Software
Verification & Validation

Natthapong Jungteerapanich

Graph Coverage

Acknowledgement

- Some slides in this lecture are adapted from

  - **Paul Ammann and Jeff Offutt's** slides for their textbook **"Introduction to Software Testing".** Cambridge University Press, 2008.

  - **Lee Copeland's** "**A Practitioner's Guide to Software Test Design**". Artech House, 2004.

Graphs for Test Design

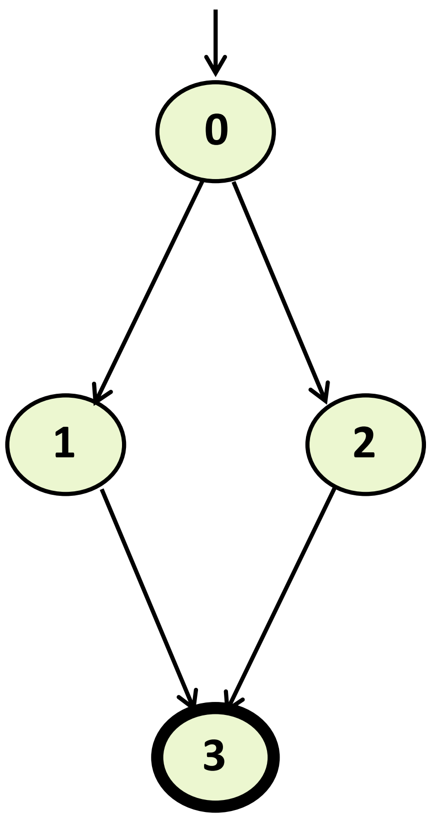
- Graphs are the most commonly used structure for testing
- Graphs can come from many sources
  - Control flow graphs
  - Design structure
  - State machines
  - Use cases
- Tests usually are intended to “cover” the graph in some way

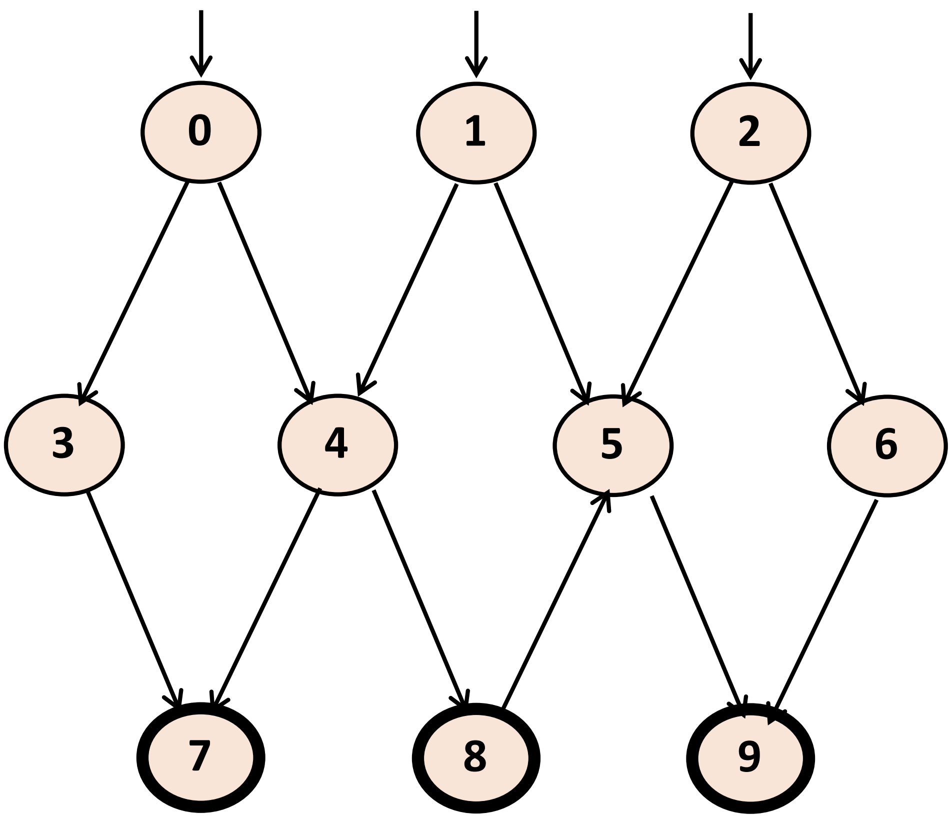
Definition of a Graph

For our purpose, we will be using a directed graph with one or more initial nodes and one or more final nodes. Precisely, we define a graph to be a structure (N, N_0, N_f, E) where

