# NAG Fortran Library Routine Document

# H03ADF

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

## 1 Purpose

H03ADF finds the shortest path through a directed or undirected acyclic network using Dijkstra's algorithm.

## 2 Specification

```
 SUBROUTINE H03ADF(N, NS, NE, DIRECT, NNZ, D, IROW, ICOL, SPLEN, PATH,
1                  IWORK, WORK, IFAIL)
 INTEGER           N, NS, NE, NNZ, IROW(NNZ), ICOL(NNZ), PATH(N),
1                  IWORK(3*N+1), IFAIL
 real              D(NNZ), SPLEN, WORK(2*N)
 LOGICAL           DIRECT
```

## 3 Description

This routine attempts to find the shortest path through a **directed** or **undirected acyclic** network, which consists of a set of points called **vertices** and a set of curves called **arcs** that connect certain pairs of distinct vertices. An acyclic network is one in which there are no paths connecting a vertex to itself. An arc whose origin vertex is $i$ and whose destination vertex is $j$ can be written as $i \rightarrow j$. In an undirected network the arcs $i \rightarrow j$ and $j \rightarrow i$ are equivalent (i.e., $i \leftrightarrow j$), whereas in a directed network they are different. Note that the shortest path may not be unique and in some cases may not even exist (e.g., if the network is disconnected).

The network is assumed to consist of $n$ vertices which are labelled by the integers $1, 2, \ldots, n$. The lengths of the arcs between the vertices are defined by the $n$ by $n$ **distance matrix** D, in which the element $d_{ij}$ gives the length of the arc $i \rightarrow j$; $d_{ij} = 0$ if there is no arc connecting vertices $i$ and $j$ (as is the case for an acyclic network when $i = j$). Thus the matrix $D$ is usually **sparse**. For example, if $n = 4$ and the network is directed, then

$$\mathrm{D} = \begin{pmatrix} 0 & d_{12} & d_{13} & d_{14} \\ d_{21} & 0 & d_{23} & d_{24} \\ d_{31} & d_{32} & 0 & d_{34} \\ d_{41} & d_{42} & d_{43} & 0 \end{pmatrix}.$$

If the network is undirected, D is **symmetric** since $d_{ij} = d_{ji}$ (i.e., the length of the arc $i \rightarrow j \equiv$ the length of the arc $j \rightarrow i$).

The method used by H03ADF is described in detail in Section 8.

## 4 References

Dijkstra E W (1959) A note on two problems in connection with graphs *Numer. Math.* **1** 269–271

## 5 Parameters

1: N – INTEGER *Input*

*On entry*: $n$, the number of vertices.

*Constraint*: $\mathrm{N} \geq 2$.

2: NS – INTEGER *Input*
3: NE – INTEGER *Input*

*On entry*: $n_s$ and $n_e$, the labels of the first and last vertices, respectively, between which the shortest path is sought.

*Constraints*:

$1 \leq$ NS $\leq$ N,
$1 \leq$ NE $\leq$ N,
NS $\neq$ NE.

4: DIRECT – LOGICAL *Input*

*On entry*: indicates whether the network is directed or undirected as follows:

if DIRECT = .TRUE., the network is directed;

if DIRECT = .FALSE., the network is undirected.

5: NNZ – INTEGER *Input*

*On entry*: the number of non-zero elements in the distance matrix $D$.

*Constraints*:

if DIRECT = .TRUE., $1 \leq$ NNZ $\leq$ N $\times$ (N $-$ 1);
if DIRECT = .FALSE., $1 \leq$ NNZ $\leq$ N $\times$ (N $-$ 1)/2.

6: D(NNZ) – ***real*** array *Input*

*On entry*: the non-zero elements of the distance matrix $D$, ordered by increasing row index and increasing column index within each row. More precisely, D($k$) must contain the value of the non-zero element with indices (IROW($k$), ICOL($k$)); this is the length of the arc from the vertex with label IROW($k$) to the vertex with label ICOL($k$). Elements with the same row and column indices are not allowed. If DIRECT = .FALSE., then only those non-zero elements in the strict upper triangle of D need be supplied since $d_{ij} = d_{ji}$. (F11ZAF may be used to sort the elements of an arbitrarily ordered matrix into the required form. This is illustrated in Section 9.)

*Constraint*: D($k$) > 0.0, for $k = 1, 2, \ldots,$ NNZ.

7: IROW(NNZ) – INTEGER array *Input/Output*
8: ICOL(NNZ) – INTEGER array *Input*

*On entry*: IROW($k$) and ICOL($k$) must contain the row and column indices, respectively, for the non-zero element stored in D($k$).

