

Introduction to Software Testing Graph Coverage (Ch. 7.1 & 7.2)

Software Testing & Maintenance

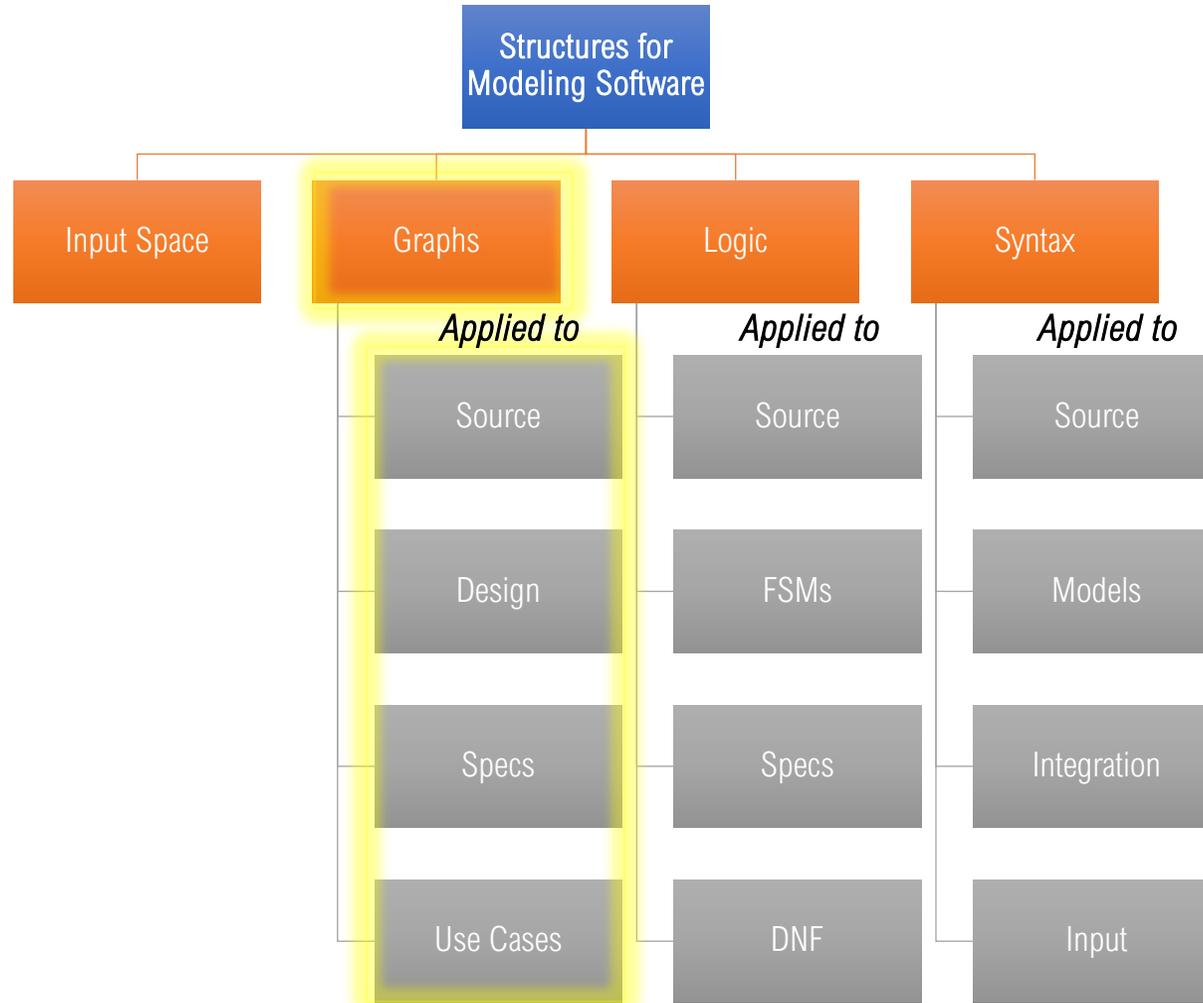
SWE 437/637

go.gmu.edu/SoftwareTestingFall24

Dr. Brittany Johnson-Matthews

(Dr. B for short)

Graph Coverage



Covering Graphs (7.1)

Graphs are the most **commonly** used structure for testing

Graphs can come from many sources

- Control flow graphs
- Design structure
- FSMs and state charts
- Use cases

Tests usually are intended to "**cover**" the graph in some way

What is a graph?