- N is a non-empty set of nodes,
- $N_0 \subseteq N$ is a non-empty set of initial nodes,
- $N_f \subseteq N$ is a non-empty set of final nodes,
- $E \subseteq N \times N$ is a set of edges, each edge from one node to another
  - If (n_i, n_j) is in E, then n_i is called a predecessor of n_j and n_j is called a successor of n_i

Three Example Graphs





$N_0 = \{ 0 \}$

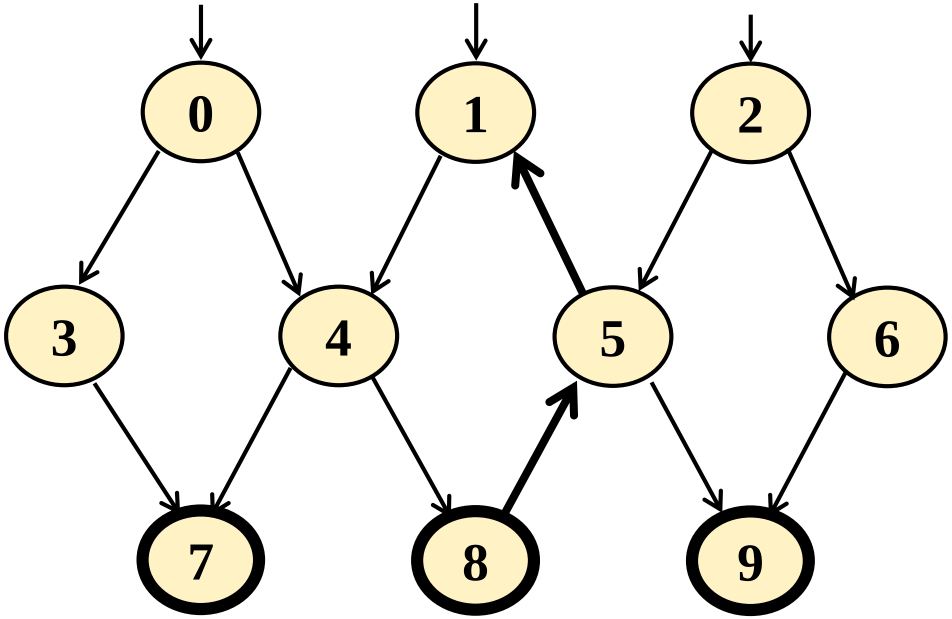
$N_f = \{ 3 \}$

$N_0 = \{ 0, 1, 2 \}$

$N_f = \{ 7, 8, 9 \}$

Paths in Graphs

- Path : A sequence of nodes – [n_1, n_2, …, n_M] such that each pair of adjacent nodes (n_i, n_{i+1}) is an edge.
- Length : The number of edges
  - A single node is a path of length 0
- Subpath : A subsequence of consecutive nodes in *p* is a subpath of *p*
- Reach(*n*) : The set of nodes that can be reached from *n*
- Reach(*A*) : The set of nodes that can be reached from some node in set A



Paths

[0, 3, 7]

[1, 4, 8, 5, 1]

[2, 6, 9]

Reach (0) = { 0, 3, 4, 7, 8, 5, 1, 9 }

Reach ({0, 2}) = G

Visiting and Touring

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

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

For example, the path [0, 1, 3, 4, 6]

visits nodes 0, 1, 3, 4, 6

visits edges (0, 1), (1, 3), (3, 4), (4, 6)

And tours paths [0, 1, 3], [1, 3, 4], [3, 4, 6], [0, 1, 3, 4], [1, 3, 4, 6]