*Constraints*:

IROW and ICOL must satisfy the following constraints (which may be imposed by a call to F11ZAF):
IROW($k - 1$) < IROW($k$), or
IROW($k - 1$) = IROW($k$) and ICOL($k - 1$) < ICOL($k$), for $k = 2, 3, \ldots,$ NNZ.
In addition, if DIRECT = .TRUE., $1 \leq$ IROW($k$) $\leq$ N, $1 \leq$ ICOL($k$) $\leq$ N and IROW($k$) $\neq$ ICOL($k$);
if DIRECT = .FALSE., $1 \leq$ IROW($k$) < ICOL($k$) $\leq$ N.

*On exit*: IROW is used as internal workspace prior to being restored and hence is unchanged.

9: SPLEN – ***real*** *Output*

*On exit*: the length of the shortest path between the specified vertices $n_s$ and $n_e$.

10: PATH(N) – INTEGER array *Output*

*On exit*: contains details of the shortest path between the specified vertices $n_s$ and $n_e$. More precisely, NS = PATH(1) $\rightarrow$ PATH(2) $\rightarrow \ldots \rightarrow$ PATH($p$) = NE for some $p \leq n$. The remaining $(n - p)$ elements are set to zero.

11: IWORK(3*N+1) – INTEGER array *Workspace*

12: WORK(2*N) – ***real*** array *Workspace*

13: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

*On exit*: IFAIL = 0 unless the routine detects an error (see Section 6).

For environments where it might be inappropriate to halt program execution when an error is detected, the value $-1$ or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, for users not familiar with this parameter the recommended value is 0. **When the value $-1$ or 1 is used it is essential to test the value of IFAIL on exit.**

## 6 Error Indicators and Warnings

If on entry IFAIL = 0 or $-1$, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL = 1

On entry, N < 2,
or NS < 1,
or NS > N,
or NE < 1,
or NE > N,
or NS = NE.

IFAIL = 2

On entry, NNZ > N × (N − 1) when DIRECT = .TRUE.,
or NNZ > N × (N − 1)/2 when DIRECT = .FALSE.,
or NNZ < 1.

IFAIL = 3

On entry, IROW($k$) < 1 or IROW($k$) > N or ICOL($k$) < 1 or ICOL($k$) > N or IROW($k$) = ICOL($k$) for some $k$ when DIRECT = .TRUE..

IFAIL = 4

On entry, IROW($k$) < 1 or IROW($k$) $\geq$ ICOL($k$) or ICOL($k$) > N for some $k$ when DIRECT = .FALSE..

IFAIL = 5

D($k$) $\leq$ 0.0 for some $k$.

IFAIL = 6

On entry, IROW($k - 1$) > IROW($k$) or IROW($k - 1$) = IROW($k$) and ICOL($k - 1$) > ICOL($k$) for some $k$.

IFAIL = 7

On entry, IROW$(k-1)$ = IROW$(k)$ and ICOL$(k-1)$ = ICOL$(k)$ for some $k$.

IFAIL = 8

No connected network exists between vertices NS and NE.

## 7 Accuracy

The results are exact, except for the obvious rounding errors in summing the distances in the length of the shortest path.

## 8 Further Comments

This routine is based upon Dijkstra's algorithm (see Dijkstra (1959)), which attempts to find a path $n_s \to n_e$ between two specified vertices $n_s$ and $n_e$ of shortest length $d(n_s, n_e)$.

The algorithm proceeds by assigning labels to each vertex, which may be **temporary** or **permanent**. A temporary label can be changed, whereas a permanent one cannot. For example, if vertex $p$ has a permanent label $(q, r)$, then $r$ is the distance $d(n_s, r)$ and $q$ is the previous vertex on a shortest length $n_s \to p$ path. If the label is temporary, then it has the same meaning but it refers only to the shortest $n_s \to p$ path found so far. A shorter one may be found later, in which case the label may become permanent.

The algorithm consists of the following steps.

