Algorithms On Graphs

H.T. Lau

TAB Professional and Reference Books

Division of TAB BOOKS Inc.

Blue Ridge Summit, PA

Algorithms On Graphs

H.T. Lau

TAB Professional and Reference Books

Division of TAB BOOKS Inc.

Blue Ridge Summit, PA

A Petrocelli book

FIRST EDITION
FIRST PRINTING

Library of Congress Cataloging in Publication Data

Lau, H. T. (Hang Tong), 1952—
Algorithms on graphs / by H.T. Lau.
p. cm.

ISBN 0-8306-3429-0
1. Graph theory—Data processing. I. Title.
QA166.L38 1989
511′.5—dc20 89-36595
CIP

Questions regarding the content of this book should be addressed to:

Reader Inquiry Branch
TAB BOOKS Inc.
Blue Ridge Summit, PA 17294-0214

NOTE TO THE READER

Standard graph-theoretic terminology can be found in texts such as F. Harary, *Graph Theory*, Addison-Wesley Publishing Company, 1969; and J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, Macmillan Press Ltd., 1976.

The background on graph algorithms and applications can be supplemented by books such as:

A.V. Aho, J.E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

V. Chachra, P.M. Ghare and J.M. Moore, *Applications of Graph Theory Algorithms*, Elsevier North-Holland, Inc., 1979.

N. Christofides, *Graph Theory, An Algorithmic Approach*, Academic Press, London, 1975.

S. Even, *Graph Algorithms*, Computer Science Press, 1979.

A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, 1985.

M. Gondran and M. Minoux, *Graphs and Algorithms*, Wiley-Interscience, 1984.

K. Mehlhorn, *Graph Algorithms and NP-Completeness*, Springer-Verlag, Inc., 1984.

E. Minieka, *Optimization Algorithms for Networks and Graphs*, Marcel Dekker, Inc., 1978.

INTRODUCTION

For convenience, the definitions of most graph-theoretic terms in this book appear in Appendix I. Every chapter is self-contained and largely independent. Each topic is presented in the same format under five subheadings:

A. *Problem description*—a general description of the problem.
B. *Method*—an outline of the solution procedure.
C. *Subroutine parameters*—a description of all parameters of the subroutine that implements the method described in B.
D. *Test example*—a simple example illustrating the usage of the subroutine.
E. *Program listing*—the complete listing of the code.

In general, the solution procedures will be only briefly outlined. References given at the end of this book should be consulted for all details.

Throughout this book, it is assumed that a graph of n nodes and m edges has its nodes numbered from 1 to n. In the implementation of each solution procedure, one of two graph representations is used: the matrix form or the forward star form. The square matrix representation is mainly used to store the edge distance for every pair of nodes in a complete graph, resulting in an n^2 storage requirement. The forward star representation lists each edge by its starting node, ending node, and its length. Furthermore, the edges in the graph are ordered by the starting node so that all edges starting at the same node appear together, resulting in only an $n + 2m$ storage requirement. In this way, if one knows which is the first edge starting at each node i, then one can determine the last edge starting from node i as the edge immediately preceding the first edge starting at node $i + 1$.

A list of all subroutines in this book is summarized in Appendix II. The programs are written in FORTRAN 77. Communication to each subroutine is made solely through the parameter list. The test runs were all performed on the Amdahl 5870 using the IBM VS FORTRAN Compiler.

PREFACE

The many applications of graph theory constantly draw the attention of researchers, especially in the search for efficient algorithms. Although many well-developed procedures have appeared in books and journals, ready-to-use computer codes are generally not easily accessible. This book attempts to provide such a source. It is not meant to be a collection of the most efficient algorithms; the choice of the topics and their solution procedures is purely based on the author's interests. The main objective of this book is to provide computer programs that can be used with minimal effort for problem-solving without much concern for their underlying methodology and implementation.

H.T. Lau
Ile des Soeurs
Quebec, Canada

CONTENTS

1

CONNECTIVITY

Maximum Connectivity

A. Problem description

Let n and k be two given positive integers. The problem is to construct a k-connected graph $G(k, n)$ on n nodes with as few edges as possible. Observe that for $k = 1$, the graph $G(1, n)$ is a spanning tree. Consequently, it is assumed that $k \geq 2$. Moreover, it is known that $G(k, n)$ has exactly $\lceil (n*k)/2 \rceil$ edges, where $\lceil x \rceil$ is the smallest integer greater than or equal to x.

