Introduction to Software Testing

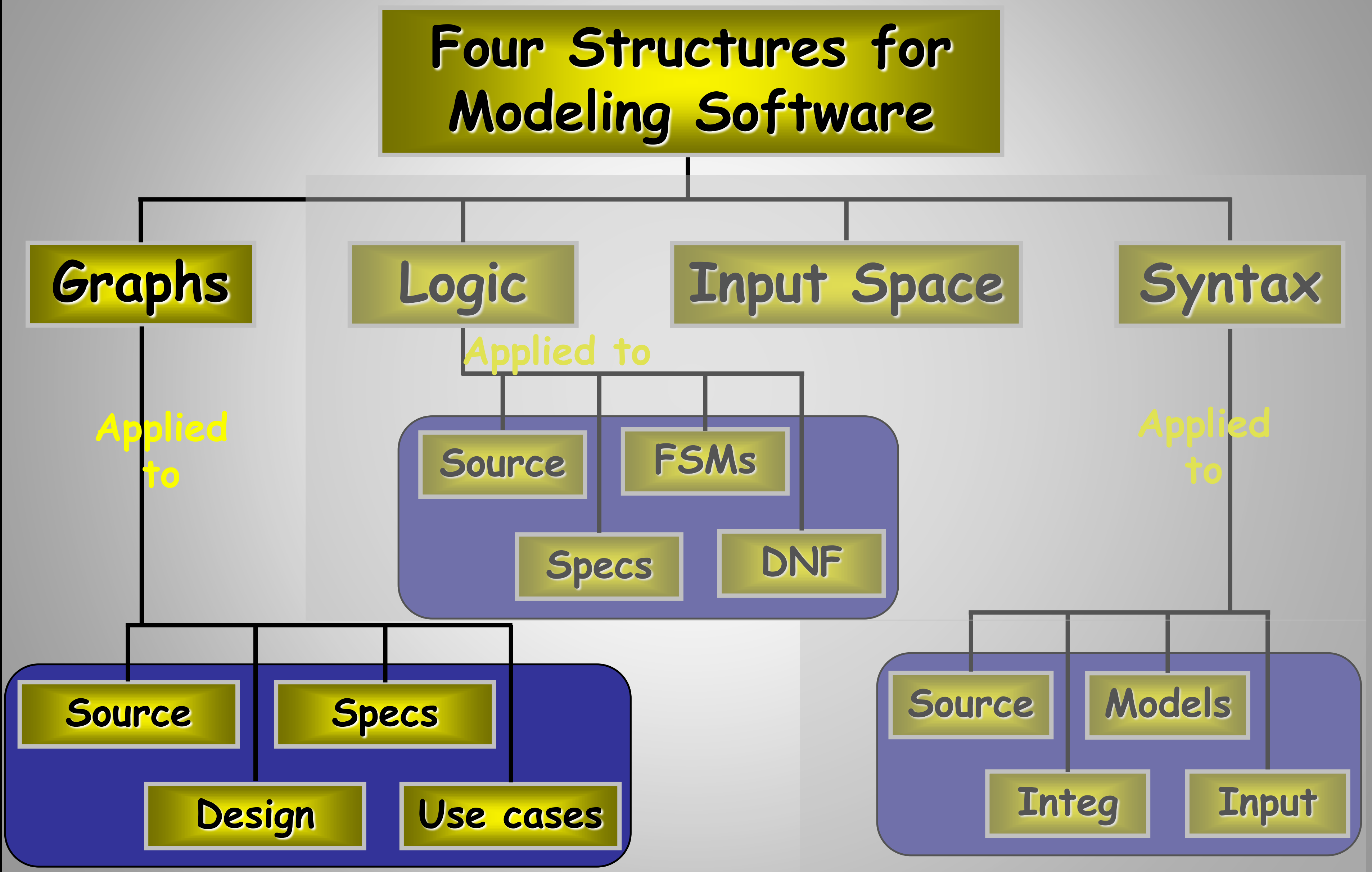
Chapter 2

Overview Graph Coverage Criteria

Paul Ammann & Jeff Offutt

Updated by Sunae Shin

Ch. 02 : Graph Coverage



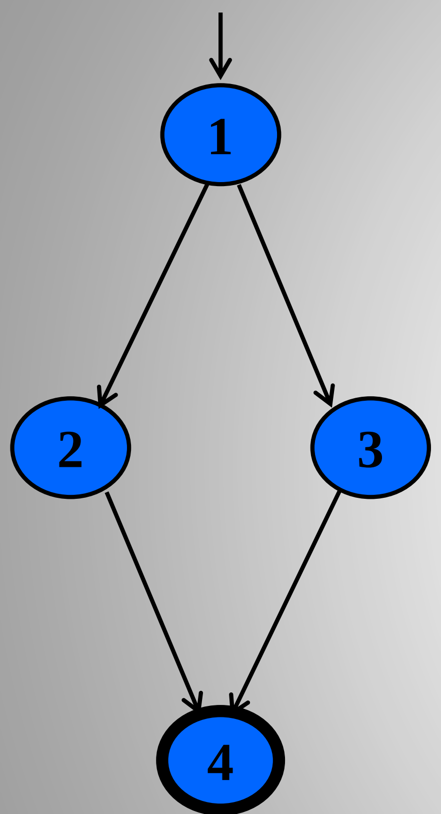
Covering Graphs

- Graphs are the most commonly used structure for testing
- Graphs can come from many sources
  - Control flow graphs
  - Design structure
  - FSMs and statecharts
  - Use cases
- Tests usually are intended to “cover” the graph in some way