Tests and Test Paths

- A test path in a graph is a path in the graph following the execution of some test case.
- path (t) : The test path executed by test t
- path (T) : The set of test paths executed by the set of tests T
- In a deterministic program, each test executes one and only one test path
- A location in a graph (node or edge) can be reached from another location if there is a sequence of edges from the first location to the second
  - *Syntactic reach* : A subpath exists in the graph
  - *Semantic reach* : A test exists that can execute that subpath

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

- 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 *(this will be studied in subsequent lectures)*

Node and Edge Coverage

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

<u>Node Coverage Criterion (NC)</u>: Test suite *T* satisfies *node coverage* on graph *G* iff, for every node *n*, some path in *path(T)* visits *n*.

Given a test suite T, the degree of node coverage of T can be calculated form:

$$C_{NC} = \frac{Number\ of\ nodes\ visited\ by\ some\ test\ path}{Number\ of\ nodes} \times 100\%$$

Node and Edge Coverage

- Edge coverage is slightly stronger than node coverage

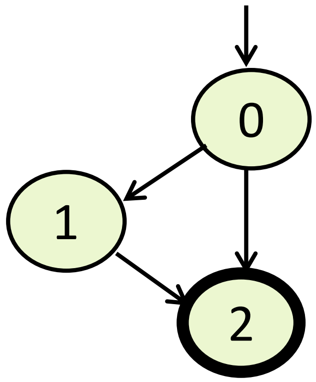
<u>Edge Coverage Criterion (EC)</u> : Test suite *T* satisfies *edge coverage* on graph *G* iff, for every edge *(m, n)*, some path in *path(T)* visits *(m, n).*

Given a test suite T, the degree of edge coverage of T can be calculated form:

$$C_{EC} = \frac{Number\ of\ edges\ visited\ by\ some\ test\ path}{Number\ of\ edges} \times 100\%$$

Node and Edge Coverage

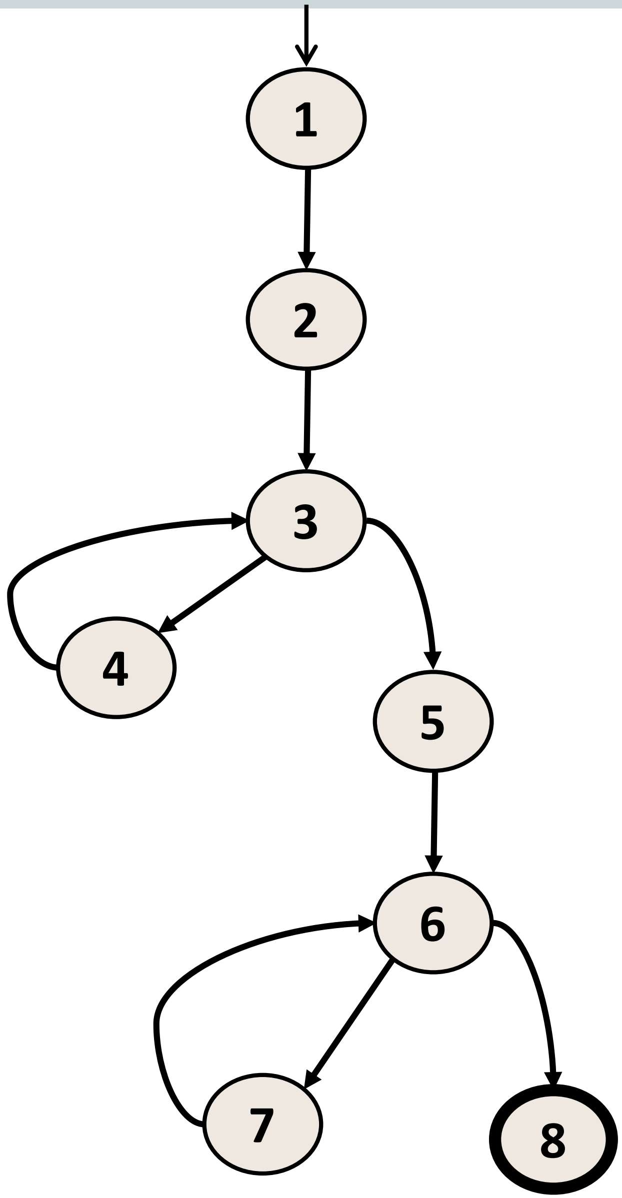
- NC and EC are only different when there is an edge and another subpath between a pair of nodes