A set N of **nodes**, N is not empty

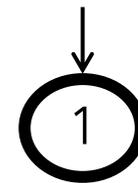
A set N_0 of **initial nodes**, N_0 is not empty

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**

Is this a graph?



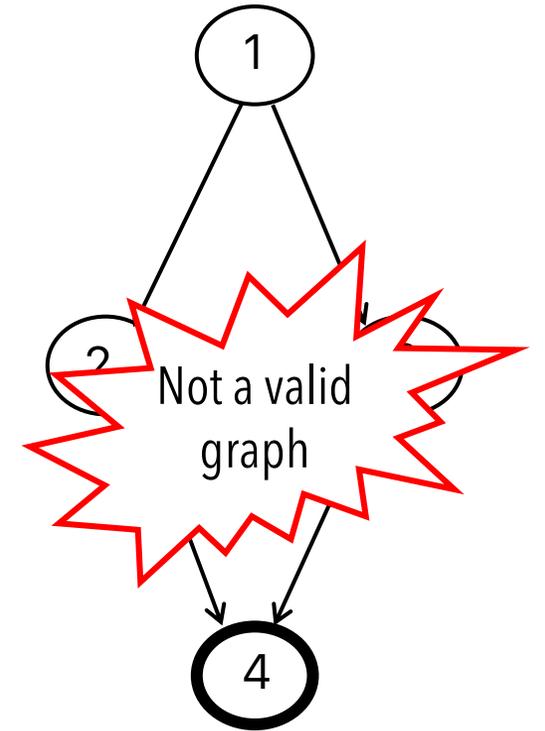
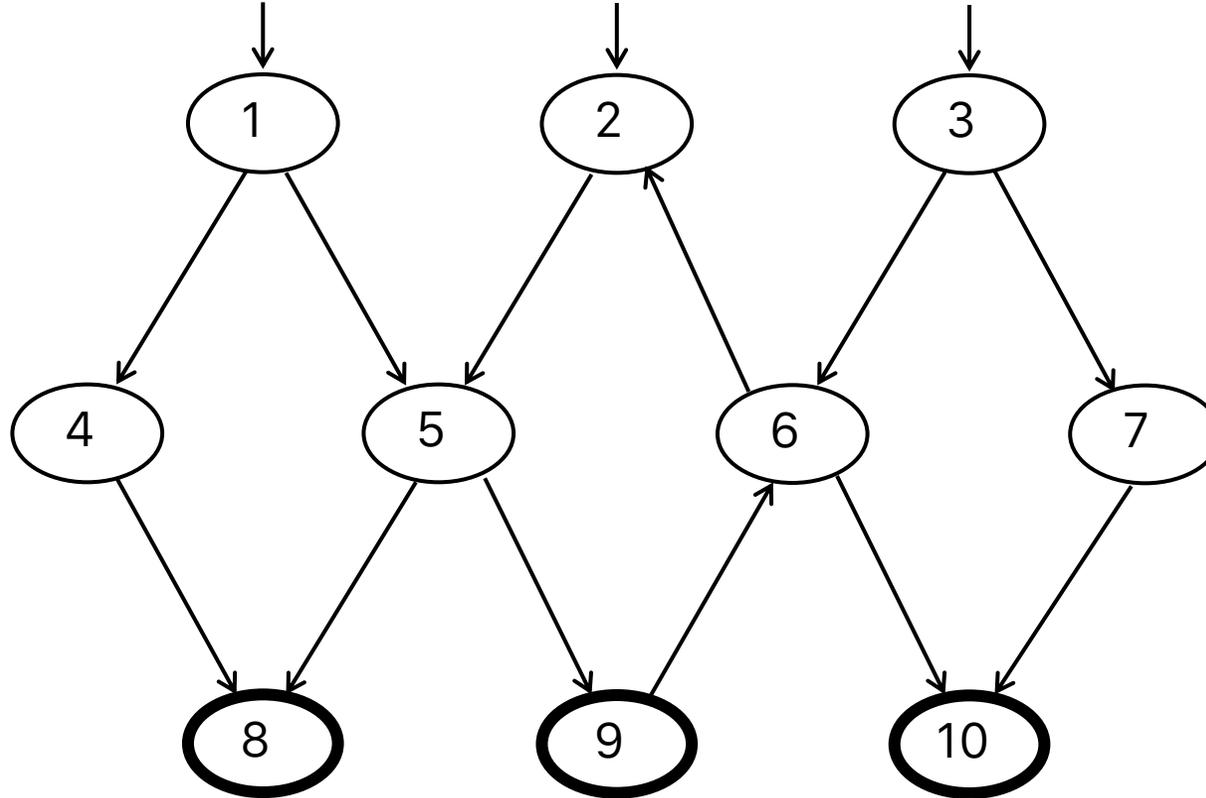
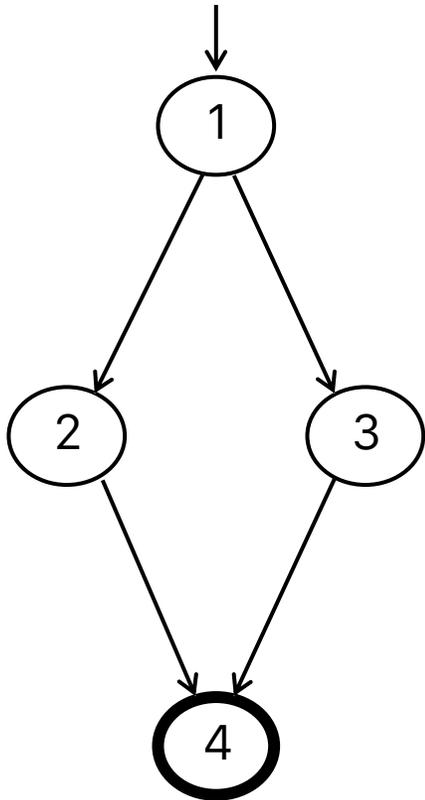
$$N_0 = \{ 1 \}$$

