Navigating INFO

presented by:
Tim Haithcoat
University of Missouri Columbia

with materials from:
Environmental Systems Research Institute

Section Eight

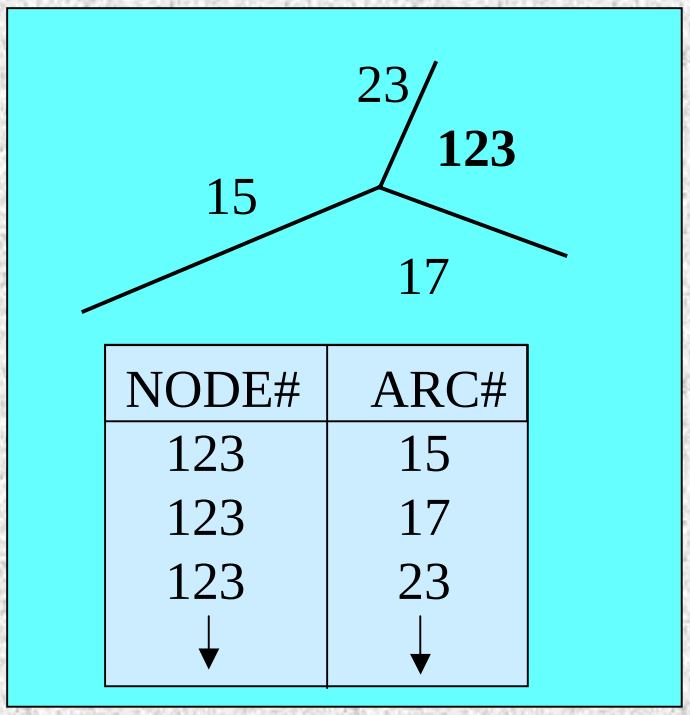
Advanced Processing

INTRODUCTION

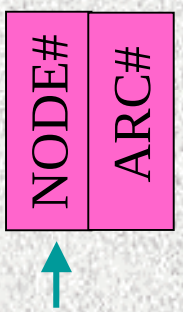
- This section deals with data structures and algorithms not inherent to ARC
- Mostly concerned with line coverages -- AAT's
- Topics include:
  - NODE-ARC LISTS (NAL)
  - NODE Valence tables (VAL)
  - Dangling and orphan ARCs
  - Travel time to a NODE (ALLOCATE)
  - Polygon neighbor list
  - BACKWALKING a least-cost path

NODE ARC LISTS

- NODE ARC LIST - “NAL”
- VERY IMPORTANT Data Structure -- used for traversing around and chaining through nodes
- A NAL is an ordered list of all ARCs surrounding a node.



<cover>.NAL



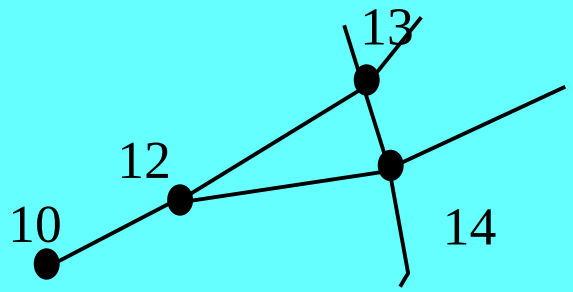
REDEFINED as
FNODE# with ALTERNATE
Item name of TNODE#

```
SEL COVER.AAT
REL COVER.NAL 1 BY FNODE# APPEND
CALC $1ARC# = COVER#
REL COVER.ANL 1 BY TNODE# APPEND
CALC $1ARC# = COVER3
SEL COVER.NAL
SORT NODE#, ARC#
```

NODE VALENCE TABLE

- VALENCE is the number of ARCs surrounding a NODE.