Definition of a Graph

- A set N of nodes, N is not empty
- A set N_0 of initial nodes, N_0 is not empty
- A set N_f of final nodes, N_f is not empty
- A set E of edges, each edge from one node to another
  - (n_i , n_j), i is predecessor, j is successor

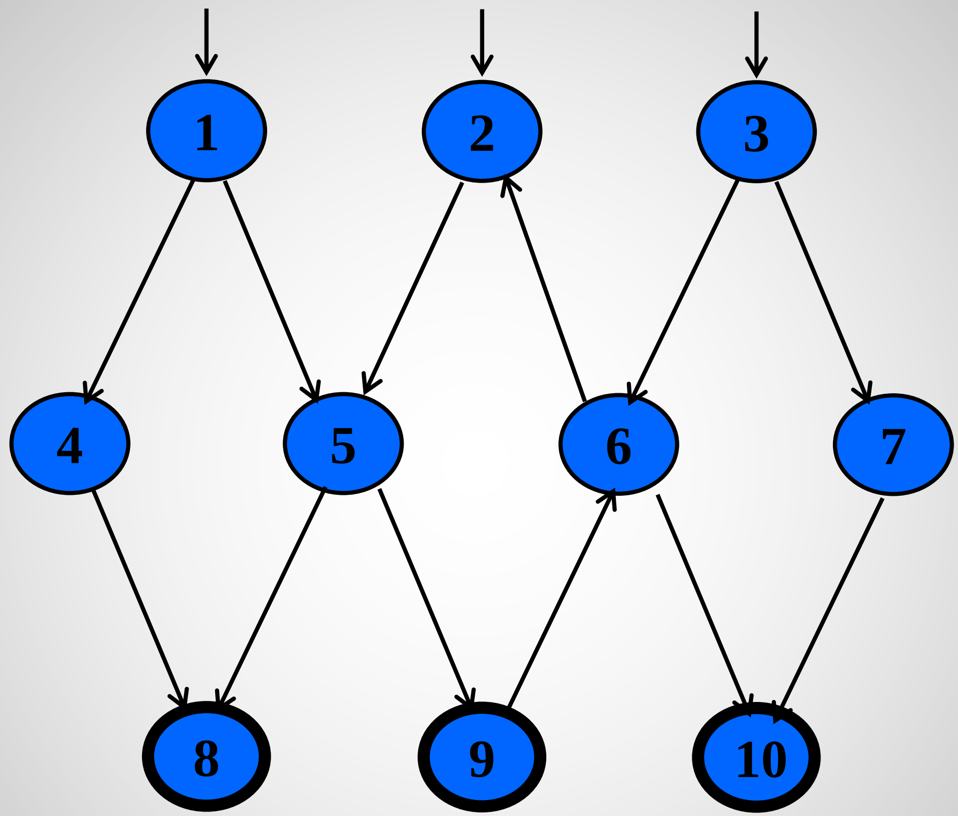
Three Example Graphs



$N_0 = \{ 1 \}$

$N_f = \{ 4 \}$

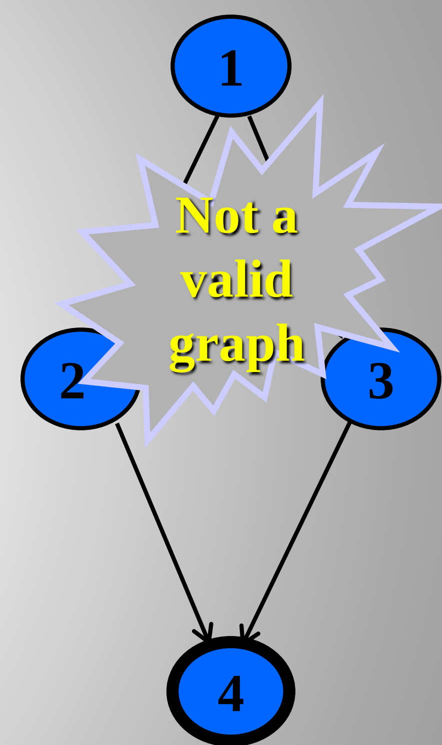
$E = \{(1,2),(1,3),..\}$



$N_0 = \{ 1, 2, 3 \}$

$N_f = \{ 8, 9, 10 \}$

$|E|=12$

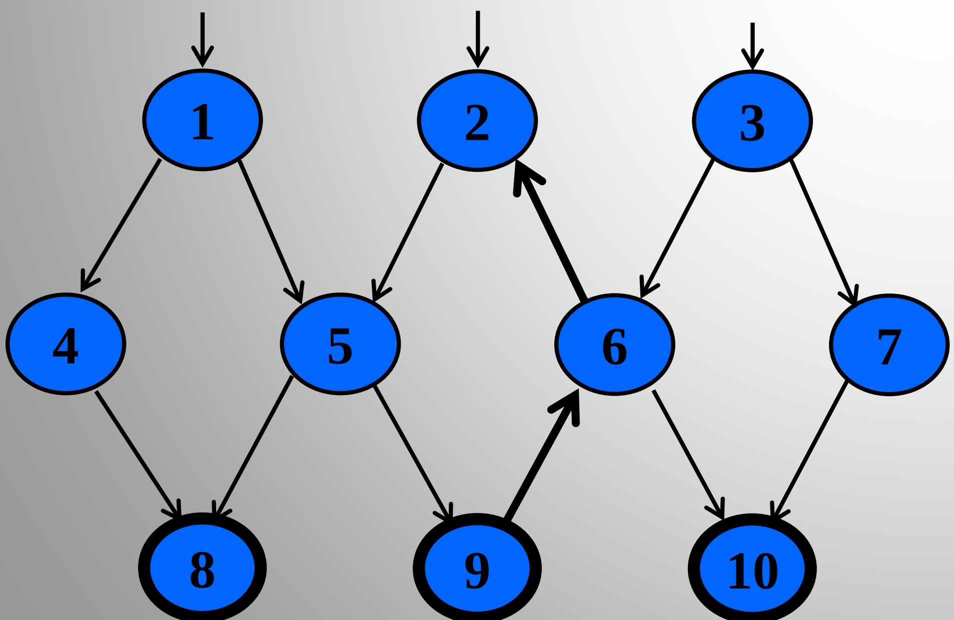


$N_0 = \{ \}$

$N_f = \{ 4 \}$

Paths in Graphs

- Path : A sequence of nodes – [n_1, n_2, …, n_M]
  - Each pair of nodes is an edge
- Length : The number of edges
  - A single node is a path of length 0
- Subpath : A subsequence of nodes in *p* is a subpath of *p*
- Reach (*n*) : Subgraph that can be reached from *n (includes the starting node)*



A Few Paths

[1, 4, 8]

[2, 5, 9, 6, 2]

[3, 7, 10]

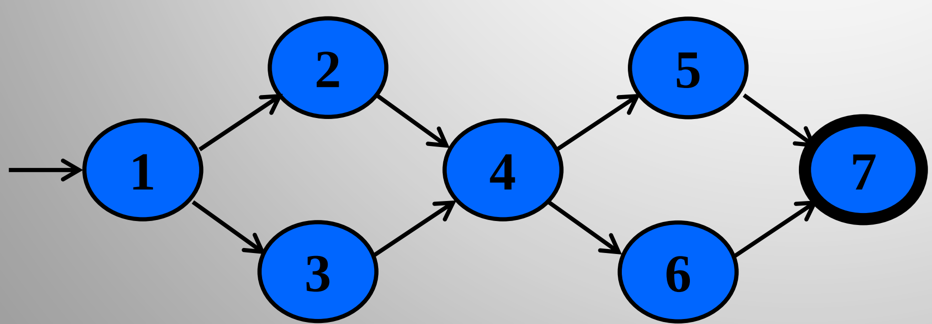
Reach (1) = { 1, 4, 5, 8, 9, 6, 2, 10 }

Reach (1, 3) = G

Reach([3,7]) = {3, 7, 10}

Test Paths and SESEs

- Test Path : <u>A path that starts at an initial node and ends at a final node</u>
- Test paths represent execution of test cases
  - Some test paths can be executed by many tests