B. Method

Label the nodes of the graph by the integers $0, 1, 2, \ldots, n - 1$.

CASE 1. k is even. Let $k = 2t$.

The graph $G(2t, n)$ is constructed as follows. First, draw an n-gon, that is, add the edges

$$(0, 1), (1, 2), (2, 3), \ldots, (n - 2, n - 1), (n - 1, 0),$$

then join nodes i and j if and only if

$$|i - j| \equiv p \pmod{n}, \text{ where } 2 \leq p \leq t.$$

CASE 2. *k* is odd, *n* is even. Let $k = 2t + 1$.

The graph $G(2t + 1, n)$ is constructed by first drawing $G(2t, n)$, and then joining node i to node

$$i + (n/2), \text{ for } 0 \leq i < n/2.$$

CASE 3. *k* is odd, *n* is odd. Let $k = 2t + 1$.

The graph $G(2t + 1, n)$ is constructed by first drawing $G(2t, n)$, and then join

node 0 to node $(n - 1)/2$,
node 0 to node $(n + 1)/2$,
node i to node $i + (n + 1)/2$, for $1 \leq i < (n - 1)/2$.

C. Subroutine MAKEG parameters

Input:

N	Number of nodes.
K	The required graph is K-connected, K ≥ 2.
NK2	The smallest integer greater than or equal to (N*K)/2.

Output:

INODE, JNODE	INODE(i), JNODE(i) are the end nodes of the ith edge in the K-connected graph, $i = 1, 2, \ldots,$ NK2.

D. Test example

Construct a 5-connected graph on eight nodes with as few edges as possible.

E. Program listing

MAIN PROGRAM

```
      INTEGER INODE(20),JNODE(20)
      N = 8
      K = 5
      NK2 = 20
      CALL MAKEG (N,K,NK2,INODE,JNODE)
      WRITE(*,10) N, K, NK2
10    FORMAT(/' NUMBER OF NODES = ',I3,',',
     +             3X,I2,'-CONNECTED,'/
     +           ' NUMBER OF EDGES = ',I3//
     +             ' LIST OF EDGES:'/)
      WRITE(*,20) (INODE(I),I=1,NK2)
20    FORMAT(1X,25I3)
      WRITE(*,20) (JNODE(I),I=1,NK2)
      STOP
      END
```

OUTPUT RESULTS

```
 NUMBER OF NODES =   8,   5-CONNECTED,
 NUMBER OF EDGES  = 20
 LIST OF EDGES:
 1 2 3 4 5 6 7 8 1 1 2 2 3 4 5 6 1 2 3 4
 2 3 4 5 6 7 8 1 3 7 4 8 5 6 7 8 5 6 7 8
```

```
      SUBROUTINE MAKEG (N,K,NK2,INODE,
     +   JNODE)
C
C     Construct a K-connected graph of N nodes with
C        the least number of edges
C
      INTEGER INODE(NK2),JNODE(NK2)
      LOGICAL EVENK,EVENN,JOIN
C
C     Make an N-gon
C
      NK2 = 0
```

```
      N1 = N - 1
      DO 10 I = 1, N1
        NK2 = NK2 + 1
        INODE(NK2) = I
        JNODE(NK2) = I + 1
 10   CONTINUE
      NK2 = NK2 + 1
      INODE(NK2) = N
      JNODE(NK2) = 1
      IF (K .EQ. 2) RETURN
C
      EVENK = .TRUE.
      KHALF = K / 2
      IF (K .NE. 2*KHALF) EVENK = .FALSE.
C
      DO 40 I = 1, N1
        I1 = I + 1
        DO 30 J = I1, N
          JOIN = .FALSE.
          JI = J - I
          DO 20 L = 2, KHALF
            IF ((MOD(L,N) .EQ. JI) .OR.
     +          (JI + L .EQ. N)) JOIN = .TRUE.
 20       CONTINUE
          IF (JOIN) THEN
            NK2 = NK2 + 1
            INODE(NK2) = I
            JNODE(NK2) = J
          ENDIF
 30     CONTINUE
 40   CONTINUE
C
C     If K is even then finish
C
      IF (EVENK) RETURN
C
      EVENN = .TRUE.
      NHALF = N / 2
      IF (N .NE. 2*NHALF) EVENN = .FALSE.
```

```
C
      IF (EVENN) THEN
C
C        K is odd, N is even
C
         DO 50 I = 1, NHALF
            NK2 = NK2 + 1
            INODE(NK2) = I
            JNODE(NK2) = I + NHALF
  50     CONTINUE
      ELSE
C
C        K is odd, N is odd
C
         NPP = (N + 1) / 2
         NMM = (N - 1) / 2
         DO 60 I = 2, NMM
            NK2 = NK2 + 1
            INODE(NK2) = I
            JNODE(NK2) = I + NPP
  60     CONTINUE
         NK2 = NK2 + 1
         INODE(NK2) = 1
         JNODE(NK2) = NMM + 1
         NK2 = NK2 + 1
         INODE(NK2) = 1
         JNODE(NK2) = NPP + 1
      ENDIF
C
      RETURN
      END
```