NODE#	VALENCE
10	1
12	3
13	4
14	4

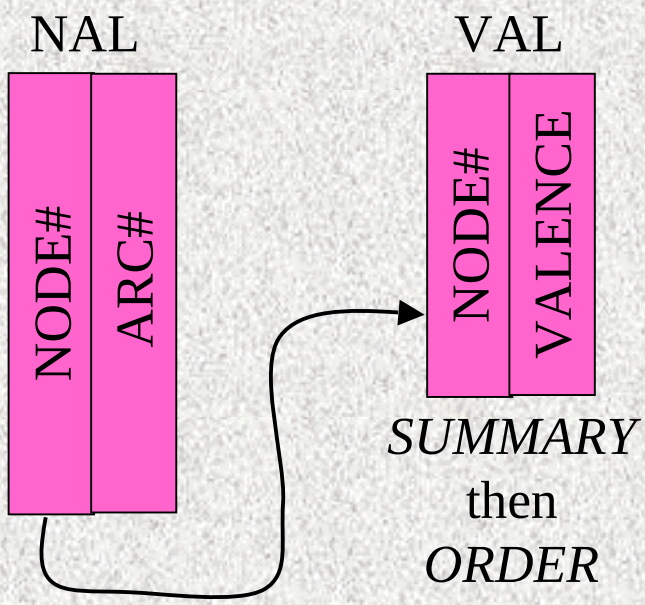


NODE#
VALENCE

NODE# is REDEFINED as FNODE# with an ALTERNATE item name of TNODE#

NODE VALENCE TABLE

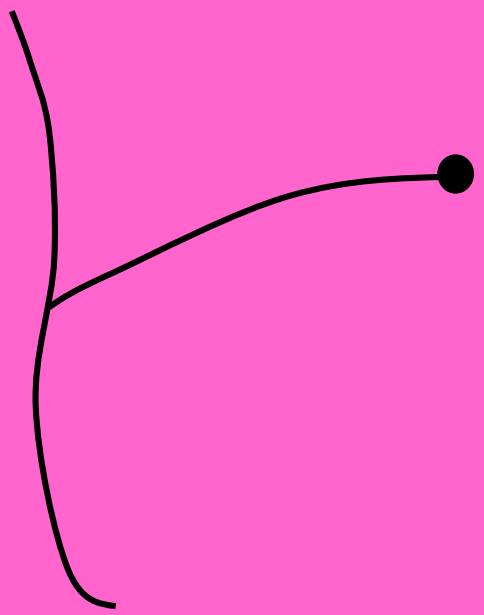
- Easiest methods to create a VAL is from an existing NAL.




```
SEL <cover>.NAL
REL<cover>.VAL 1 BY NODE# SUMMARY
CALC $NUM1 = $NUM`
SEL <cover>.NAL
REL <cover>.VAL 1 BY NODE# ORDER
CALC $1VALENCE +1
```