  - Some test paths cannot be executed by any tests
- SESE graphs : All test paths start at a single node and end at another node
  - Single-entry, single-exit
  - N0 and Nf have exactly one node



<u>Double-diamond graph</u>
Four test paths
[1, 2, 4, 5, 7]
[1, 2, 4, 6, 7]
[1, 3, 4, 5, 7]
[1, 3, 4, 6, 7]

Visiting and Touring

- Visit : A test path *p* *visits* node *n* if *n* is in *p*

  A test path *p* *visits* edge e if e is in *p*
- Tour : A test path *p* *tours* subpath *q* if *q* is a subpath of *p*

Path [1, 2, 4, 5, 7]

Visits nodes 1, 2, 4, 5, 7

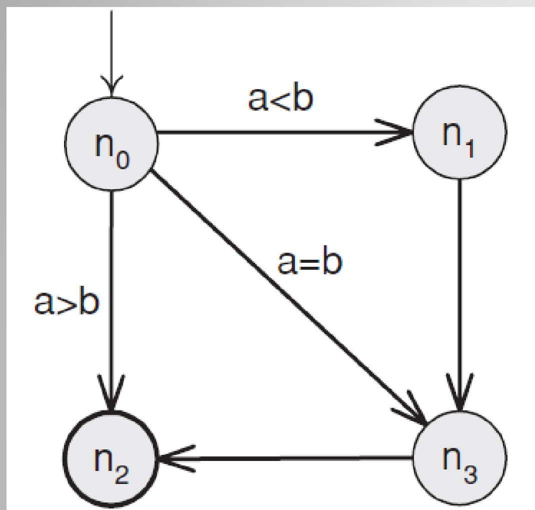
Visits edges (1, 2), (2, 4), (4, 5), (5, 7)

Tours subpaths [1, 2, 4], [2, 4, 5], [4, 5, 7], [1, 2, 4, 5], [2, 4, 5, 7]

Tests and Test Paths

- path (*t*) : The test path executed by test *t*
- path (*T*) : The set of test paths executed by the set of tests *T*

Test cases and Corresponding Test Paths



- Graph for testing the case with input integers a, b and output (a+b)

Map to

Test case t_1 : (a=0, b=1) $\longrightarrow$ [Test path p_1 : n_0, n_1, n_3, n_2]

Test case t_2 : (a=1, b=1) $\longrightarrow$ [Test path p_2 : n_0, n_3, n_2]

Test case t_3 : (a=2, b=1) $\longrightarrow$ [Test path p_3 : n_0, n_2]

