

1. ，或
2. 且s是v的一个儿子，或
3. 是B中的一条回边。

(3.1)式等价于

LOW[v]=MIN({v}{LOW[s] | s是v的一个儿子}{w | (v, w)B}) (3.2)

### 四．算法

输入：一个连通无向图

输出：G的每个2-连通分支的边集

方法：

将集合T初始化为，COUNT初值为1。将V中每一个顶点标记为“new”。选择V中任一顶点，调用过程SEARCHB()，建立深度优先搜索生成树，并对V中每个v计算LOW[v]。

当SEARCHB的第5行遇到顶点w，如果边不在栈STACK中，则将其放入STACK。当在第10行发现一对顶点满足：w是v的儿子且LOW[w]后，把STACK从栈顶到(v, w)的所有边弹出来。这些边构成G的一个2-连通分支。

PROCEDURE SEARCHB (v);

BEGIN

Mark v “old”;

DFNUMBER[v] := COUNT;

COUNT := COUNT+1;

LOW[v] := DFNUMBER[v];

FOR each vertex w on L[v] DO

IF w is marked “new” THEN

BEGIN

add (v, w) to T;

FATHER[w] := v;

SEARCHB(w);

IF LOW[w]DFNUMBER[v] THEN

a biconnected component has been found;

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

END

ELSE IF w is not FATHER[v] THEN

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

END;

### 五．算法分析

步骤1的时间为O(e)，按3.2节的Search过程的分析，可得本结论。步骤2考察每一条边一次，将其放入栈中，然后弹出来，故步骤2的时间为O(e)。总的时间为：O(e).