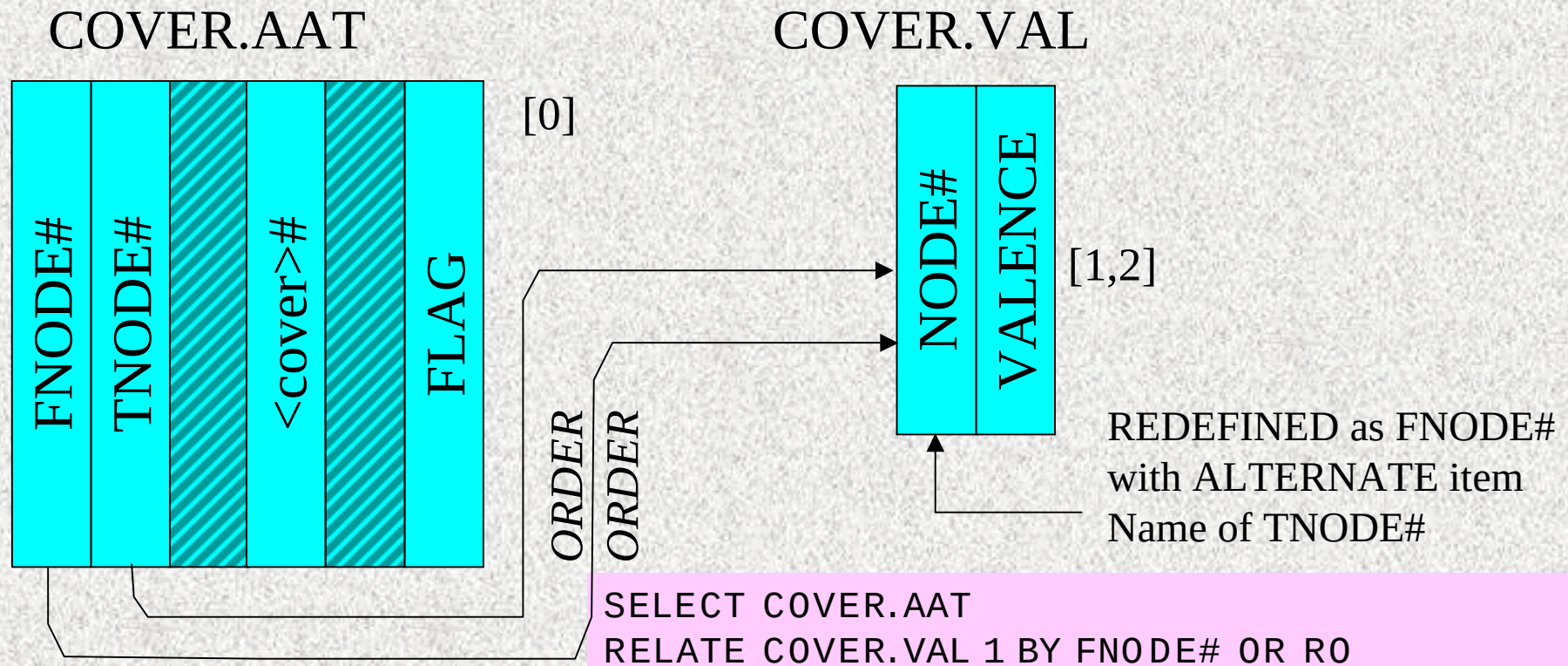
NODE VALENCE TABLE

DANGLING ARC



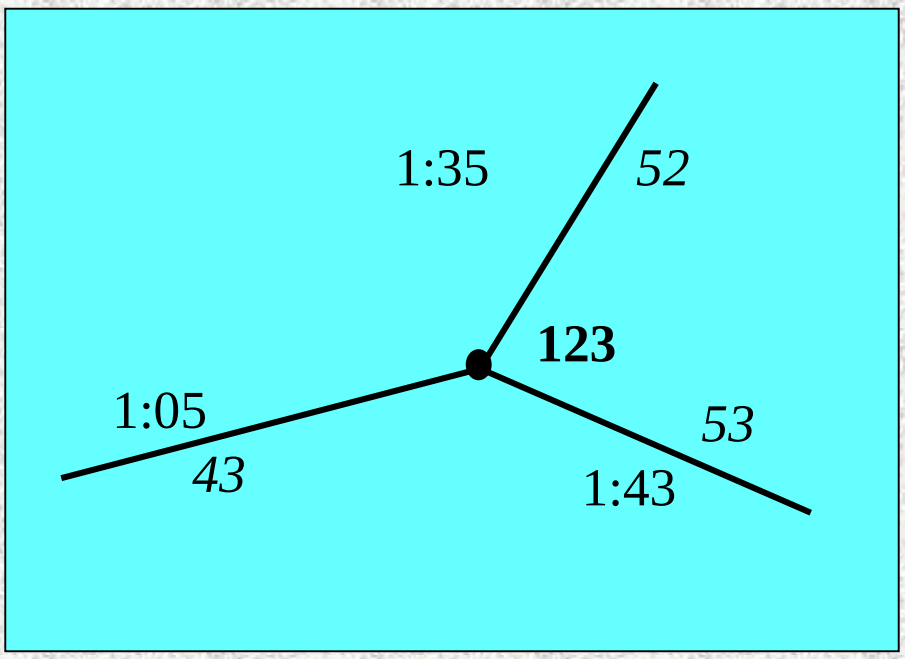
ORPHAN ARC

Dangling & Orphan ARC (Continued)




```
SELECT COVER.AAT
RELATE COVER.VAL 1 BY FNODE# OR RO
RELATE COVER.VAL 2 BYTNODE# OR RO
PROGRAM SECTION EVEN
IF ( $1VALENCE EQ 1 ) OR ( $2VALENCE EQ 1 )
    CALC FLAG = (dangle arc)
ENDIF
IF ( $1VALENCE EQ 1 ) AND ( $2VALENCE EQ 1)
    CALC FLAG = (orphan arc)
END IF
```