Testing and Covering Graphs

- We use graphs in testing as follows :
  - Developing a model of the software as a graph
  - Requiring tests to visit or tour specific sets of nodes, edges or subpaths

- **Test Requirements (TR)** : Describe properties of test paths
  - ***Specific element of a software artifact that a test case must cover***
- Test Criterion : Rules that define test requirements

Testing and Covering Graphs

- **Structural Coverage Criteria** : Defined on a graph just in terms of nodes and edges
  - Has variously been called – statement coverage, block coverage, state coverage and **node coverage**

- **Data Flow Coverage Criteria** : Requires a graph to be annotated with references to variables

Structural Coverage Criteria

Node and Edge Coverage

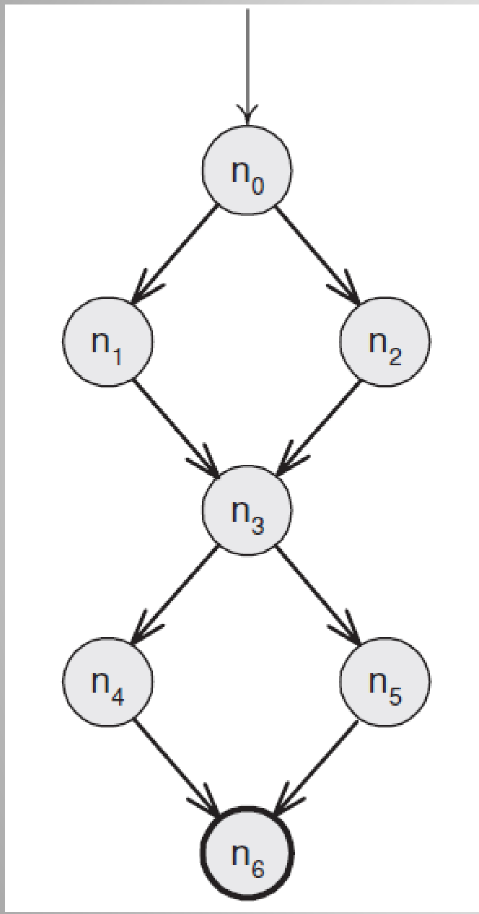
- The first (and simplest) two criteria require that each node and edge in a graph be executed

<u>Node Coverage (NC)</u> : Test set *T* satisfies node coverage on graph *G* iff for every syntactically reachable node *n* in *N*, there is some path *p* in *path(T)* such that *p* visits *n*.

- **This statement is a bit cumbersome, so we abbreviate it in terms of the set of test requirements**

<u>Node Coverage (NC)</u> : TR contains each reachable node in G.

Node Coverage Definition Example



- TR = $\{n_0, n_1, n_2, n_3, n_4, n_5, n_6\}$
- Test path
  - $P_1 = [n_0, n_1, n_3, n_4, n_6]$
  - $P_2 = [n_0, n_2, n_3, n_5, n_6]$
- Therefore, if test set T contains $\{t_1, t_2\}$, where $path(t_1) = p_1$ and $path(t_2) = p_2$
- Then T satisfies node coverage on G

Node and Edge Coverage

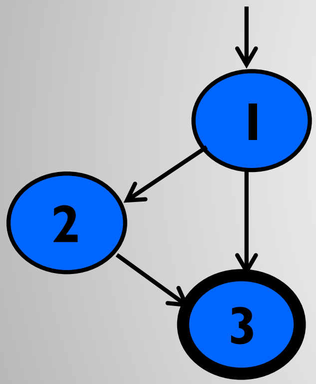
- Edge coverage is slightly stronger than node coverage

<u>Edge Coverage (EC)</u> : TR contains each reachable path of length up to 1, inclusive, in G.

- The phrase "*length up to 1*" allows for graphs with one node and no edges

Node and Edge Coverage

- NC and EC are only different when there is an edge and another subpath between a pair of nodes (as in an “if-else” statement)



<u>Node Coverage</u> : TR = { 1, 2, 3 }
Test Path = [1, 2, 3]

<u>Edge Coverage</u> :TR = { (1, 2), (1, 3), (2, 3) }
Test Paths = [1, 2, 3]
[1, 3]

Covering Multiple Edges

- Edge-pair coverage requires pairs of edges, or subpaths of length 2

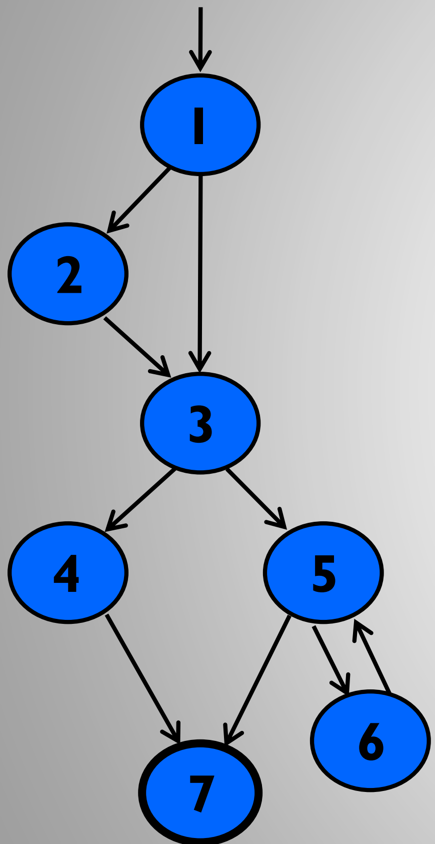
<u>Edge-Pair Coverage (EPC)</u> : TR contains each reachable path of length up to 2, inclusive, in G.

