NAG Toolbox

nag_mip_shortestpath (h03ad)

1 Purpose

nag_mip_shortestpath (h03ad) finds the shortest path through a directed or undirected acyclic network using Dijkstra's algorithm.

2 Syntax

```
[splen, path, ifail] = nag_mip_shortestpath(n, ns, ne, direct, d, irow, icol, 'nnz', nnz)
[splen, path, ifail] = h03ad(n, ns, ne, direct, d, irow, icol, 'nnz', nnz)
```

3 Description

nag_mip_shortestpath (h03ad) 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

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

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 nag_mip_shortestpath (h03ad) is described in detail in Section 9.

4 References

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

5 Parameters

5.1 Compulsory Input Parameters

1: **n** – INTEGER

n, the number of vertices.

Constraint: $\mathbf{n} \geq 2$.

2: **ns** – INTEGER
3: **ne** – INTEGER

n_s and n_e, the labels of the first and last vertices, respectively, between which the shortest path is sought.

Constraints:

$1 \leq \mathbf{ns} \leq \mathbf{n}$;
$1 \leq \mathbf{ne} \leq \mathbf{n}$;
$\mathbf{ns} \neq \mathbf{ne}$.

4: **direct** – LOGICAL

Indicates whether the network is directed or undirected.

direct $= true$
The network is directed.

direct $= false$
The network is undirected.

5: **d**(**nnz**) – REAL (KIND=nag_wp) array

The nonzero elements of the distance matrix D, ordered by increasing row index and increasing column index within each row. More precisely, $\mathbf{d}(k)$ must contain the value of the nonzero element with indices $(\mathbf{irow}(k), \mathbf{icol}(k))$; this is the length of the arc from the vertex with label $\mathbf{irow}(k)$ to the vertex with label $\mathbf{icol}(k)$. Elements with the same row and column indices are not allowed. If **direct** $= false$, then only those nonzero elements in the strict upper triangle of **d** need be supplied since $d_{ij} = d_{ji}$. (nag_sparse_real_gen_sort (f11za) may be used to sort the elements of an arbitrarily ordered matrix into the required form. This is illustrated in Section 10.)

Constraint: $\mathbf{d}(k) > 0.0$, for $k = 1, 2, \ldots, \mathbf{nnz}$.

6: **irow**(**nnz**) – INTEGER array
7: **icol**(**nnz**) – INTEGER array

$\mathbf{irow}(k)$ and $\mathbf{icol}(k)$ must contain the row and column indices, respectively, for the nonzero element stored in $\mathbf{d}(k)$.

Constraints:

irow and **icol** must satisfy the following constraints (which may be imposed by a call to nag_sparse_real_gen_sort (f11za)):

$\mathbf{irow}(k-1) < \mathbf{irow}(k)$;
$\mathbf{irow}(k-1) = \mathbf{irow}(k)$ and $\mathbf{icol}(k-1) < \mathbf{icol}(k)$, for $k = 2, 3, \ldots, \mathbf{nnz}$.

In addition, if **direct** $= true$, $1 \leq \mathbf{irow}(k) \leq \mathbf{n}$, $1 \leq \mathbf{icol}(k) \leq \mathbf{n}$ and $\mathbf{irow}(k) \neq \mathbf{icol}(k)$;

if **direct** $= false$, $1 \leq \mathbf{irow}(k) < \mathbf{icol}(k) \leq \mathbf{n}$.

5.2 Optional Input Parameters

1: **nz** – INTEGER

Default: the dimension of the arrays **irow**, **icol**, **d**. (An error is raised if these dimensions are not equal.)

The number of nonzero elements in the distance matrix D.

Constraints:

if **direct** $= true$, $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n} - 1)$;
if **direct** $= false$, $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n} - 1)/2$.

5.3 Output Parameters

1: **splen** – REAL (KIND=nag_wp)

The length of the shortest path between the specified vertices n_s and n_e.

2: **path**(**n**) – INTEGER array

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.

3: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

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** $\times$ (**n** $- 1$) when **direct** $= \mathit{true}$,
or **nnz** $>$ **n** $\times$ (**n** $- 1)/2$ when **direct** $= \mathit{false}$,
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** $= \mathit{true}$.

ifail $= 4$

On entry, **irow**$(k) < 1$ or **irow**$(k) \geq$ **icol**(k) or **icol**$(k) >$ **n** for some k when **direct** $= \mathit{false}$.

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**.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

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

nag_mip_shortestpath (h03ad) is based upon Dijkstra's algorithm (see Dijkstra (1959)), which attempts to find a path $n_s \rightarrow n_e$ 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 $n_s \rightarrow p$ path. If the label is temporary, then it has the same meaning but it refers only to the shortest $n_s \rightarrow p$ 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 $\left(k, q + d_{ky}\right)$ 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 $n_s \rightarrow n_e$ 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 $n_s \rightarrow y$ path. This process continues until vertex n_s is reached. Hence the shortest path is

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

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

9 Example

This example finds the shortest path between vertices 1 and 11 for the undirected network

9.1 Program Text

```
function h03ad_example

fprintf('h03ad example results\n\n');

n  = nag_int(11);
ns = nag_int(1);
ne = n;
direct = false;
```

```
d = [  5;   6;   5;
            2;        4;
                 1;        4;
                           1;   3;
                           1;             9;
                                1;   6;        7;
                                               8;
                                               1;        4;
                                                    2;   2;
                                                         4];
irow = [nag_int(1);1;1;  2;2;  3;3;  4;4;  5;5;  6;6; 6;  7; 8; 8;   9; 9; 10];
icol = [nag_int(2);3;4;  3;5;  4;6;  6;7;  6;9;  7;8;10;  9; 9;11;  10;11; 11];

[splen, path, ifail] = h03ad(n, ns, ne, direct, d, irow, icol);

fprintf('Shortest path = ');
j = 1;
while path(j)~=0;
  if j>1
    fprintf('  to  ');
  end
  fprintf('%2d', path(j));
  j = j+1;
end
fprintf('\n\nLength of shortest path = %16.6g\n', splen);
```

9.2 Program Results

```
     h03ad example results

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

Length of shortest path =              15
```