求有向图的所有 Euler回路算法

牟 廉 明

(四川省高等学校数值仿真重点实验室 /内江师范学院 数学系, 四川 内江 641112)

摘 要: 首先利用图的深度优先搜索方法给出了有向图为强连通图的判定算法, 然后利用图的广度优先搜索方法给出了有向图是欧拉图和有向边是桥的判定算法, 最后给出了求有向图的所有欧拉回路算法, 并通过实例验证了算法的有效性. 从而有效地解决了欧拉回路的判定、计数和求解问题.

关键词: Euler回路; 回溯法; 深度优先搜索; 广度优先搜索

中图分类号: O157. 5 文献标识码: A 文章编号: 1671- 1785(2008) 02- 0011- 04

1 引言

经过有向图中所有弧一次且仅一次的有向回路称为有向欧拉回路, 含有欧拉回路的图称为欧拉图[1]. 欧拉回路在计算机鼓轮设计 (模数转换)、中国邮递员等问题中有广泛应用. 对于欧拉回路问题, 从四个层次来理解: 一是如何判断一个有向图中是否含有欧拉回路? 二是如何找出一条欧拉回路? 三是存在多少欧拉回路? 四是如何找出所有欧拉回路? 对于第一个问题欧拉已经给出判定的充要条件[1], 第二个问题数学家 Fleury在 1921年给出了求解算法[2], 对于第三和第四个问题却很少有人讨论,
Tom~》Dvo r~k[3- 4]研究了特殊的完全二部图的欧拉回路构造问题. 本文全面深入探讨欧拉回路的四个问题, 首先借助图的深度优先搜索方法 (DFS)设计了有向图是强连通图的判定算法, 然后利用图的广度优先搜索方法[5] (BFS)给出了有向图是欧拉图和有向边是桥的判定算法, 最后利用 Fleury算法和回溯法的思想给出了求解有向图的所有欧拉回路算法, 并用实例验证算法的有效性.

2 强连通有向图的判定算法

根据有向欧拉图判定的充要条件, 有向图存在欧拉回路的前提条件是有向图为强连通图. 下面利用有向图的 DFS算法和标号法给出有向图为强连通图的判定算法. 其主要思想是在同一有向回路上的标号相同, 且等于回路上最小的访问次序. 记: visited(u)为顶点 u是否被访问, father(u)为顶点 u的父结点, stack(u)为 u是否在栈 S中, num(u)为被访问的次序; mark(u)为 u的标号, out(u)为顶点 u的出边集中未被访问的边集; S. top( )为取栈顶, S. push为入栈, S. pop( )为出栈. 在以下每个算法中含义相同.

算法 1 判断一个有向图 D是否为强连通图算法 JudgeCon(D)

```
输入: 有向图 D
输出: 是否为强连通图
Begin
1. while( u∈ V(D)) { //对所有顶点和标号变量初始化.
2.    visited( u) = 0;  father( u) = 0;  stack( u) = 0; }
3. while( e∈ E(D)) { //对所有边初始化为未访问
4.    visited( e) = 0; }
5.    i= 1; C= φ,  initialize( S); //初始化变量和栈
6. 任取一顶点 r∈ V(D) {
      u= r;  num( u) = i;  mark( u) = i;  S. push( u);  stack
( u) = 1; }
7. while( S≠ φ) {
8.    u= S. top( );
9.    while( out( u) ≠ φ) { //u存在未被访问的出边;
10     任取弧 < u, v> ∈ out ( u);  visitd( < u, v> ) = 1;
11     if visited( v) = 0{
12     i= i+ 1;  num( v)= i;  mark( v) = i;  father( v) = u;
       visited( v) = 1;  S. push( v);  stack( v) = 1;  u= v; }
13     else {
14        if num( v) < num( u) && stack( v) = 1 {
15        mark( u) = min{mark( u), num( v) }}  //End of if
16    } //End of while
17 u= S. top( );
18 if mark( u) = num( u) { //标号等于次序, 进栈并判断
19    C= C∪ { u};
20    if C= V( D) { return( D是强连通图); }
21    else { return( D不是强连通图); } } //End of if
22 if father( u) ≠ 0{
23    mark( father( u)) = min{mark( father( u)), mark( u) };
```

收稿日期: 2007- 10- 22

基金项目: 国家自然科学基金资助项目 (10472042, 10672151), 四川省教育厅青年基金资助项目 (2004B020), 内江师范学院重点课程基金.