<u>Node Coverage</u> : Nodes = { 0, 1, 2 }
Test Path = [0, 1, 2]

<u>Edge Coverage</u> : Edges = { (0,1), (0, 2), (1, 2) }
Test Paths = [0, 1, 2]
[0, 2]

Control Flow TRs and Test Paths – EC

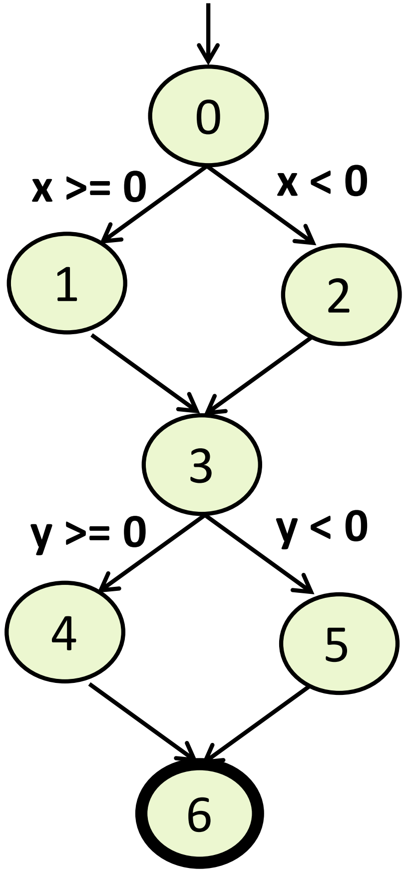


Edge Coverage	
Edges	**Test Path**
A. (1, 2)	[1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
B. (2, 3)	
C. (3, 4)	
D. (3, 5)	
E. (4, 3)	
F. (5, 6)	
G. (6, 7)	
H. (6, 8)	
I. (7, 6)	

Limitation of Edge Coverage

- **Edge coverage** requires that all alternatives in each choice node in the graph must be chosen in some test path.

- However, in graph with consecutive choice nodes, edge coverage criteria does **not** require a test path for each **combination of alternatives** of the choice nodes.



- For example, in this graph, nodes ***0*** and ***3*** are choice nodes.
- Only two paths are required to covered every edge. For example, ***[0, 1, 3, 4, 6]*** and ***[0, 2, 3, 5, 6].***
- The first path covers the case where ***x>=0*** and ***y>=0***. The second path covers the case where ***x<0*** and ***y<0***.
- But no paths cover the case where **x>=0** and **y<0** nor the case where **x<0** and **y>=0**.
- To covers these cases, we need to add more paths. For example, ***[0, 1, 3, 5, 6]*** and ***[0, 2, 3, 4, 6].***
- We will look at some coverage criteria which are more comprehensive than edge coverage.

Edge-Pair Coverage

- Edge-pair coverage is a stronger version than edge coverage

<u>Edge-Pair Coverage Criterion (EC)</u> : Test suite *T* satisfies *edge-pair coverage* on graph *G* iff, for every path p of length up to 2, some path in *path(T)* tours p.

Given a test suite T, the degree of edge coverage of T can be calculated form:

$$C_{EC} = \frac{Number\ of\ paths\ of\ length\ up\ to\ 2\ toured\ by\ some\ test\ path}{Number\ of\ paths\ of\ length\ up\ to\ 2} \times 100\%$$

Complete Path Coverage

- The Complete Path Coverage requires that every possible path in the graph is toured by some test path.