$$N_f = \{ 1 \}$$

$$E = \{ \}$$



Example graphs



$\mathbf{N}_0 = \{1\}; \mathbf{N}_f = \{4\}$
 $E = \{(1,2), (1,3), (2,4), (3,4)\}$

$\mathbf{N}_0 = \{1, 2, 3\}; \mathbf{N}_f = \{8, 9, 10\}$
 $E = \{(1,4), (1,5), (2,5), (3,6), (3,7), (4,8), (5,8), (5,9), (6,2), (6,10), (7,10), (9,6)\}$

$\mathbf{N}_0 = \{\}; \mathbf{N}_f = \{4\}$
 $E = \{(1,2), (1,3), (2,4), (3,4)\}$

Paths in graphs

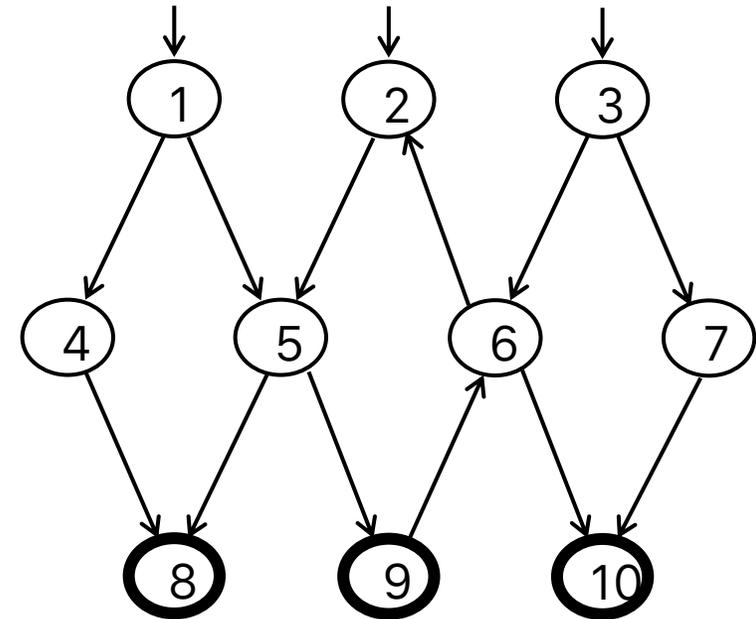
Path: A sequence of nodes - $[n_1, n_2, \dots, n_M]$