作者简介: 牟廉明 (1971-), 男, 重庆万州人, 内江师范学院副教授, 硕士. 主研方向: 计算智能, 数据挖掘.

```
24.  C= C∪ S pop(); }
25. else {
26.    if(∃ v∈ V(G) && visited(v) = 0) {
27.      return(D不是强连通图 ); } //End of if
28. } //End ofwhile
END
```

根据深度优先搜索的特点,该算法的时间复杂度为 O(|E(D)|), E(D)为有向图 D的边集.

3 有向欧拉图判定算法

在有向图 D是欧拉图的充要条件中不仅要求 D是强连通图,还要求每个顶点的入度都等于出度.因为强连通图的广度优先搜索方法(BFS)所经过的边构成一棵包含所有顶点的外向树.下面利用有向图的 BFS来构造判定算法.记: $d^-(u)$为顶点 u的入度, $d^+(u)$为顶点 u的出度, $\tau^+(u)$为顶点 u未被访问的后继顶点集合, Q. push为入队, Q. pop()为出队, Q. front为取队头.在以下每个算法中含义相同.

算法 **2** 判断一个强连通图 D是欧拉图算法 JudgeEuler(D)

输入:强连通图 D

输出:是否为欧拉图

```
Begin
1. while( u∈ V(D) ) {  //所有顶点标记为未访问
2.    visited(u) = 0; }
3. initialize(Q);  //初始化队列
4. 任取一顶点 u∈ V(D) {
5.    visited(u) = 1; //标记为已访问
6.    if d⁻(u) = d⁺(u) { Q. push(u);  }
7.    else{  return(D不是欧拉图 ); } }
8. while(Q≠φ) {
9.    v= Q. front( ); //取队头元素
10.    Q. pop( ); //队头元素出队
11.      while(τ⁺(v)≠φ) {
12.       visited(w) = 1;  //w∈ τ⁺(v)标记为已访问
13.       if d⁻(w) = d⁺(w) { Q. push(w);  }
14.       else{ return(D不是欧拉图 ); }
15.    } //End ofwhile
16. } //End ofwhile
17. return(D是欧拉图 );
End
```

根据有向图的广度优先算法的特点,该算法的时间复杂度为 O(|V(D)|),其中 V(D)为 D的顶点集.

4 有向边为桥的判定算法

Fleury算法的本质就是在边的遍历过程中要尽量"避桥",因此下边给出判断有向边为桥的算法.有向图 D_1中的一边有向边 <u, v>是否为桥,即判断是否存在从 v到 u的有向通路,如果没有就是桥,有就不是桥.而且以 v为起点的广度优先搜索所对应的外向树即反映了从 v到其余各顶点的有向路径.所以用 BFS算法构造判定算法如下.

算法 **3** 判断有向图 D_1中有向边 <u, v>是否为桥的算法, JudgeBridge(D_1, <u, v>)