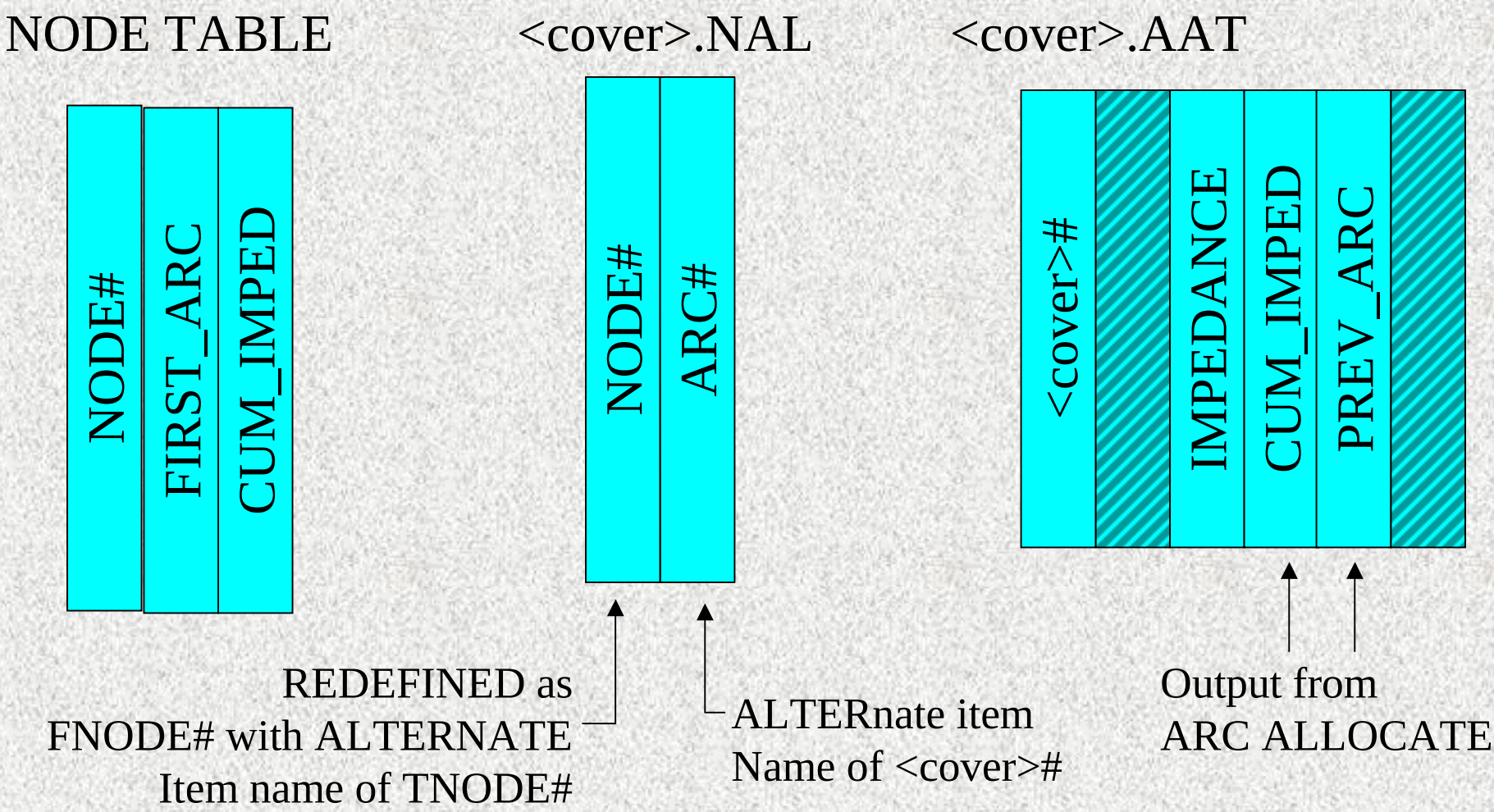
TRAVEL TIME TO A NODE



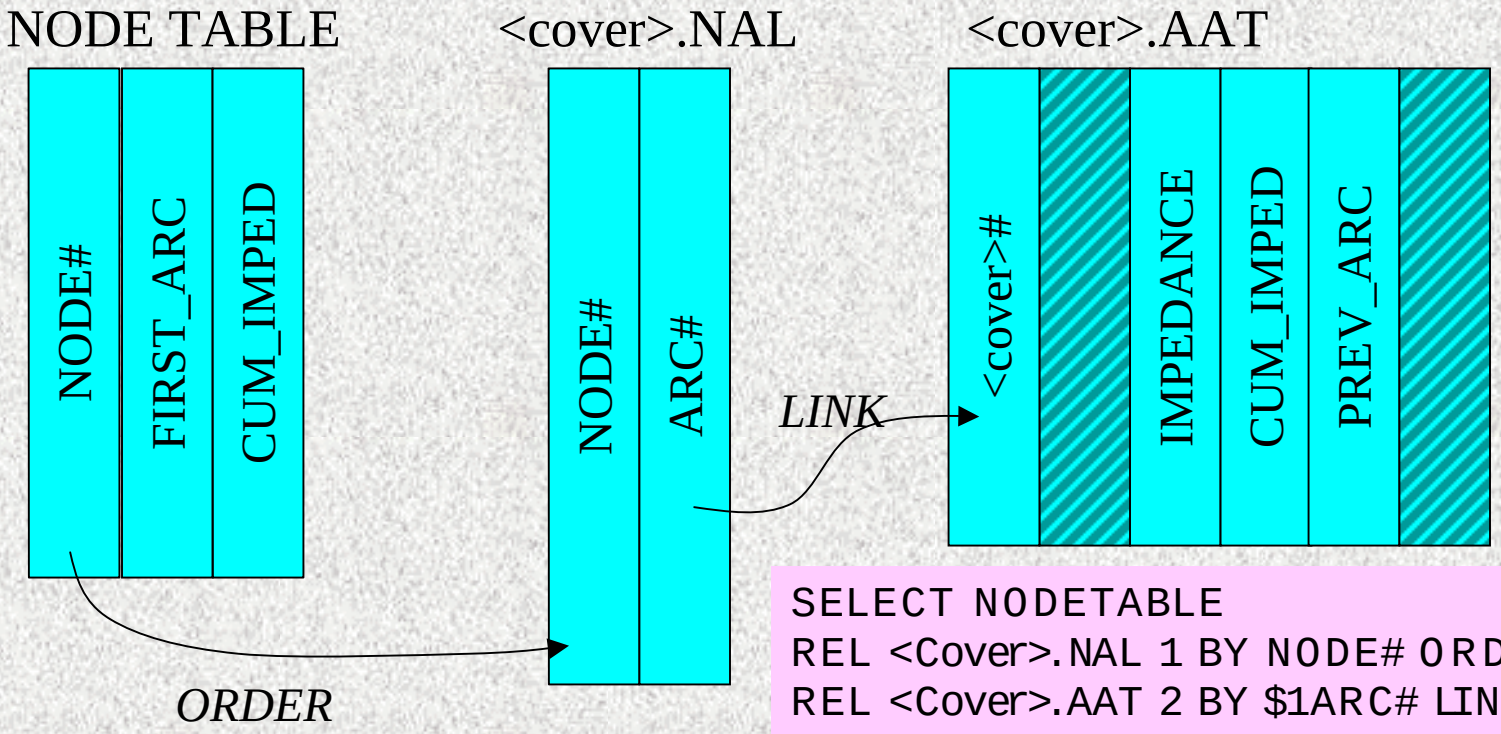
ARC ALLOCATE calculates cumulative impedance to the end of an arc, and does not provide cumulative impedance to a node (which is what we're often interested in).

In this example, the file NODETABLE contains a list of NODE#s that we want to find the cumulative impedance for, as well as the ARC# with the least cumulative impedance used to reach the node. In this example, arc$ 43 is the least-cost arc with a cumulative impedance of 1 hour and 5 minutes.

Travel Time to a Node (continued)

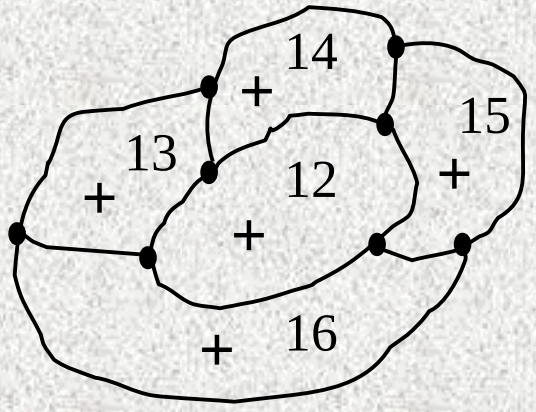


Travel Time to a Node (continued)




```
SELECT NODETABLE
REL <Cover>.NAL 1 BY NODE# ORDER RO
REL <Cover>.AAT 2 BY $1ARC# LINK RO
CALC $NUM1 = 9 ** 9
PROGRAM SECTION EVEN
IF $2CUM_IMPED LT $NUM1
   CALC FIRST_ARC = $1ARC#
   CALC CUM_IMPED = $2CUM_IMPED
   CALC $NUM1 = $2CUM_IMPED
ENDIF
NEXT 1
CALC $NUM1 = 9 ** 9
PROGRAM SECTION ODD
```

POLYGON NEIGHBOR LISTS



<Cover>.PAT

<cover>#	ATTRIBUTE
12	N
13	Y
14	Y
15	Y
16	Y

Polygon neighbor lists are very useful for finding polygons that are completely surrounded by some attribute, or for determining "edge effects".