- The phrase "length up to 2" is used to include graphs that have less than 2 edges
- The logical extension is to require all paths …

<u>Complete Path Coverage (CPC)</u> : TR contains all paths in G.

- Unfortunately, this is impossible if the graph has a loop, so a weak compromise is to make the tester decide which paths:

Structural Coverage Example



Node Coverage

TR = { 1, 2, 3, 4, 5, 6, 7 }

Test Paths: [1, 2, 3, 4, 7] [1, 2, 3, 5, 6, 5, 7]

Edge Coverage

TR = { (1,2), (1, 3), (2, 3), (3, 4), (3, 5), (4, 7), (5, 6), (5, 7), (6, 5) }

Test Paths: [1, 2, 3, 4, 7] [1, 3, 5, 6, 5, 7]

Edge-Pair Coverage

TR = {[1,2,3], [1,3,4], [1,3,5], [2,3,4], [2,3,5], [3,4,7], [3,5,6], [3,5,7], [5,6,5], [6,5,6], [6,5,7] }

Test Paths: [1, 2, 3, 4, 7] [1, 2, 3, 5, 7] [1, 3, 4, 7] [1, 3, 5, 6, 5, 6, 5, 7]

Complete Path Coverage

Test Paths: [1, 2, 3, 4, 7] [1, 2, 3, 5, 7] [1, 2, 3, 5, 6, 5, 7] [1, 2, 3, 5, 6, 5, 6, 5, 7] [1, 2, 3, 5, 6, 5, 6, 5, 6, 5, 7] …

Loops in Graphs

- If a graph contains a loop, it has an infinite number of paths

- Thus, CPC is not feasible

- Attempts to “deal with” loops:
  – 1970s : Execute cycles once ([4, 5, 4] in previous example, informal)
  – 1980s : Execute each loop, exactly once (formalized)
  – 1990s : Execute loops 0 times, once, more than once (informal description)
  – 2000s : Prime paths

Simple Paths and Prime Paths

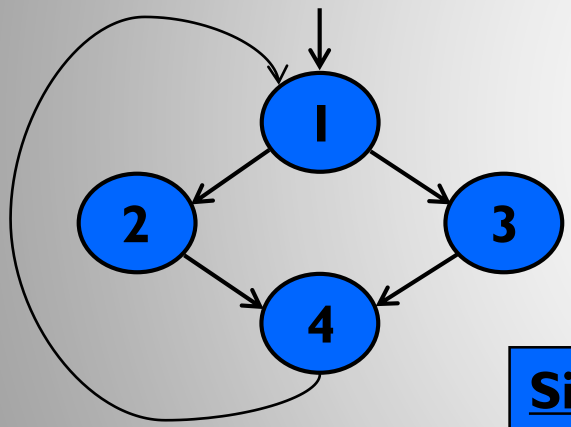
- <u>Simple Path</u>: *A path from node n_i to n_j is simple* ***if no node appears more than once****, except possibly the first and last nodes are the same*
  - No internal loops
  - A loop is a simple path

- For a coverage criterion, we would like to avoid enumerating the entire set of simple paths
  - List only maximal length simple paths

- <u>Prime Path</u>: *A simple path that does not appear as a proper subpath of any other simple path*
  - ***Maximal length simple path***

Simple Paths and Prime Paths

- Prime Path : *A simple path that does not appear as a proper subpath of any other simple path*
  - ***Maximal length simple path***
  - *Reduces the number of test cases for path coverage*



Simple Paths : [1,2,4,1], [1,3,4,1], [2,4,1,2], [2,4,1,3], [3,4,1,2], [3,4,1,3], [4,1,2,4], [4,1,3,4], [1,2,4], [1,3,4], [2,4,1], [3,4,1], [4,1,2], [4,1,3], [1,2], [1,3], [2,4], [3,4], [4,1], [1], [2], [3], [4]

Prime Paths : [2,4,1,2], [2,4,1,3], [1,3,4,1], [1,2,4,1], [3,4,1,2], [4,1,3,4], [4,1,2,4], [3,4,1,3]

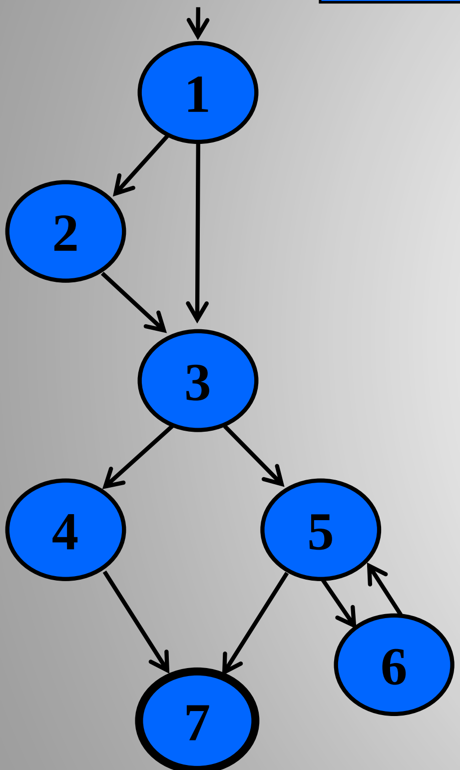
Prime Path Coverage

- A simple, elegant and finite criterion that requires loops to be executed as well as skipped

<u>Prime Path Coverage (PPC)</u> : TR contains each prime path in G.