输入:有向图 D_1,有向边 <u, v>

输出: <u, v>是否为桥

```
Begin
1. if ( d⁻ (u) = 0)  { //只能走这边,故判定为不是桥
2.    return( <u, v>不是桥);  }
3. else {
4.    if ( d⁺ (v) = 0)  {  return( <u, v>是桥); } }
   //不满足前两种特殊情况就用下面的判定方法
5. while( ∈ V(D₁) )  {  visited(r) = 0;  } //标记 D₁中所有顶点未访问
6. initialize(Q);  //初始化队列
7. visited(v) = 1;  //标记起点 v为已被访问,从 v开始广度遍历
8. Q. push(v);   //进队
9. while(Q≠φ) {
10.    w= Q. front( );  //取队头
11.    Q. pop( );  //队头元素出队
12.    while(τ⁺(w)≠φ) {
13.       visited(s) = 1;  // s∈ τ⁺(v)标记为已访问
14.       if s= u { return( <u, v>不是桥);  }
15.       else{ Q. push(s);  } } End ofwhile
16. } End ofwhile
17. return( <u, v>是桥 );
End
```

根据有向图的广度优先算法的特点,该算法的时间复杂度为 O(|V(D_1)|),其中 V(D_1)为 D_1的顶点集.

5 求有向欧拉图的所有欧拉回路算法

算法 1和算法 2给出了判断一个有向图是否为欧拉图的方法,算法 3给出了判断有向边是否为桥的算法,下面利用回溯法思想构造求解有向欧拉图的所有欧拉回路算法.记: $e^+(S, u)$为顶点 u的出边不在栈 S中的集合; getfrom(e)为取有向边 e的起点; getto(e)为取有向边 e的终点; $e^+(D_1, k)$为顶点 k在有向图 D_1中的出边集; next($e^+(D_1, k)$)为取边集 $e^+(D_1, k)$的下一条边.

算法 **4** 求有向欧拉图的所有欧拉回路算法 EulerCycles(D)

输入:有向欧拉图 D

输出:所有的欧拉回路

```
Begin
1. while( e∈ E(D) ) {
2.    visited(e) = 0;  stack(e) = 0; } //初始化
3. initialize(S);  D₁= D;  //初始化变量和栈
4. 任取一顶点 u∈ V(D)作为起点;
5. while( e⁺(S, u)存在未被访问的边 ) {
6.    取一条 e⁺(S, u)中未被访问的边 <u, v>;
7.    s push( <u, v>;  visited( <u, v> ) = 1;  stack( <u, v> ) = 1;
      D₁= D₁- { <u, v> };  e⁺(D₁, u) = e⁺(D₁, u) - { <u, v> };
```

```
8. While (S≠φ) {
9.    e= S. top  //取栈顶元素
10.   k= getto(e);
11.   while( e+ (D1, k)≠φ) {
12.      < k, t> = first( e+ (D1, k)); //取第一条边
13.      if JudgeBridge(D1, < k, t> ) {< k, t> = next( e+
(D1, k) ); }
14.      else { //边不是桥就进栈
15.      S. push( < k, t> ); Visited( < k, t> ) = 1; stack
( < k, t> ) = 1;
          e+ (D1, k) = e+ (D1, k) - {< k, t> }; D1 = D1
- {< k, t> }; }
16.      } //End of if
17.   } //End of while
18.   output(S) //所有边进栈,输出一条有向欧拉回路
19.   m= S. top(); p= getfrom(m); //取栈顶元素
20.   While (e+ (S, p)中未被访问的边集为空) {
21.      m= S. pop(); p= getfrom(m); //退栈
         e+ (S, p)中的边恢复为未访问;
         visited(m) = 0; stack(m) = 0; e+ (D1, p) = e+
(D1, p) ∪ {m};
         D1 = D1∪ {m}; } End of while
22.      if S≠φ{取 e+ (D1, p)中一条未被访问的边入栈; }
23.      }End of while
24.   }End of while
25. }End of while
```

该算法的时间复杂度为 O(|E(D) |* |m(D) |* |V(D) |), 其中 m(D)为 D 中有向欧拉回路的条数.