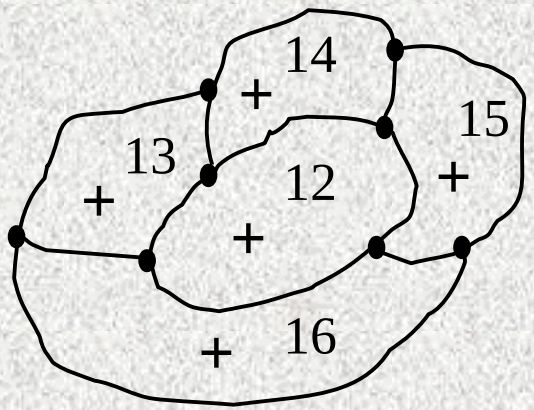
<Cover>.PNL

POLY#	NEIGHBOR#
12	13
12	14
12	15
12	16
13	1
13	12
13	14
13	16

ALTERNATE item name of <cover># REDEFINED as LPOLY# with ALTERNATE item name of RPOLY#

Polygon Neighbor Lists (continued)

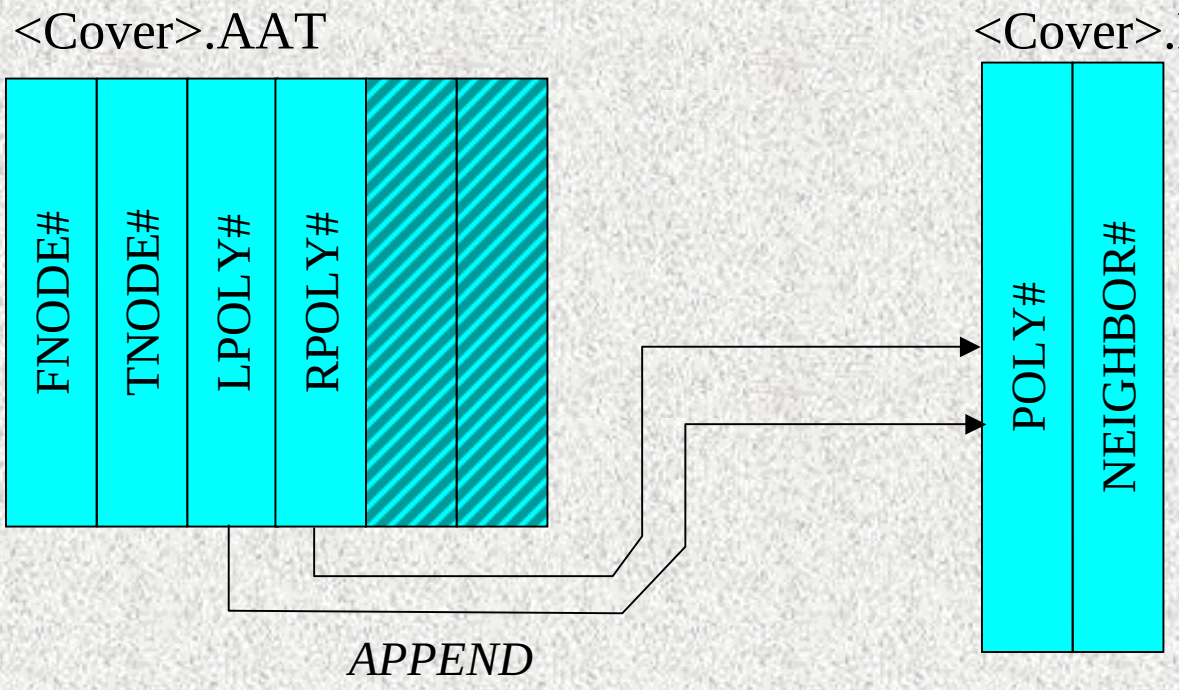
A Polygon Neighbor List is constructed from an *AAT*, which is built from a *PAT*.



<Cover>.AAT

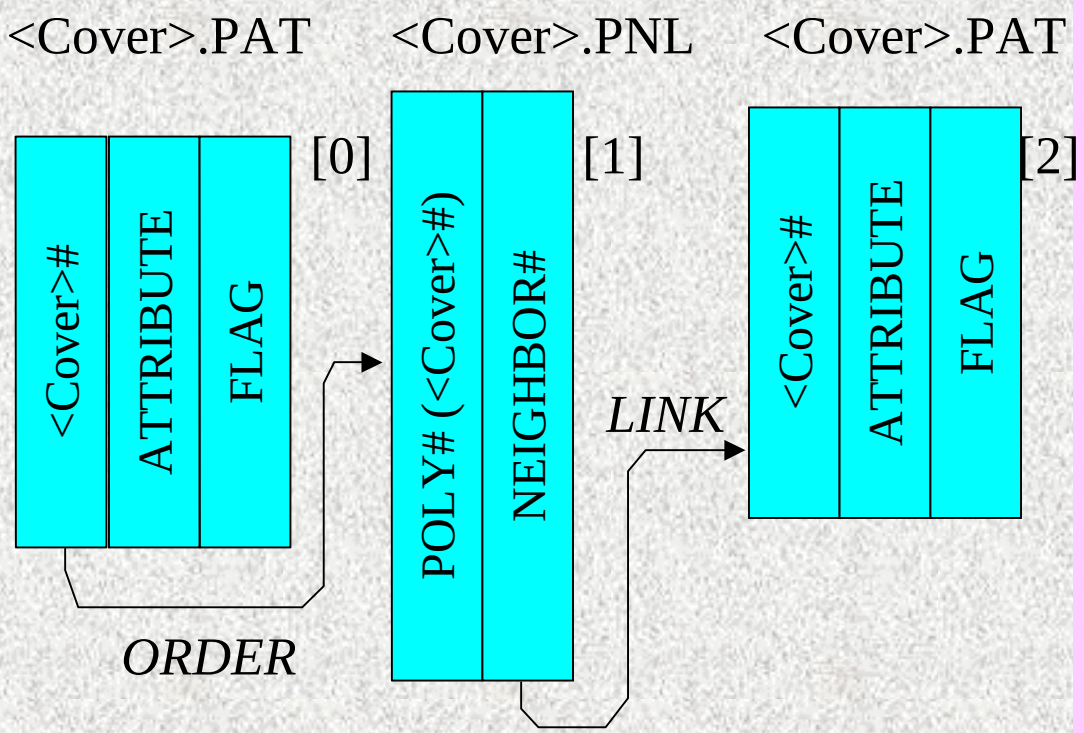
LPOLY#	RPOLY#
12	13
13	14
15	14
12	15
14	12
16	13
15	16
13	1
1	14
15	1
16	1