- Will tour all paths of length 0, 1, …
- That is, it subsumes node and edge coverage
- PPC does NOT subsume EPC
  - If a node *n* has an edge to itself, EPC will require [*n*, *n*, *m*]
  - [*n*, *n*, *m*] is not prime

Simple & Prime Path Example



Simple paths

‘!’ means path
es

s path
es

Len 0
[1]
[2]
[3]
[4]
[5]
[6]
[7] !

Len 1
[1, 2]
[1, 3]
[2, 3]
[3, 4]
[3, 5]
[4, 7] !
[5, 7] !
[5, 6]
[6, 5]

Len 2
[1, 2, 3]
[1, 3, 4]
[1, 3, 5]
[2, 3, 4]
[2, 3, 5]
[3, 4, 7] !
[3, 5, 7] !
[3, 5, 6] !
[5, 6, 5] *
[6, 5, 7] !
[6, 5, 6] *

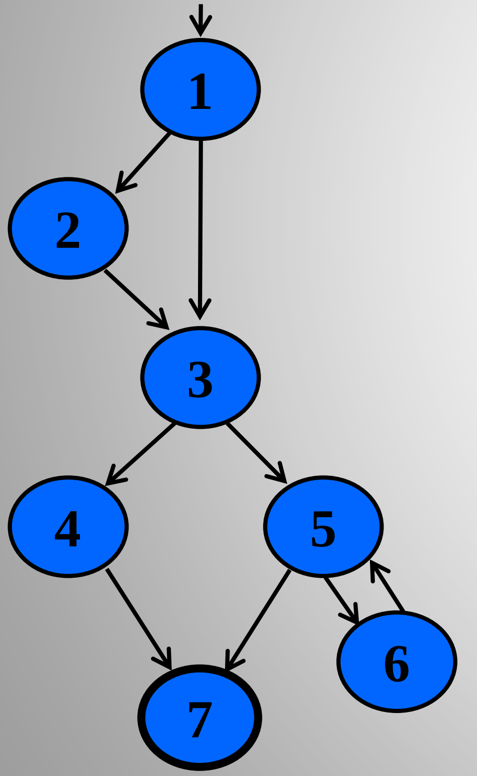
Len 3
[1, 2, 3, 4]
[1, 2, 3, 5]
[1, 3, 4, 7] !
[1, 3, 5, 7] !
[1, 3, 5, 6] !
[2, 3, 4, 7] !
[2, 3, 5, 6] !
[2, 3, 5, 7] !

Len 4
[1, 2, 3, 4, 7] !
[1, 2, 3, 5, 7] !
[1, 2, 3, 5, 6] !

Prime Paths

Prime Path Example

- The previous example has 38 simple paths
- Only nine *prime paths*



<u>Prime Paths</u>

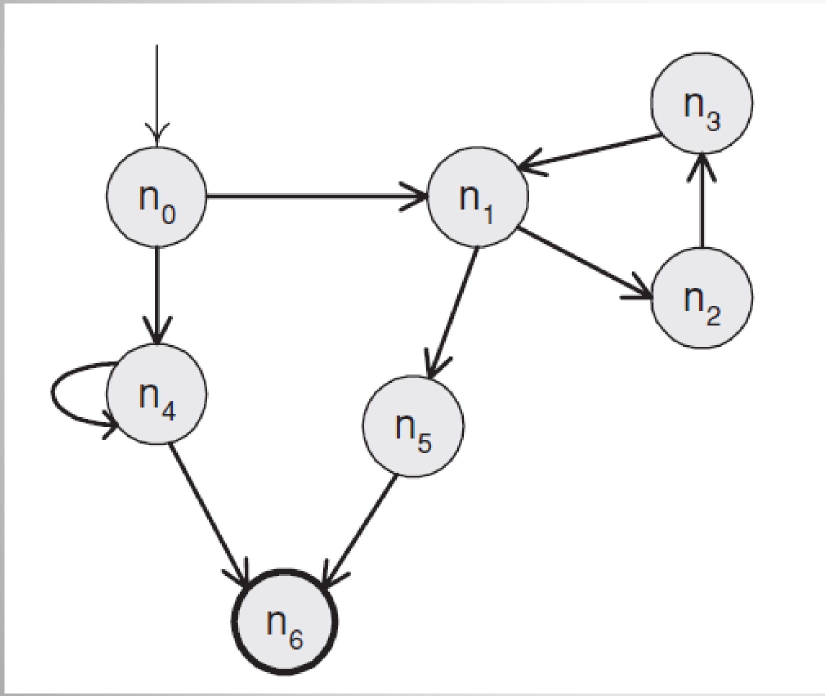
[1, 2, 3, 4, 7]
[1, 2, 3, 5, 7]
[1, 2, 3, 5, 6]
[1, 3, 4, 7]
[1, 3, 5, 7]
[1, 3, 5, 6]
[6, 5, 7]
[6, 5, 6]
[5, 6, 5]

Execute loop 0 times

Execute loop once

Execute loop more than once

Prime Path Example



Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

Simple paths of length 1 (9):

8) [0, 1]
9) [0, 4]
10) [1, 2]
11) [1, 5]
12) [2, 3]
13) [3, 1]
14) [4, 4] *
15) [4, 6] !
16) [5, 6] !

