Coverage Criteria -1

Graph Coverage 1

Structural Coverage Criteria

Coverage Criterion

- A Coverage Criterion is simply a recipe for generating test requirements in a systematic way

In other words:

- Coverage Criterion: A coverage criterion is a rule or collection of rules that impose test requirements on a test set.

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
- Test Criterion (C): Rules that define test requirements
- Satisfaction : i)A set TR of test requirements for a criterion C is given

  ii) A set of tests T satisfies C on a graph if and only if for every test requirement in TR

  iii) There is a test path in path(T) that meets the test requirement tr

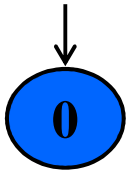
Testing and Covering Graphs

- Structural Coverage Criteria : Defined on a graph just in terms of nodes and edges

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

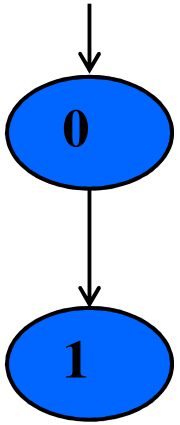
Paths of Length 1 and 0

- A graph with only one node will not have any edges



- Edge Coverage needs to require Node Coverage on this graph

- Edge Coverage will not subsume Node Coverage
  - So we define "length up to 1" instead of simply "length 1"



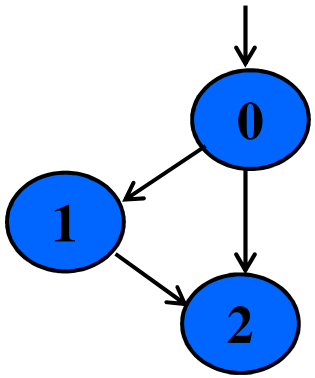
- We have the same issue with graphs that only have one edge – for Edge Pair Coverage …

Node and Edge Coverage

- Edge coverage is slightly stronger than node coverage

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

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



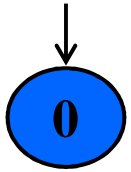
Node Coverage (NC) : TR = { 0, 1, 2 }
Test Path = [0, 1, 2]

Edge Coverage(EC) : TR = { (0,1), (0, 2), (1, 2) }
Test Paths = [0, 1, 2]
[0, 2]

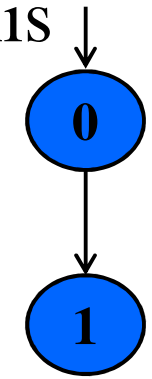
- NC and EC are only different when there is an edge and another subpath between a pair of nodes (as in an "if-else" statement)

Paths of Length 1 and 0

- A graph with only one node will not have any edges



- Edge Coverage needs to require Node Coverage on this graph



- Edge Coverage will not subsume Node Coverage.

Therefore:

we define "length up to 1" instead of simply "length 1"

- We have the same issue with graphs that only have one edge – for Edge Pair Coverage …

Covering Multiple Edges

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

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

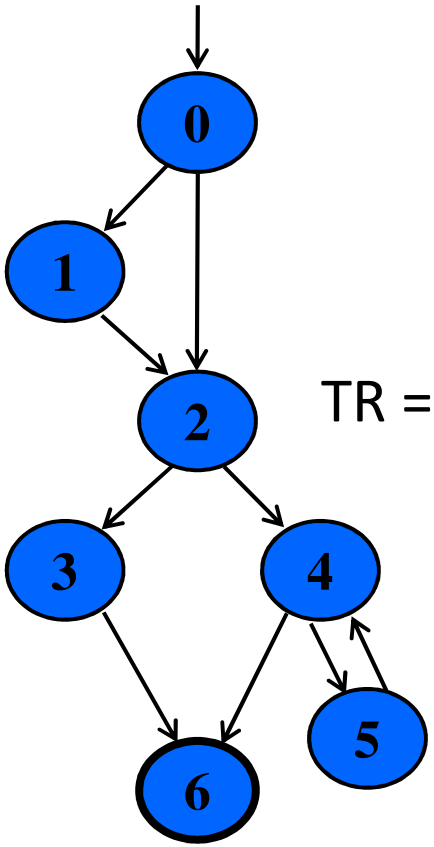
- The "length up to 2" is used to include graphs that have less than 2 edges
- The logical extension is to require all paths …

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

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

> Specified Path Coverage (SPC) : TR contains a set S of test paths, where S is supplied as a parameter.