Polygon Neighbor Lists (continued)




```
SEL <Cover>.AAT
REL <Cover>.PNL 1 BY LPOLY# APPEND
CALC $1NEIGHBOR# = RPOLY#
REL <Cover>.PNL 1 BY RPOLY# APPEND
CALC $1NEIGHBOR# = LPOLY#
SEL <Cover>.PNL
SORT POLY#, NEIGHBOR
```

Polygon Neighbor Lists (continued)

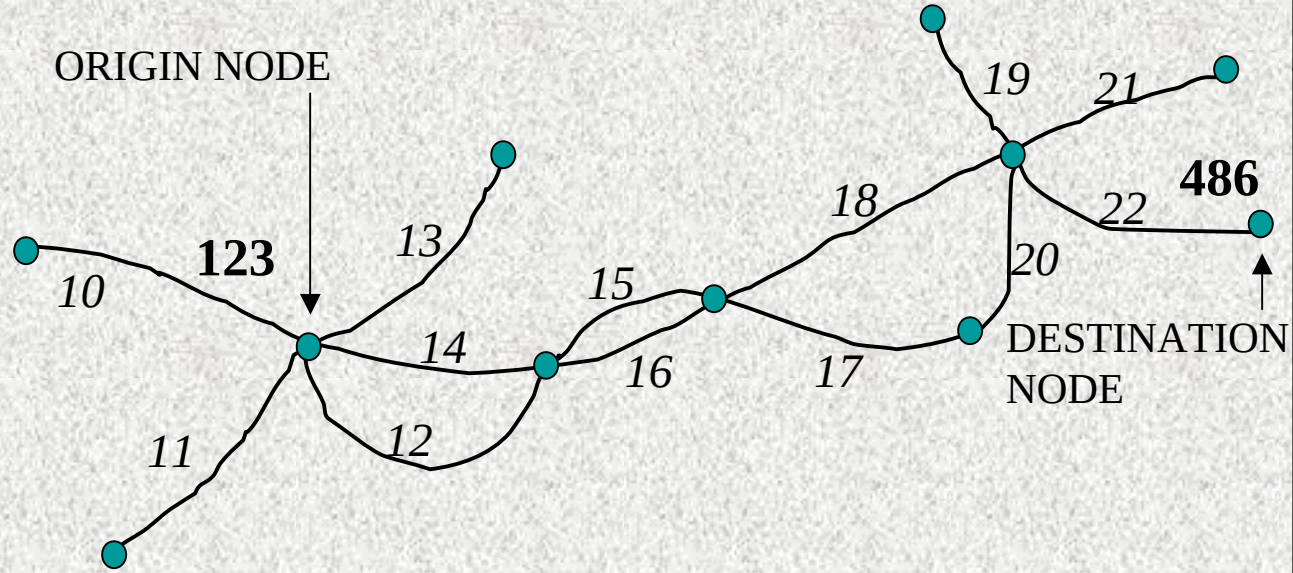



```
SEL <Cover>.PAT
REM
REM Initialize FLAG to '1', meaning
    that polygon
REM is completely surrounded by
    attribute 'Y'.
REM
CALC FLAG = 1
REL <Cover>.PNL 1 BY <Cover>#
   ORDERED RO
REL <Cover>.PAT 2 BY
   $1NEIGHBOR# LINK RO
PROGRAM SECTION EVEN
REM
REM If any of the polygons
    surrounding current
REM polygon have attributes other
    than Y', then
REM set FLAG to '0'
REM
IF $2ATTRIBUTE NC 'Y'
   CALC FLAG =0
ENDIF
NEXT1
```

BACKWALKING

- This is the output from ARC ALLOCATE. PREV_ARC points to the previous arc along the least-cost path.
- A data structure such as this is called a LINKED LIST.
- It is a very powerful data structure.
- This section is meant to demonstrate how one can "BACKWALK" along this linked list.
- BACKWALKING is useful for:
  - DETERMINING THE LEAST-COST ROUTE and
  - ACCUMULATING SOME VARIABLE ALONG THIS ROUTE.

BACKWALKING (continued)