1-2 Edge-Connectivity

A. Problem description

The problem is to find the edge-connectivity of a given connected undirected graph.

B. Method

As a preliminary, a *network* is defined to be a directed graph G in which each edge (i, j) is associated with a nonnegative number $c(i, j)$ called the *capacity* of the edge. Let the number $f(i, j)$ be the *flow* from node i to node j. A flow in the network is *feasible* if $f(i, j)$ does not exceed $c(i, j)$ for each edge (i, j) in G, and the sum of all flows incoming to node i is equal to the sum of all flows outgoing from node j.

Let s and t be some specified nodes, called the *source* and *sink*, respectively. The *maximum network flow problem* is to find a flow in the network from s to t such that the amount of the flow into t is maximum.

A *cut* is a subset S of the nodes of G with the capacity equal to:

$$\sum_{\substack{i \in S \\ j \notin S}} c(i, j)$$

The well-known max-flow min-cut theorem states that the maximum flow is equal to the minimal cut in a network. The subroutine NFLOW below finds a maximum flow and a minimal cut set in a given network with specified source and sink nodes.

With the background of maximum network flow, the method of finding the edge-connectivity of an undirected graph is quite straightforward.

Denote the nodes of the input connected, undirected graph G by $1, 2, \ldots, n$. For $j = 2$ to n do the following: Take node 1 as the source, node j as the sink in G, assign a unit capacity to all edges in both directions, and find the value of a maximum flow $g(j)$ in the resulting network. The edge-connectivity is equal to the minimum of all $g(j)$, for $j = 2, 3, \ldots, n$.

The subroutine EDGECN below finds the edge-connectivity of a given undirected graph with the help of subroutine NFLOW.

The maximum network flow algorithm requires $O(n^3)$

operations. The edge-connectivity of a graph will therefore be found in $O(n^4)$ operations.

C. Subroutine EDGECN parameters

Input:

N	Number of nodes.
M	Number of edges.
M4	Equal to 4*M.
INODE, JNODE	Each is an integer vector of length M, INODE(i), JNODE(i) are the end nodes of the ith edge in the connected undirected graph.

Output:

KCONCT	The edge-connectivity of the graph.

Working storages:

For the description of the following working arrays, see the parameters of subroutine NFLOW.

IEDGE	Integer vector of length M4.
JEDGE	Integer vector of length M4.
CAPAC	Integer vector of length M4.
MINCUT	Integer vector of length N.
FLOW	Integer vector of length M4.
NODFLO	Integer vector of length N.
POINT	Integer vector of length N.
IMAP	Integer vector of length N.
JMAP	Integer vector of length N.

Subroutine NFLOW parameters

Let G be a network of E edges.

Input:

N	Number of nodes.

M — Equal to 2*E.

INODE, JNODE — Each is an integer vector of length M; an edge in G directed from node u to node v will be represented by two directed edges (u, v) and (v, u), where

$$\text{INODE}(i) = \text{u}, \quad \text{JNODE}(i) = \text{v},$$
$$\text{INODE}(j) = \text{u}, \quad \text{JNODE}(j) = \text{v},$$

for some i and j. On output, the edges will be sorted lexicographically.

CAPAC — Integer vector of length M; CAPAC(i) is the edge capacity of edge (u, v) in G, and the artificially created edge (v, u) will have an edge capacity CAPAC(j) equal to zero.

ISORCE, ISINK — A maximum flow is required from node ISORCE to node ISINK in the network.

Output:

MINCUT — Integer vector of length N; MINCUT(i) = 1 if node i is in the minimal cut set; otherwise, it is equal to zero.

FLOW — Integer vector of length M; FLOW(i) is the amount of flow on edge i.