Prime Path Example (cont.)

Simple paths of length 2 (8):

17) [0, 1, 2]
18) [0, 1, 5]
19) [0, 4, 6] !
20) [1, 2, 3]
21) [1, 5, 6] !
22) [2, 3, 1]
23) [3, 1, 2]
24) [3, 1, 5]

Simple paths of length 3 (7):

25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
29) [2, 3, 1, 5]
30) [3, 1, 2, 3] *
31) [3, 1, 5, 6] !

Prime paths of length 4 (1):

32) [2, 3, 1, 5, 6]!

- There are 8 prime paths

14) [4, 4] *
19) [0, 4, 6] !
25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
30) [3, 1, 2, 3] *
32) [2, 3, 1, 5, 6]!

Round Trips

- Round-Trip Path : *A prime path that starts and ends at the same node*

<u>Simple Round Trip Coverage (SRTC)</u> : TR contains at least one round-trip path for each reachable node in G that begins and ends a round-trip path.

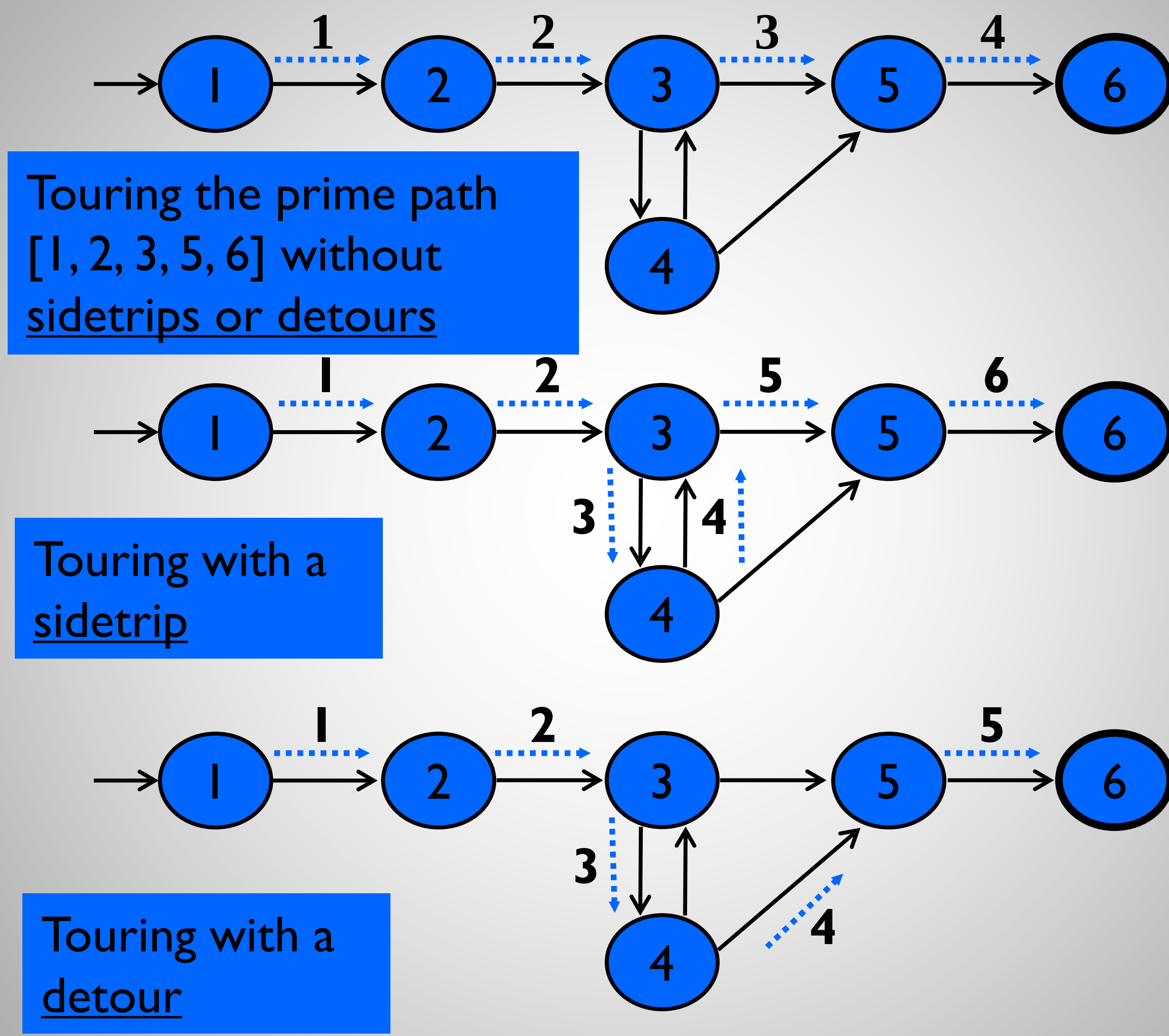
<u>Complete Round Trip Coverage (CRTC)</u> : TR contains all round-trip paths for each reachable node in G.