-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



A Few Paths

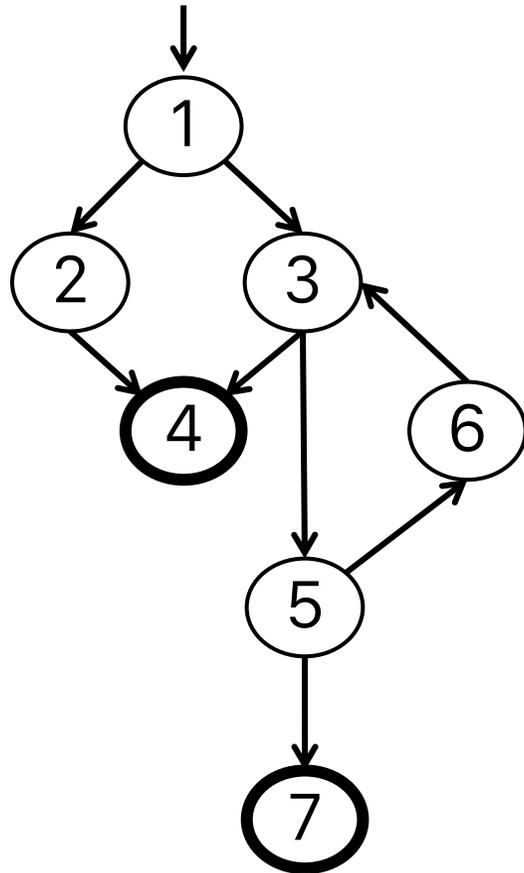
[1, 4, 8]

[2, 5, 9, 6, 2]

[3, 7, 10]

In-class Exercise

Graph Definitions



Answer the following questions for the graph on the left

1. How many nodes are in the graph?
2. How many edges are in the graph?
3. What is the set of initial nodes?
4. What is the set of final nodes?
5. Write two paths in the graph.
6. Write a subpath of one of your paths.



Test paths and SESEs

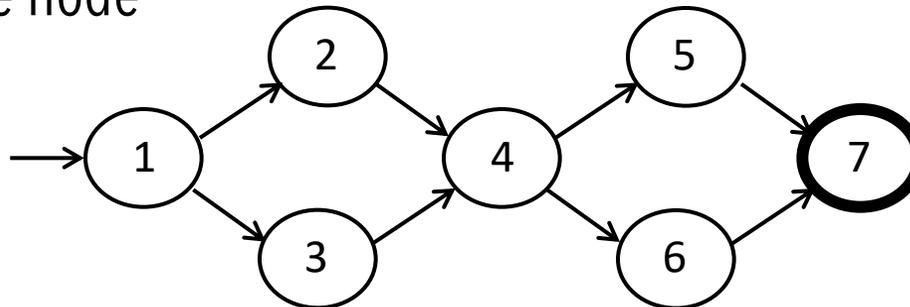
Test path: A path that starts at an initial node and ends at a final node

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
- N_0 and N_f have exactly one node



Double-diamond graph

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

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

(Technically, each edge is also a subpath)

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

Each test executes **one and only one** test path

- Complete execution from a start node to a final node

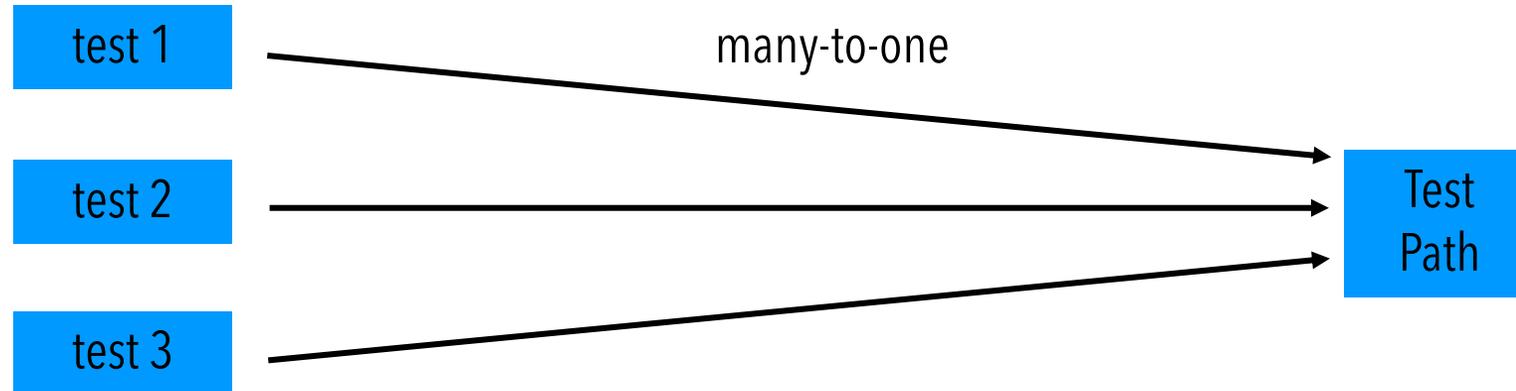
A location in a graph (node or edge) can be **reached** from another location if there is a sequence of edges from the first location to the second