Structural Coverage Examples



Node Coverage

TR = { 0, 1, 2, 3, 4, 5, 6 }

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 5, 4, 6]

Edge Coverage

TR = { (0,1), (0,2), (1,2), (2,3), (2,4), (3,6), (4,5), (4,6), (5,4) }

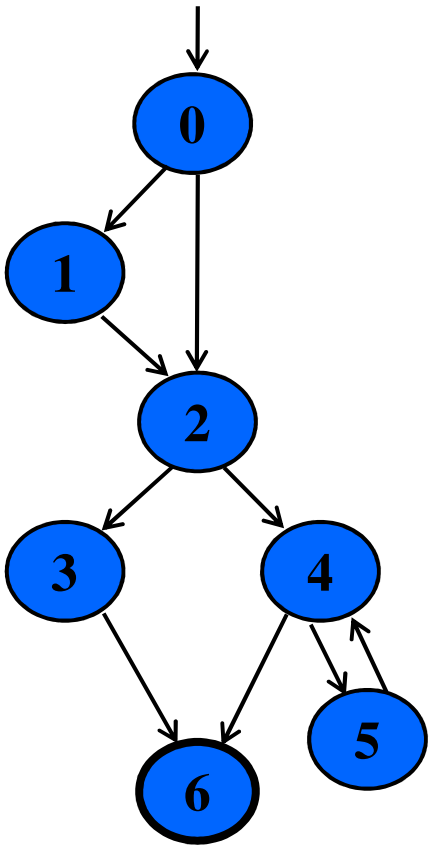
Test Paths: [0, 1, 2, 3, 6] [0, 2, 4, 5, 4, 6]

path (t_1)= [0, 1, 2, 3, 6]

path (t_2)=[0, 2, 4, 5, 4, 6]

T={t_1 , t_2}

Edge-Pair Coverage



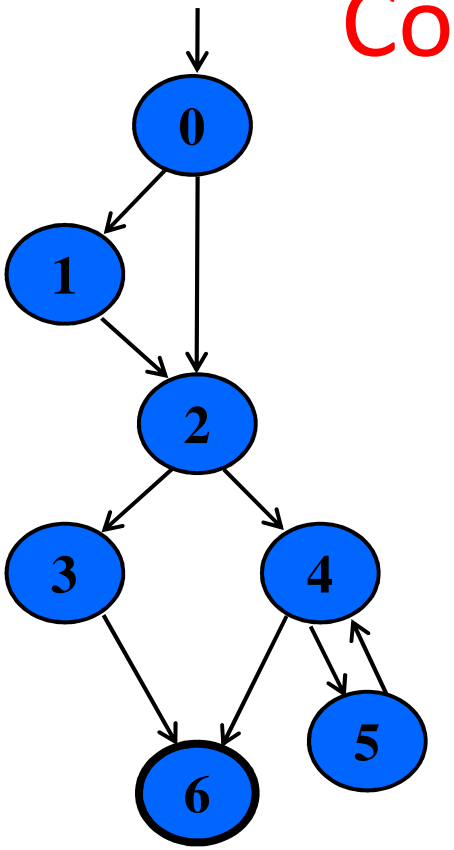
TR = { [0,1,2], [0,2,3], [0,2,4], [1,2,3], [1,2,4], [2,3,6], [2,4,5], [2,4,6], [4,5,4], [5,4,5], [5,4,6] }

path (t_1)= [0, 1, 2, 3, 6]
path (t_2)= [0, 1, 2, 4, 6]
path (t_3)=[0, 2, 3, 6]
path (t_4)= [0, 2, 4, 5, 4, 5, 4, 6]

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 6] [0, 2, 3, 6] [0, 2, 4, 5, 4, 5, 4, 6]

T={t_1 , t_2 , t_3 , t_4 }

Complete Path Coverage



Complete Path Coverage (CPC) : TR contains all paths in G.

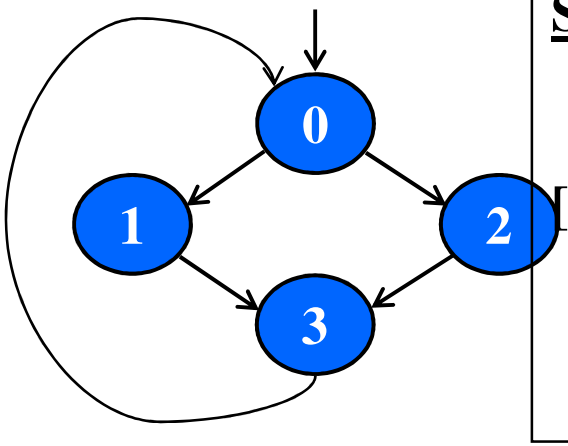
This is impossible if the graph has a loop, so a weak compromise is to make the tester decide which paths

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 6] [0, 1, 2, 4, 5, 4, 6] [0, 1, 2, 4, 5, 4, 5, 4, 6] [0, 1, 2, 4, 5, 4, 5, 4, 5, 4, 6] .....