6 实例分析

计算机鼓轮设计就是要将鼓轮表面分成 16个扇区, 每块扇区用导体或绝缘体制成, 四个触点与扇区接触时, 接触导体输出 1, 接触绝缘体输出 0 鼓轮顺时针旋转, 触点每转过一个扇区就输出一个二进制信号. 问鼓轮上的 16个扇区应如何安排导体或绝缘体, 使得鼓轮旋转一周, 触点输出一组不同的二进制信号? 按照题目要求, 鼓轮的 16个位置与触点输出的 16个四位二进制信号应该一一对应, 亦即 16个二进制数排成一个循环序列, 使每四位接连数字所组成的 16个四位二进制子序列均不相同. 这个循环序列通常称为笛波滤恩序列. 该问题对应着在图 1中找出有向欧拉回路, 由回路中每条边对应码的第一个符号构成的循环序列就是所求结果.

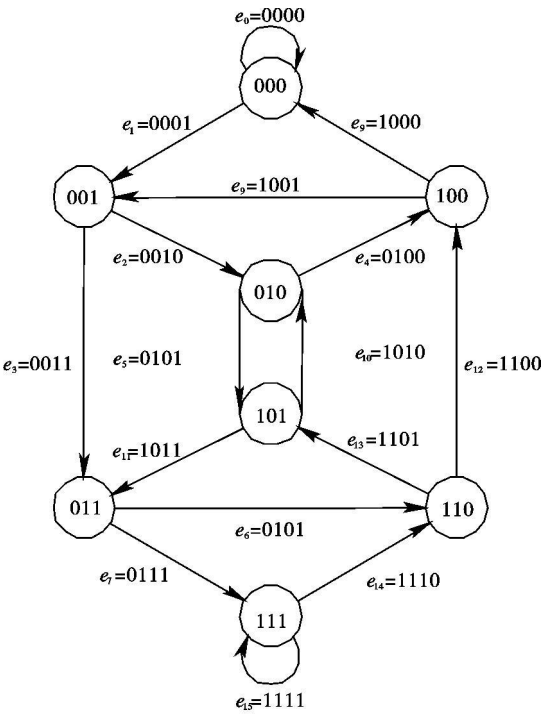

图 1　有向图

根据算法 1和算法 2利用 MATLAB编程判断该图为有向欧拉图, 根据算法 4利用 MATLAB编程算出该问题所有的 16种设计方案, 如表 1所示.