- These criteria omit nodes and edges that are not in round trips
- That is, they do not subsume edge-pair, edge, or node coverage

Touring, Sidetrips and Detours

- Prime paths do not have internal loops … test paths <u>might</u>
- Tour : *A test path p tours subpath q if q is a subpath of p*
- Tour With Sidetrips : *A test path p tours subpath q with sidetrips iff every edge in q is also in p in the same order*
  - The tour can include a sidetrip, as long as it comes back to the same node
- Tour With Detours : *A test path p tours subpath q with detours iff every node in q is also in p in the same order*
  - The tour can include a detour from node *ni*, as long as it comes back to the prime path at a successor of *ni*

Sidetrips and Detours Example

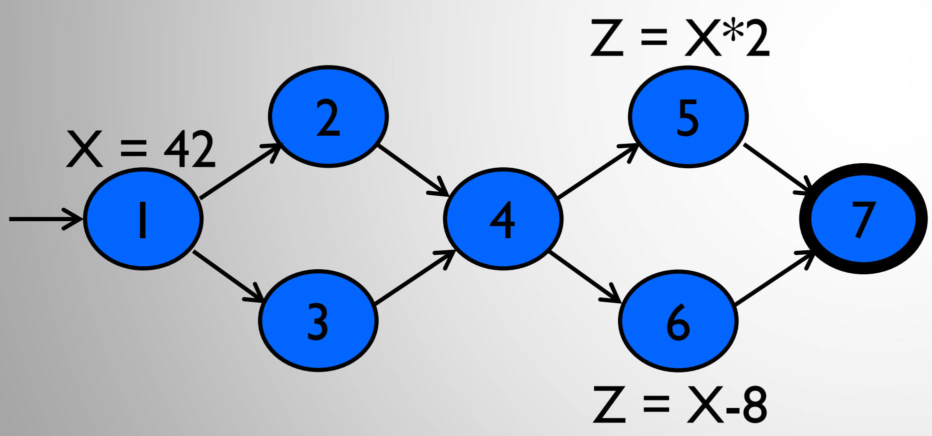


Data Flow Coverage Criteria

Data Flow Criteria

<u>Goal</u>: Try to ensure that values are computed and used correctly

- Definition (def) : A location where a value for a variable is stored into memory
- Use : A location where a variable's value is accessed



<u>Defs</u>: def (1) = {X}

def (5) = {Z}

def (6) = {Z}

<u>Uses</u>: use (5) = {X}

use (6) = {X}

The values given in defs should reach at least one, some, or all possible uses

DU Pairs and DU Paths

- def (n) or def (e) : The set of variables that are defined by node n or edge e
- use (n) or use (e) : The set of variables that are used by node n or edge e

- DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

- du-path : A simple subpath that is def-clear with respect to v from a def of v to a use of v
- du (n_i, n_j, v) – the set of du-paths from n_i to n_j
- du (n_i, v) – the set of du-paths that start at n_i

Data Flow Test Criteria

- First, we make sure every def reaches a use

<u>All-defs coverage (ADC)</u> : For each set of du-paths $S = du(n, v)$, TR contains at least one path d in S.

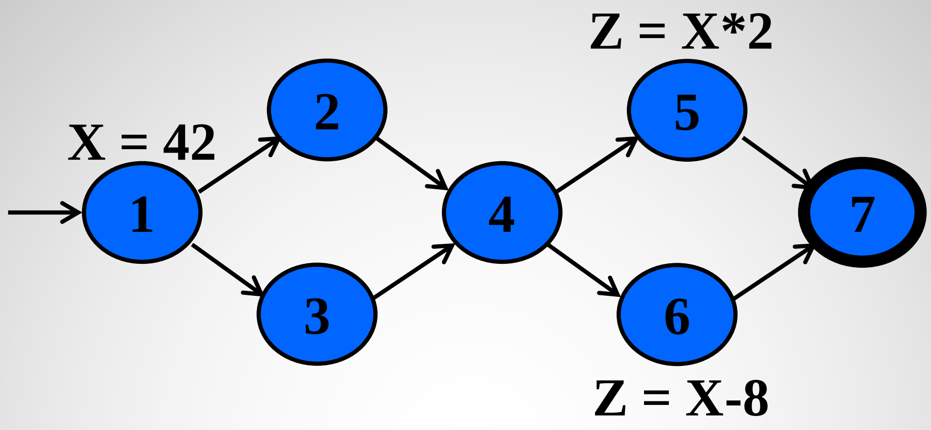
- Then we make sure that every def reaches all possible uses

<u>All-uses coverage (AUC)</u> : For each set of du-paths to uses $S = du(n_i, n_j, v)$, TR contains at least one path d in S.

- Finally, we cover all the paths between defs and uses

<u>All-du-paths coverage (ADUPC)</u> : For each set $S = du(ni, nj, v)$, TR contains every path d in S.

Data Flow Testing Example



All-defs for *X*
[1, 2, 4, 5]

All-uses for *X*
[1, 2, 4, 5]
[1, 2, 4, 6]

All-du-paths for *X*
[1, 2, 4, 5]
[1, 3, 4, 5]
[1, 2, 4, 6]
[1, 3, 4, 6]

Exercise

Graph I.

$N = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}$
$N_0 = \{0\}$
$N_f = \{7\}$
$E = \{(0,1),\ (1,2),\ (1,7),\ (2,3),\ (2,4),\ (3,2),$
$(4,5),\ (4,6),\ (5,6),\ (6,1)\}$
$def(0) = def(3) = use(5) = use(7) = \{x\}$

- (a) Draw the graph.
- (b) List all of the du-paths with respect to x. (Note: Include all du-paths, even those that are subpaths of some other du-path).

Graph Coverage Criteria Subsumption