NODFLO — Integer vector of length N; NODFLO(i) is the amount of flow through node i.

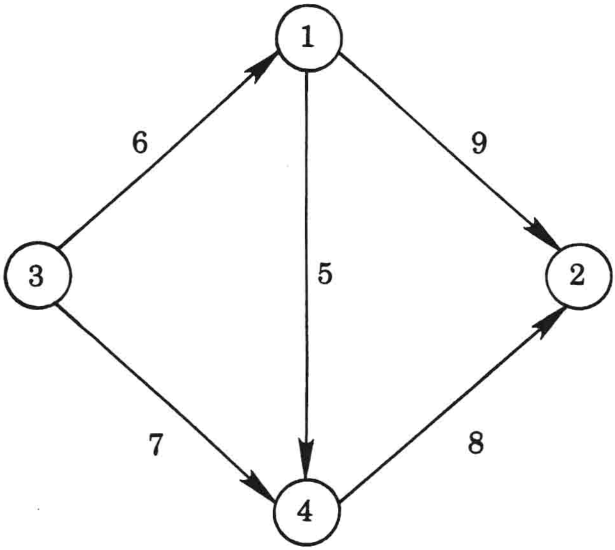
Working storages:

POINT — Integer vector of length N; POINT(i) is the first edge from node i.

IMAP — Integer vector of length N; pointer array.

JMAP — Integer vector of length N; pointer array.

REMARK. As an example for using NFLOW, we want to find the maximum flow from node 3 to node 2 in the following network of $E = 5$ edges.



The numbers on the edges represent the edge capacity.

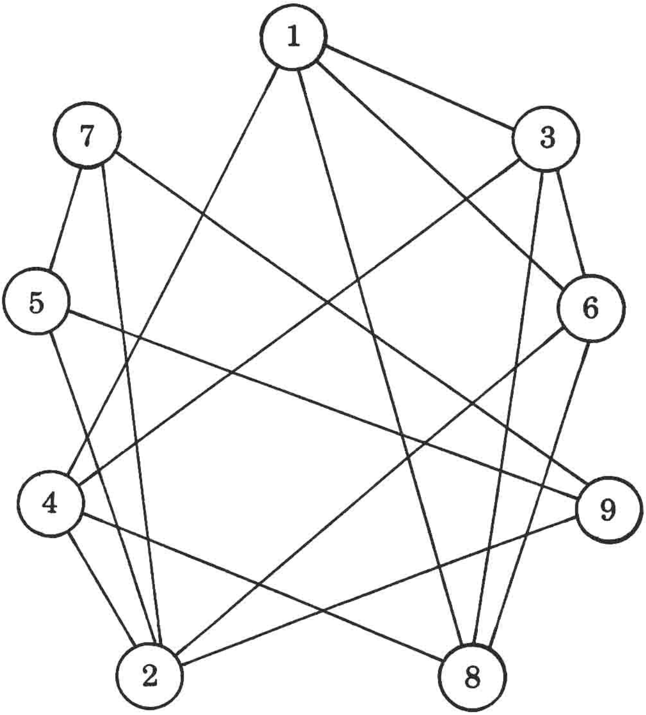
The input data to subroutine NFLOW might be:

N = 4
M = 10
INODE: 4 2 3 1 1 2 3 4 1 4
JNODE: 2 4 1 3 2 1 4 3 4 1
CAPAC: 8 0 6 0 9 0 7 0 5 0
ISORCE = 3
ISINK = 2

Notice that the edges can be arranged in an arbitrary order.

D. Test example

Find the edge-connectivity of the following graph with nine nodes and 17 edges.



E. Program listing

MAIN PROGRAM

```
      INTEGER INODE(17),JNODE(17),IEDGE(68),
     +              JEDGE(68),CAPAC(68),MINCUT(9),
     +              FLOW(68),NODFLO(9),POINT(9),
     +              IMAP(9),JMAP(9)
      DATA INODE / 6,2,3,6,7,1,4,7,3,4,9,6,5,4,2,9,4/,
     +          JNODE / 8,5,1,3,2,8,3,5,8,1,2,1,9,8,6,7,2/
C
      N = 9
      M = 17
      M4 = 4*M
      CALL EDGECN(N,M,M4,INODE,JNODE,
     +                   KCONCT,IEDGE,JEDGE,CAPAC,
     +                   MINCUT,FLOW,NODFLO,
     +                   POINT,IMAP,JMAP)
```