Simple Paths

Simple Path : 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

A loop is a simple path if it has no internal loops and it includes all other subpaths



Simple Paths : [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1], [0, 1, 3],[0, 2, 3],[1, 3, 0],[2, 3, 0],[3, 0, 1], [3, 0, 2], [0, 1], [0, 2], [1, 3], [2, 3], [3, 0], [0], [1], [2], [3]

Prime Path

- Prime paths are ***maximal length*** **simple paths**
  - A simple path from node n_i to n_j is a *prime path* if it is simple and does not appear as a proper subpath of any other simple path.
- Prime paths do not have any internal loops,
  - But the entire path may be a loop.

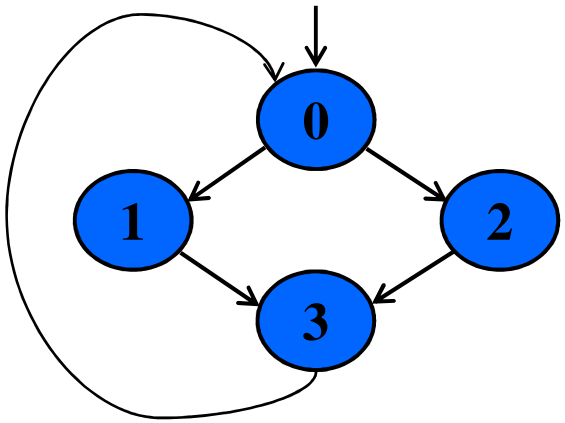
Prime Paths and Prime Path Coverage

- Prime path (PP) gives strong coverage of loops without requiring an infinite number of paths
- In prime path coverage, tests must tour each prime path in the graph *G*.
- Prime path coverage requires touring all subpaths of length 0 (all nodes), of length 1 (all edges), length 2, 3, etc.
- Prime path subsumes node coverage, edge coverage and edge-pair coverage
- In prime path coverage, tests must tour each prime path in the graph *G*.

Prime Paths: Example I

Prime Paths : [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1]

Accept that you wrote all simple paths previously



Prime paths are maximal length simple paths

Example 2: Simple Paths

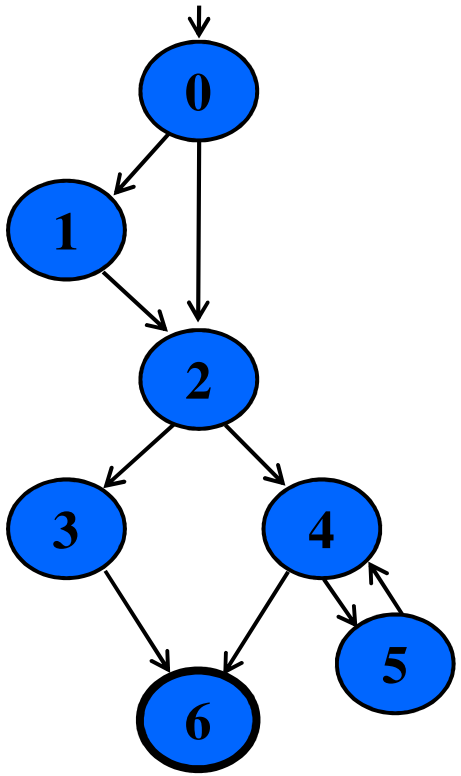
Len 0
[0]
[1]
[2]
[3]
[4]
[5]
[6] !

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

Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

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

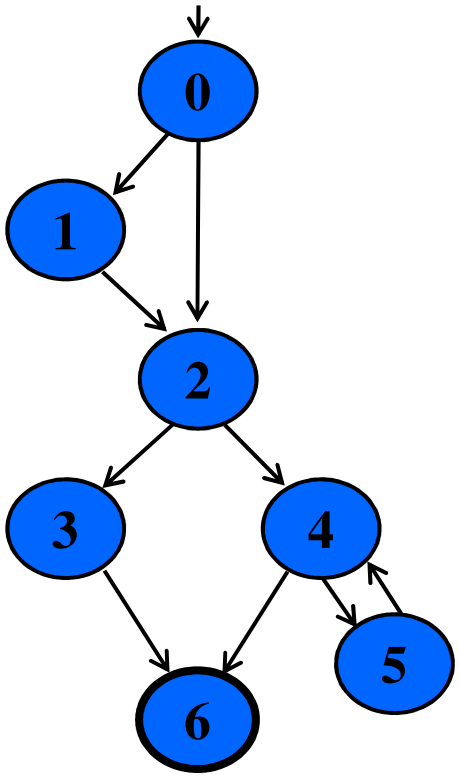
Len 4
[0, 1, 2, 3, 6] !
[0, 1, 2, 4, 6] !
[0, 1, 2, 4, 5] !



! means path terminates
* means path cycles

Prime Path : Example 2

Write all signed simple patsh as ! and *, then eliminate the repeted subpaths to find the prime paths.



Prime Paths

[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

Execute loop 0 times

Execute loop once

Execute loop more than once