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$G=(V,E)$，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(u,v)，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(u,v)，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$(v_1,v_2),(v_2,v_3),\cdots,(v_{n-1},v_n)$。其中$(v_{i-1},v_i)$是从$v_{i-1}$到$v_i$的边，$i=2,3,\cdots,n$。则称该路径是从$v_1$到$v_n$长度为$n-1$的路径，用$v_1v_2\cdots v_n$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$G=(V,E)$，G 的一棵生成树 T 是一棵树，且$V(T)=V(G)$及$E(T)\subseteq E(G)$。

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$G=(V,E)$和邻接表$L[v], v \in V(G)$
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
T := ∅;
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 p ≠ NIL 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。$\forall e_1, e_2 \in E, e_1 R e_2$当且仅当$e_1 = e_2$或存在 G 中的圈包含

e_1和e_2。

等价关系 R 将 E 划分成等价类$E_1, E_2, \cdots, E_k$，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$1 \le i \le k$，令V_i为E_i中边的端点的集合，则$G_i = (V_i, E_i)$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$1 \le i \le k$，设$G_i = (V_i, E_i)$为连通无向图$G = (V, E)$的 2 连通分支。那么
对每个$i, 1 \le i \le k$，G_i是 2 连通的；
对任意$i \ne j, V_i \cap V_j$至多含一个顶点；
a 是 G 的割点当且仅当对某两个$i \ne j, a \in V_i \cap V_j$。

证明：1.假设V_i中存在 3 个不同顶点 v, w, a，使得G_i中所有从 v 到 w 的路径都经过 a，则(v, w)不是G_i中的边。因此存在E_i中的边(v, v')和(w, w')，并且存在G_i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E_i和V_i。故在G_i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$V_i \cap V_j$中有两个不同的顶点 v 和 w。那么G_i中存在圈C_1包含 v 和 w，G_j中存在圈C_2也包含 v 和 w。且C_1与C_2的边不相交。从而我们可以用C_1和C_2的边构成一个圈 C，C 中既有E_i的边，又有E_j的边。因而E_i中至少有一条边与E_j中一条边等价，这就证明了E_i和E_j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(x, a)和(y, a)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(x, a)和(y, a)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$a \in V_i \cap V_j$，那么存在边(x, a)和(y, a)分别属于E_i和E_j。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$G = (V, E)$是连通无向图，设$S = (V, T)$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$x = s$)。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(v, w)。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(v', w')。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$G = (V, E)$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$v \in V$，定义

LOW[v]=MIN({v}∪{w | 存在回边$(x, w) \in B$，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(x, w)是回边，满足$w < v$，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. $w = v$，或
2. $w = LOW[s]$且 s 是 v 的一个儿子，或
3. (v, w)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$G = (V, E)$

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v_0，调用过程 SEARCHB(v_0)，建立深度优先搜索生成树$S = (V, T)$，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(v, w)不在栈 STACK 中，则将其放入 STACK。当在第 10 行发现一对顶点(v, w)满足：w 是 v 的儿子且 LOW[w]≥ v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).