- Syntactic reach**: a subpath exists in the graph

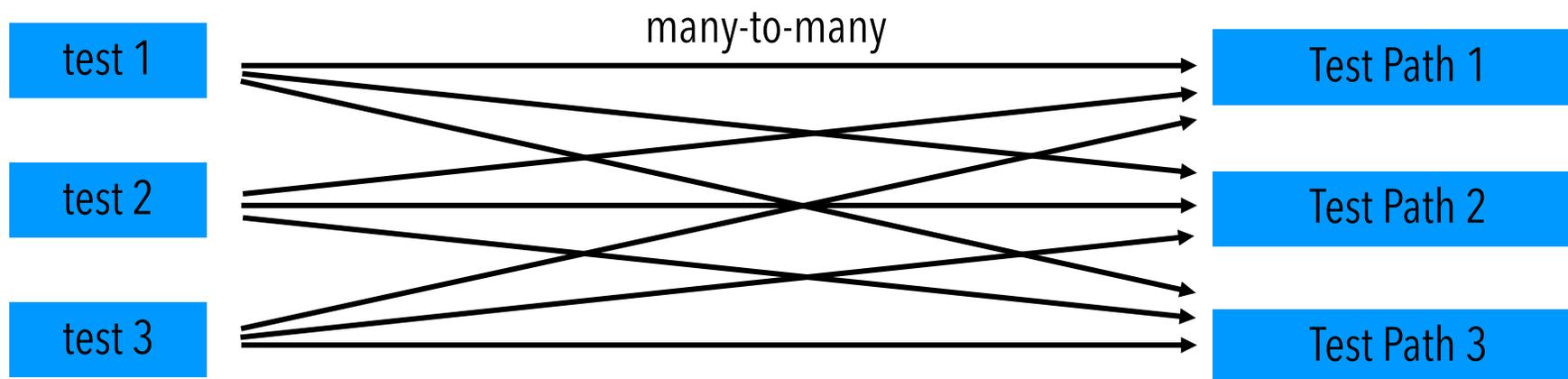
- Semantic reach**: a test exists that can execute that subpath

- This distinction becomes important in **section 7.3**

Tests and test paths



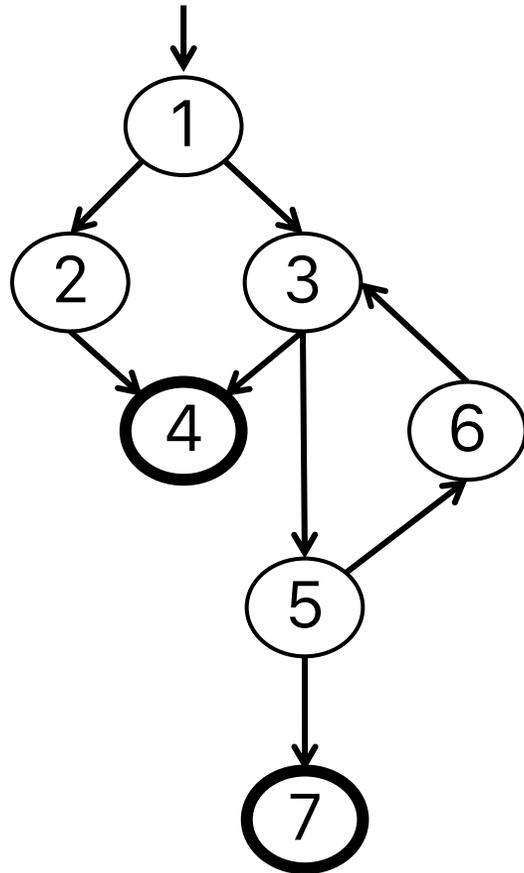
Deterministic software-test always executes the same test path



Non-deterministic software-the same test can execute different test paths

In-class Exercise

Test paths



Answer the following questions for the graph on the left

1. Identify the cycle in the graph.
2. Write all test paths that go through the cycle no more than once.
3. Write one path in the graph that is not a test path.
4. Write one test path in the graph.
5. How many test paths are in the graph?