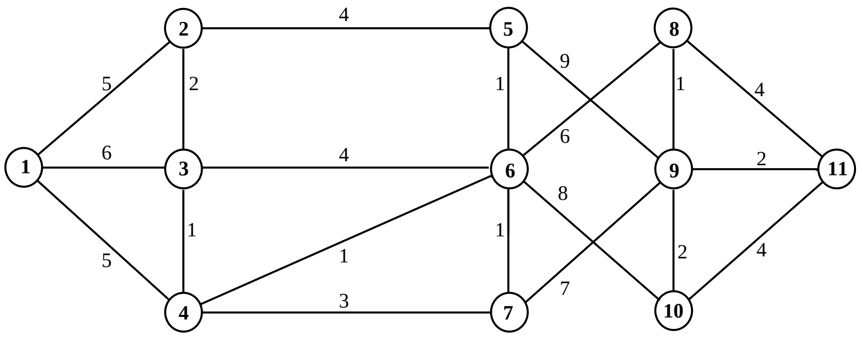
1. Assign the permanent label $(-, 0)$ to vertex $n_s$ and temporary labels $(-, \infty)$ to every other vertex. Set $k = n_s$ and go to (2).
2. Consider each vertex $y$ adjacent to vertex $k$ with a temporary label in turn. Let the label at $k$ be $(p, q)$ and at $y(r, s)$. If $q + d_{ky} < s$, then a new temporary label $(k, q + d_{ky})$ is assigned to vertex $y$; otherwise no change is made in the label of $y$. When all vertices $y$ with temporary labels adjacent to $k$ have been considered, go to (3).
3. From the set of temporary labels, select the one with the smallest second component and declare that label to be permanent. The vertex it is attached to becomes the new vertex $k$. If $k = n_e$ go to (4). Otherwise go to (2) unless no new vertex can be found (e.g., when the set of temporary labels is 'empty' but $k \neq n_e$, in which case no connected network exists between vertices $n_s$ and $n_e$).
4. To find the shortest path, let $(y, z)$ denote the label of vertex $n_e$. The column label $(z)$ gives $d(n_s, n_e)$ while the row label $(y)$ then links back to the previous vertex on a shortest length $n_s \to n_e$ path. Go to vertex $y$. Suppose that the (permanent) label of vertex $y$ is $(w, x)$, then the next previous vertex is $w$ on a shortest length $n_s \to y$ path. This process continues until vertex $n_s$ is reached. Hence the shortest path is

$$n_s \to \ldots \to w \to y \to n_e,$$

which has length $d(n_s, n_e)$.

## 9 Example

To find the shortest path between vertices 1 and 11 for the undirected network



### 9.1 Program Text

```
*     H03ADF Example Program Text
*     Mark 20 Revised. NAG Copyright 2001.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NMAX, NNZMAX
      PARAMETER        (NMAX=100,NNZMAX=1000)
      CHARACTER        DUP, ZERO
      PARAMETER        (DUP='F',ZERO='R')
*     .. Local Scalars ..
      real             SPLEN
      INTEGER          IFAIL, J, LENC, N, NE, NNZ, NS
      LOGICAL          DIRECT
*     .. Local Arrays ..
      real             D(NNZMAX), WORK(2*NMAX)
      INTEGER          ICOL(NNZMAX), IROW(NNZMAX), IWORK(3*NMAX+1),
     +                 PATH(NMAX)
*     .. External Subroutines ..
      EXTERNAL         F11ZAF, H03ADF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'H03ADF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N, NS, NE, NNZ, DIRECT
      IF (N.LE.NMAX .AND. NNZ.LE.NNZMAX) THEN
*
*        Read D, IROW and ICOL from data file.
*
         READ (NIN,*) (D(J),IROW(J),ICOL(J),J=1,NNZ)
*
*        Reorder the elements of D into the form required by H03ADF.
*
         IFAIL = 0
         CALL F11ZAF(N,NNZ,D,IROW,ICOL,DUP,ZERO,IWORK,IWORK(N+2),IFAIL)
*
*        Find the shortest path between vertices NS and NE.
*
         IFAIL = 0
         CALL H03ADF(N,NS,NE,DIRECT,NNZ,D,IROW,ICOL,SPLEN,PATH,IWORK,
     +               WORK,IFAIL)
*
         IF (IFAIL.EQ.0) THEN
*
*           Print details of shortest path.
*
            DO 20 J = 0, N - 1
               IF (PATH(J+1).EQ.0) THEN
```

```
                  LENC = J
                  GO TO 40
               END IF
   20       CONTINUE
            LENC = N
   40       CONTINUE
            WRITE (NOUT,99999) 'Shortest path = ', (PATH(J),J=1,LENC)
            WRITE (NOUT,99998) 'Length of shortest path = ', SPLEN
         END IF
      END IF
      STOP
*
99999 FORMAT (/1X,A,10(I2,:'  to '))
99998 FORMAT (/1X,A,G16.6)
      END
```

## 9.2 Program Data

```
H03ADF Example Program Data
11  1  11  20  F   :Values of N, NS, NE, NNZ and DIRECT
6.0   6   8
1.0   8   9
2.0   9  11
4.0   2   5
1.0   3   4
6.0   1   3
4.0   3   6
1.0   4   6
2.0   2   3
3.0   4   7
5.0   1   2
7.0   6  10
1.0   5   6
4.0   8  11
9.0   5   9
1.0   6   7
8.0   7   9
4.0  10  11
2.0   9  10
5.0   1   4        :End of D, IROW, ICOL
```

## 9.3 Program Results

```
 H03ADF Example Program Results

 Shortest path =  1  to  4  to  6  to  8  to  9  to 11

 Length of shortest path =      15.0000
```