表 1　欧拉回路

序号	有向欧拉回路																对应设计方案
1	e_0	e_1	e_3	e_7	e_{15}	e_{14}	e_{13}	e_{11}	e_6	e_{12}	e_9	e_2	e_5	e_{10}	e_4	e_8	0000111101100101
2	e_0	e_1	e_3	e_7	e_{15}	e_{14}	e_{13}	e_{10}	e_5	e_{11}	e_6	e_{12}	e_9	e_2	e_4	e_8	0000111101011001
3	e_0	e_1	e_3	e_7	e_{15}	e_{14}	e_{13}	e_{10}	e_4	e_9	e_2	e_5	e_{11}	e_6	e_{12}	e_8	0000111101001011
4	e_0	e_1	e_3	e_7	e_{15}	e_{14}	e_{12}	e_9	e_2	e_5	e_{11}	e_6	e_{13}	e_{10}	e_4	e_8	0000111100101101
5	e_0	e_1	e_3	e_6	e_{13}	e_{11}	e_7	e_{15}	e_{14}	e_{12}	e_9	e_2	e_5	e_{10}	e_4	e_8	0000110111100101
6	e_0	e_1	e_3	e_6	e_{13}	e_{10}	e_5	e_{11}	e_7	e_{15}	e_{14}	e_{12}	e_9	e_2	e_4	e_8	0000110101111001
7	e_0	e_1	e_3	e_6	e_{13}	e_{10}	e_4	e_9	e_2	e_5	e_{11}	e_7	e_{15}	e_{14}	e_{12}	e_8	0000110100101111
8	e_0	e_1	e_3	e_6	e_{12}	e_9	e_2	e_5	e_{11}	e_7	e_{15}	e_{14}	e_{13}	e_{10}	e_4	e_8	0001110010111101
9	e_0	e_1	e_2	e_5	e_{11}	e_7	e_{15}	e_{14}	e_{13}	e_{10}	e_4	e_9	e_3	e_6	e_{12}	e_8	0000101111010011
10	e_0	e_1	e_2	e_5	e_{11}	e_7	e_{15}	e_{14}	e_{12}	e_9	e_3	e_6	e_{13}	e_{10}	e_4	e_8	0000101111001101
11	e_0	e_1	e_2	e_5	e_{11}	e_6	e_{13}	e_{10}	e_4	e_9	e_3	e_7	e_{15}	e_{14}	e_{12}	e_8	0000101101001111
12	e_0	e_1	e_2	e_5	e_{11}	e_6	e_{12}	e_9	e_3	e_7	e_{15}	e_{14}	e_{13}	e_{10}	e_4	e_8	0000101100111101
13	e_0	e_1	e_2	e_5	e_{10}	e_4	e_9	e_3	e_7	e_{15}	e_{14}	e_{13}	e_{11}	e_6	e_{12}	e_8	0000101001111011
14	e_0	e_1	e_2	e_5	e_{10}	e_4	e_9	e_3	e_6	e_{13}	e_{11}	e_7	e_{15}	e_{14}	e_{12}	e_8	0000101001101111
15	e_0	e_1	e_2	e_4	e_9	e_3	e_7	e_{15}	e_{14}	e_{13}	e_{10}	e_5	e_{11}	e_6	e_{12}	e_8	0000100111101011
16	e_0	e_1	e_2	e_4	e_9	e_3	e_6	e_{13}	e_{10}	e_5	e_{11}	e_7	e_{15}	e_{14}	e_{12}	e_8	0000100110101111

结语

文中针对欧拉回路问题的四个方面应用有向图的 DFS和 BFS算法分别设计了强连通有向图判定算法, 有向欧拉图判定算法以及有向边为桥的判定算法, 在此基础上给出了求有向欧拉图的所有欧拉回路算法, 并对算法复杂度进行了分析, 比较彻底有效地解决了欧拉回路的判定、计数和求解问题. 对算法略加修改也可求半欧拉图的所有欧拉通路和无向

欧拉图的所有欧拉回路,为欧拉图的进一步应用奠定了基础.

参考文献:

[1] 耿素云,屈婉玲. 离散数学 [M]. 北京: 高等教育出版社, 2004.

[2] 王树禾. 离散数学引论 [M]. 合肥: 中国科学技术大学出版社, 2001.

[3] Tom´y Dvoř´k, Ivan Havel, Petr Liebl. Euler Cycles in K_{2m} Plus Perfect Matching [J]. Europ. J. Combinatorics, 1999, 20: 227-232.

[4] Tom´y Dvoř´k, Ivan Havel, Petr Liebl. Euler cycles in the complete graph K_{2m+1} [J]. Discrete Mathematics, 1997, 171: 89-102.

[5] 许卓群. 数据结构与算法 [M]. 北京: 高等教育出版社, 2004.

An Algorithm for Finding All Euler Cycles in Digraph

MOU Lian-ming

(Key Laboratory of Numerical Simulation of Sichuan Province// Department of Mathematics, Neijiang Normal University, Neijiang, Sichuan 641112, China)

Abstract Firstly the judge-algorithm whether a digraph is complete connected graph is designed by the depth-first-search algorithm of digraph. Secondly the judge-algorithm whether a digraph is Euler graph is designed, the judge-algorithm whether a directed edge is bridge is designed by the breadth-first-search algorithm of digraph. Lastly the algorithm to output all Euler cycles in the digraph is constructed. The algorithmic validity is validated by example. Judgment, count and output of Euler cycle are effectually solved.

Key words Euler cycle; Retractable method; depth-first-search; breadth-first-search

(责任编辑:胡　蓉)