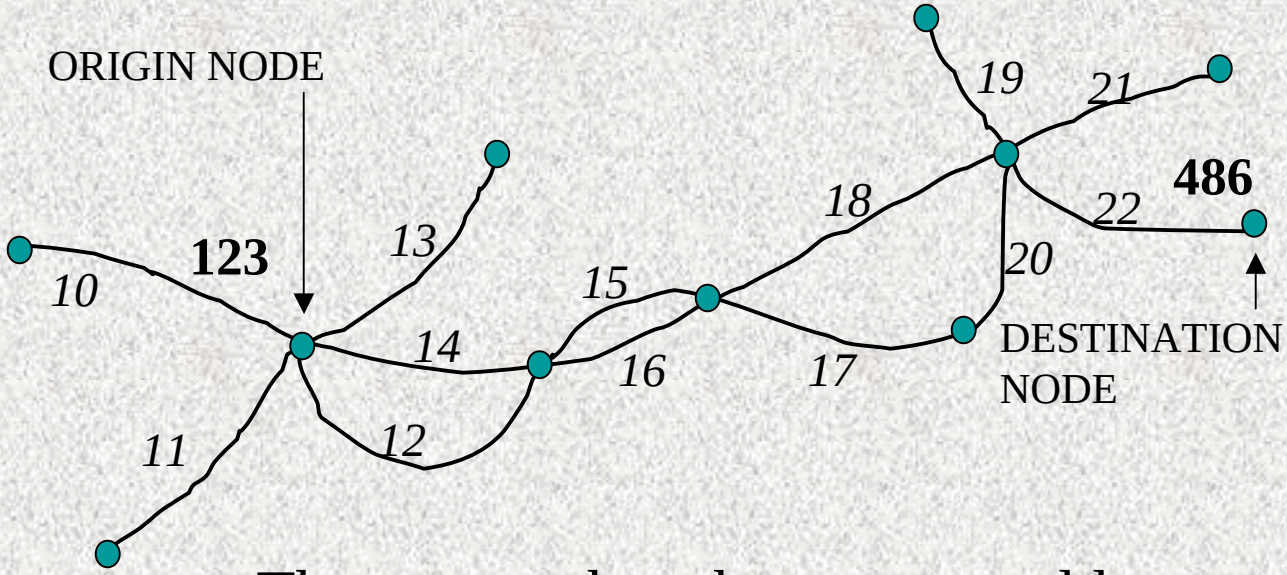
ARC#	PREV_ARC
10	-123
11	-123
12	-123
13	-123
14	-123
15	14
16	14
17	16
18	16
19	18
20	17
21	18
22	18

PREV-ARC is negative if it connects up with the ORIGIN NODE. The contents of such an arc is the negative of the ORIGIN NODE number, in this case, 123.

BACKWALKING (continued)

- PROBLEM: In an item PATH on an AAT, set PATH = 1 if arc is on least-cost path from destination node to origin node.
  - This is exactly what ARC ROUTE would give you. We typically run ALLOCATE and then use this procedure to find paths, especially if there are lots of destination nodes.

BACKWALKING (continued)

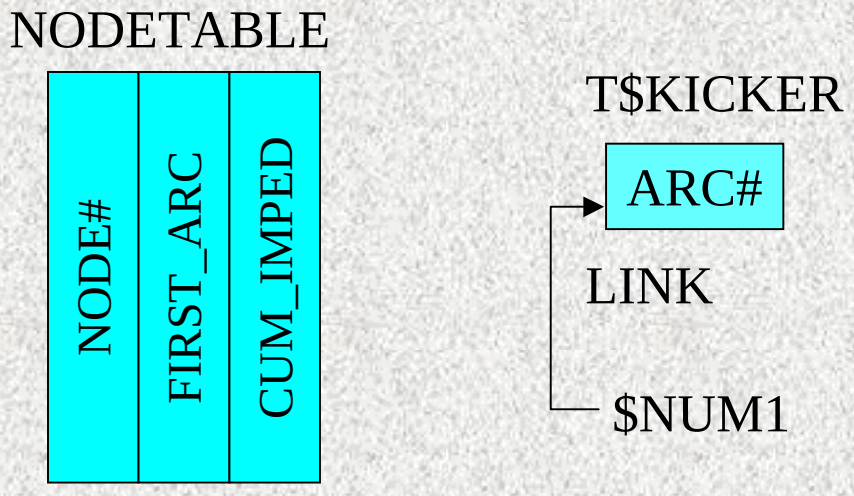


The arrows show how one would BACKWALK along a LINKED LIST

ARC#	PREV_ARC
10	-123
11	-123
12	-123
13	-123
14	-123
15	14
16	14
17	16
18	16
19	18
20	17
21	18
22	18

To initiate the BACKWALK, we must first find the least-cost arc that connects with the destination node.

BACKWALKING (Continued)



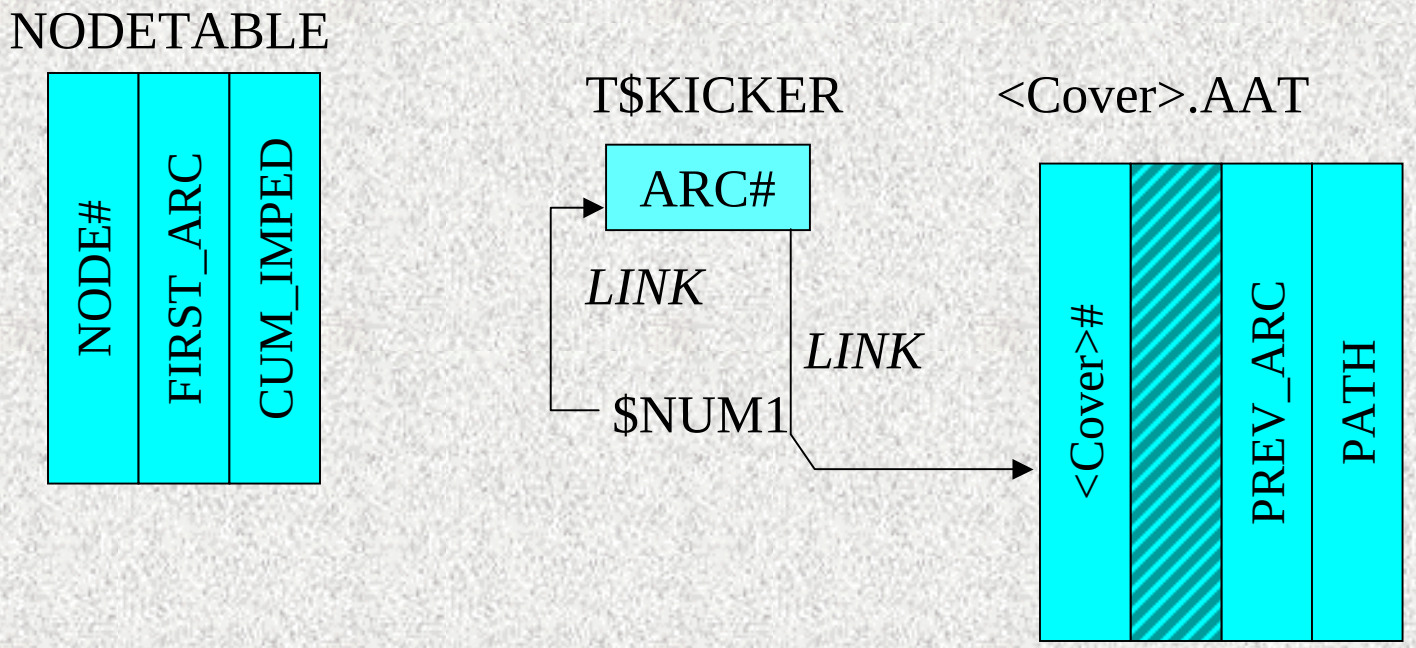

```
SEL NODETABLE
REL T$KICKER 1 BY $NUM1 LINK
RESEL BY NODE# EQ 486
CACL $1ARC# = FIRST_ARC
```