Reaching Graph Locations

A location (node or edge) in a graph 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 from the first location to the second

This is based only on the graph structure

Semantic reach: a test exists that can execute that subpath

This considers the actual implementation logic

Covering Graphs (7.2)

We use graphs in testing to:

- Develop a model of the software (as a graph)

- Require tests to visit or tour nodes, edges, or subpaths

Test requirements (TRs): Describe the properties of test paths.

Test criteria: Rules that define the test requirements.

Satisfaction: Given a set TR of test requirements for a criterion C, a set of tests T satisfies C on a graph if and only if for every test requirement in TR, there is a test path in $\text{path}(T)$ that meets the test requirement tr.

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

Node Coverage

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

DEFINITION

Node Coverage (NC) – test set T satisfies node coverage on graph G if and only if 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.

DEFINITION

Node Coverage (NC) – TR contains each reachable node in G .

Edge Coverage

Edge coverage is slightly stronger than NC

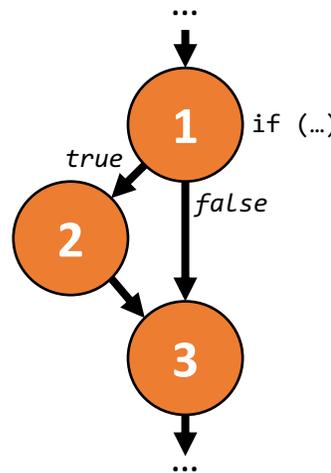
DEFINITION

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

“length up to 1” allows for graphs with one node and no edges

EC TRs differ from NC TRs only when there is a path with length > 1 and a path with length $= 1$ between two nodes

Example: if-then statement

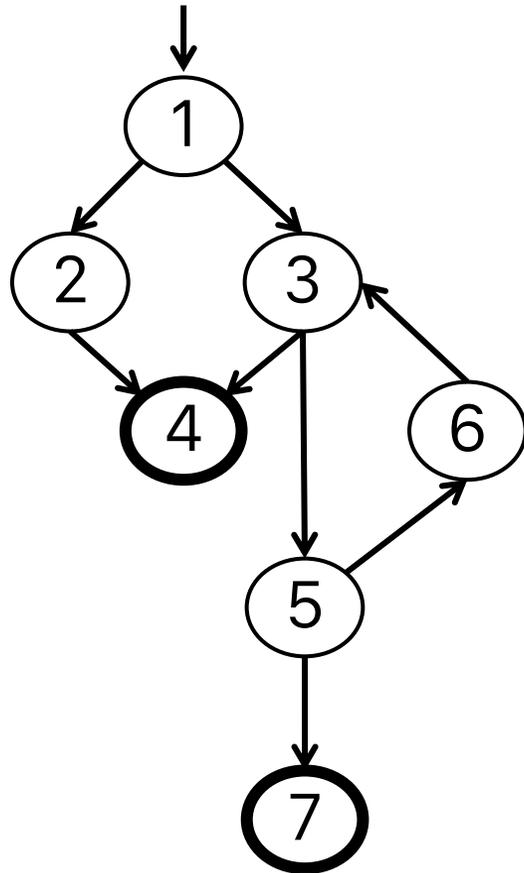


Node Coverage : ? $TR = \{ 1, 2, 3 \}$
Test Path = [1, 2, 3]

Edge Coverage : ? $TR = \{ (1, 2), (1, 3), (2, 3) \}$
Test Paths = [1, 2, 3]
[1, 3]

In-class Exercise

Graph Criteria EC

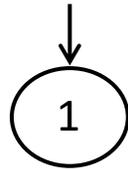


Answer the following questions for the graph on the left

1. List test paths that satisfy edge coverage.
2. Write the set of test requirements for edge-pair coverage.
3. List test paths that satisfy edge-pair coverage.
4. Write the set of test requirements for prime path coverage.
5. List test paths that satisfy prime path coverage.

Node and Edge Coverage

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

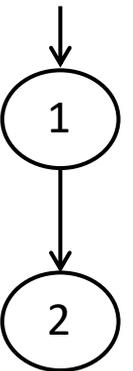


It may seem trivial, but formally, Edge Coverage needs to require Node Coverage on this graph

Otherwise, 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 have only **one edge** – for Edge-Pair Coverage...