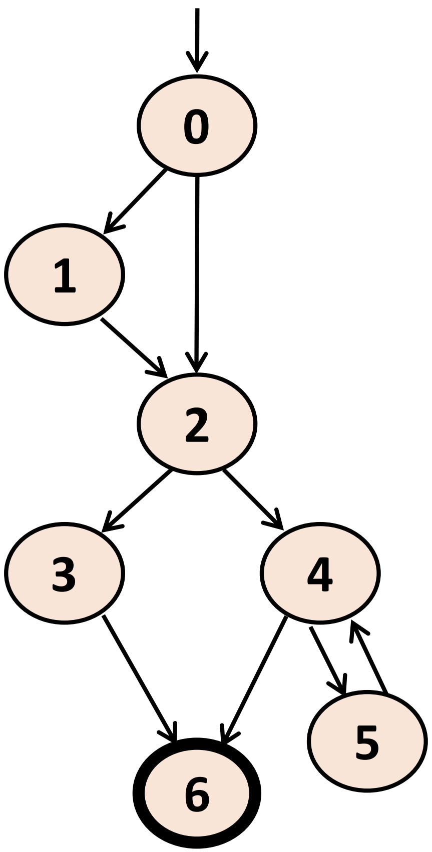
<u>Complete Path Coverage (CPC)</u> : Test suite *T* satisfies *complete path coverage* on graph *G* iff, for every path *q*, some path in *path(T)* tours *q*.

Given a test suite T, the degree of complete path coverage of T can be calculated form:

$$C_{CPC} = \frac{Number\ of\ paths\ toured\ by\ some\ test\ path}{Number\ of\ paths} \times 100\%$$

- Unfortunately, this is impossible if the graph has a loop because such graph will have infinitely many paths.

Structural Coverage Example



Node Coverage

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

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

Edge Coverage

Edges = { (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]

Complete Path Coverage

Paths = { [0], [0,1], [0,2], [0,1,2], [1,2], [1,2,3], … }

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] …

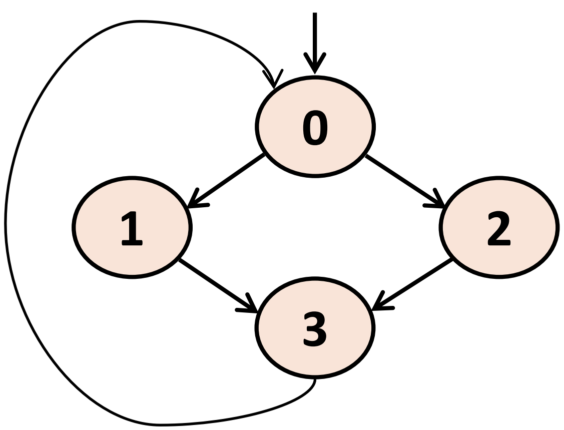
Loops in Graphs

- If a graph contains a loop, it has an infinite number of paths
- Thus, Complete Path Coverage is not feasible
- History of attempts to “deal with” loops:
  - Years 1980s : Execute each loop, exactly once ([4, 5, 4] in previous example)
  - Years 1990s : Execute loops 0 times, once, twice, …, up to specified limit
  - Years 2000s : *Prime paths*

Simple Paths and Prime Paths

- Simple Path : A path from node ***m*** to ***n*** is simple if no node appears more than once, except possibly the first and last nodes are the same
  - No internal loops,
  - But the path it self may be a loop.

- Prime Path : A simple path that does not appear as a proper subpath of any other simple path



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 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]

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> : Test suite *T* satisfies *prime path coverage* on graph *G* iff, for every prime path *q*, some path in *path(T)* tours *q* with sidetrips allowed.