BACKWALKING (continued)

ARC#	PREV_ARC
10	-123
11	-123
12	-123
13	-123
14	-123
15	14
16	14
17	16
18	16
19	18
20	17
21	18
22	18

```
FORMAT $NUM1,4,5,B
CALC $NUM1 = 1
SEL NODTABLE
REL T$KICKER 1 BY $NUM1 LINK
REL <Cover>.AAT 2 BY $!ARC# LINK
RESEL BY NODE # = 486
CALC $1ARC# = FIRST_ARC
PROGRAM SECTION EVEN
DO UNTIL $2PREV_ARC LIT 0
   CALC $2PATH = 1
   CALC $1ARC# = $2PREV_ARC
DOEND
```

BACKWALKING (continued)



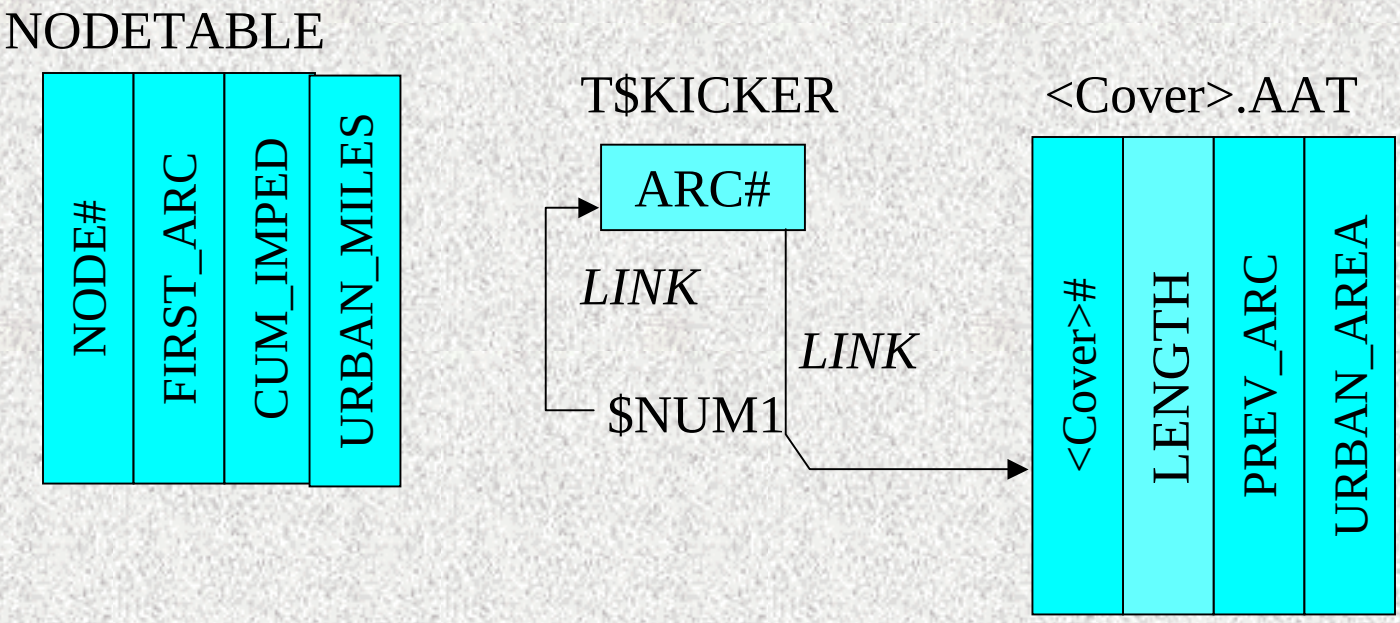
BACKWALKING (continued)

ARC#	PREV_ARC
10	-123
11	-123
12	-123
13	-123
14	-123
15	14
16	14
17	16
18	16
19	18
20	17
21	18
22	18

```
PROGRAM SECTION EVEN
DO UNTIL $2PREV_ARC LT 0
   IF $2URBAN_AREA CN 'Y'
        CALC URBAN _MILES =
        URBAN_MILES + $2LENGTH
   ENDIF
   CALC $1ARC# = $#2PREV_ARC
DOEND
```

- There are all sorts of things we can do with BACKWALKING.
- For instance, we can accumulate some variable, such as
  - miles through urban areas along the least-cost path,
  - or types of road class traveled on,
  - or number and types of intersections encountered.

BACKWALKING (continued)



Although these algorithms get into the realm of geographic modeling, they can prove to be very useful for many applications.