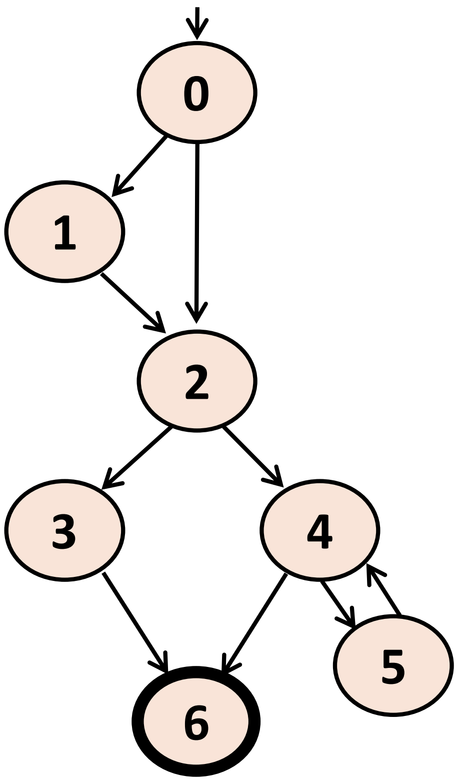
- Prime Path Coverage subsumes node and edge coverage, i.e. if a test set satisfies prime path coverage, it also satisfies node and edge coverage.

Given a test suite T, the degree of prime path coverage of T can be calculated form:

$$C_{PPC} = \frac{Number\ of\ prime\ paths\ toured\ by\ some\ test\ path}{Number\ of\ prime\ paths} \times 100\%$$

Prime Path Example

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



Prime Paths

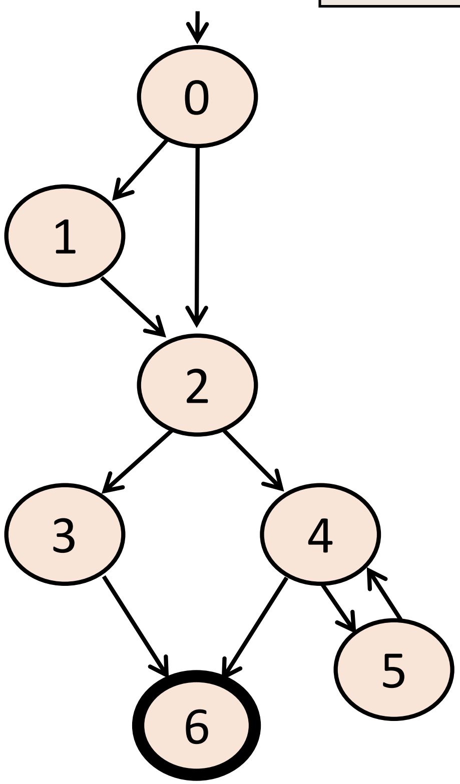
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

Execute loop 0 times

Execute loop once

Execute loop more than once

How to Find Prime Paths



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]

‘!’ means path
es

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] *

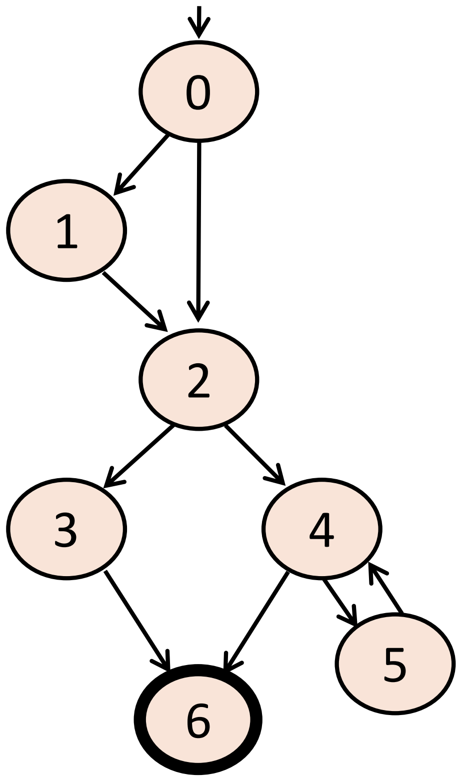
Len 3
[0, 1, 2, 3]
[0, 1, 2, 4]
[0, 2, 3, 6] !
[0, 2, 4, 6] !
[0, 2, 4, 5] !
[1, 2, 3, 6] !
[1, 2, 4, 5] !
[1, 2, 4, 6] !

ns path
es

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

Prime Paths

Prime Path Coverage



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]

- An example of a set of test paths that covers all prime paths:

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