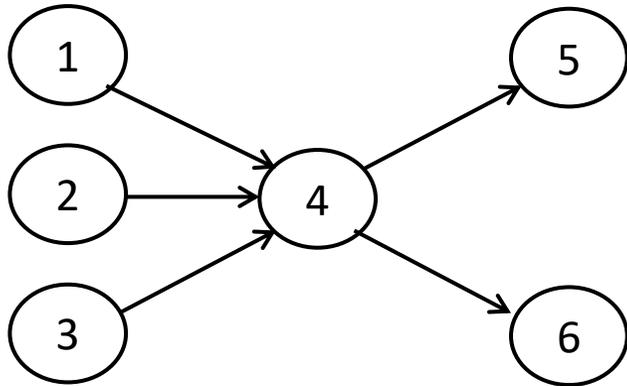
Covering Multiple Edges

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

DEFINITION

Edge-Pair Coverage (EPC) – 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



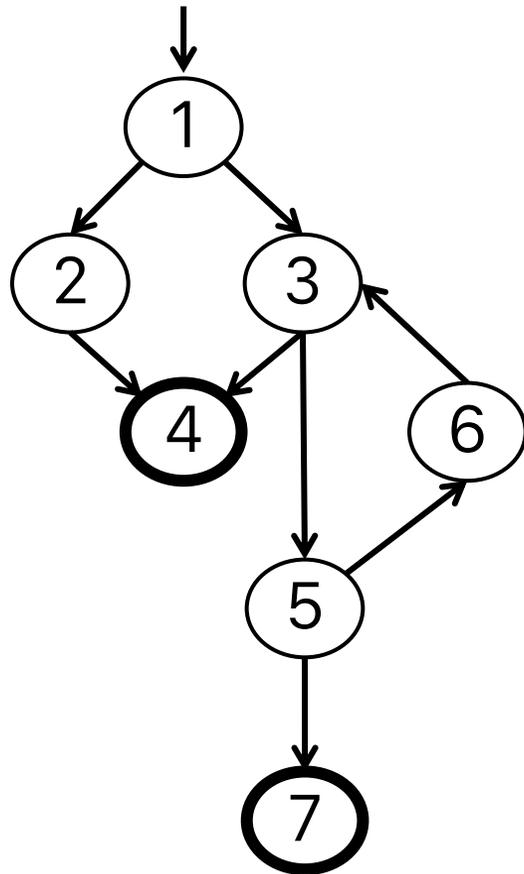
Edge-Pair Coverage : ?

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

The logical extension is to require all paths...

In-class Exercise

Graph Criteria EPC



Answer the following questions for the graph on the left

1. List test paths that satisfy edge coverage.
2. Write the set of test requirements for edge-pair coverage.
3. List test paths that satisfy edge-pair coverage.
4. Write the set of test requirements for prime path coverage.
5. List test paths that satisfy prime path coverage.

Covering Multiple Edges

DEFINITION

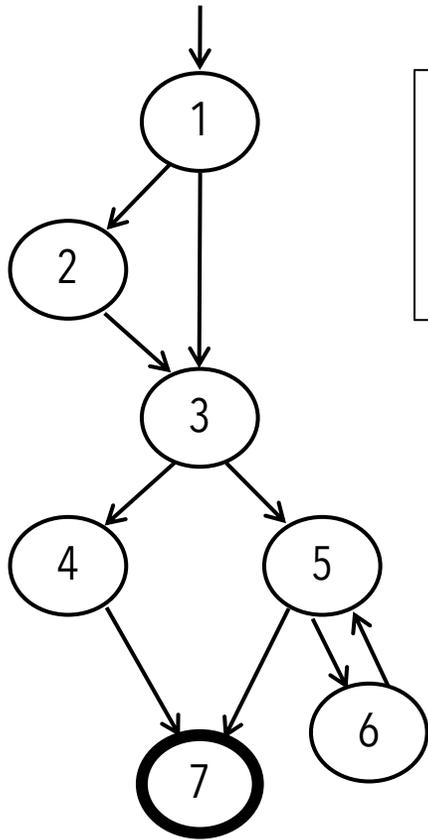
Complete Path Coverage (CPC) – TR contains *all paths* in G .

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

DEFINITION

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

Covering Multiple Edges



Write down the TRs and Test Paths for these criteria

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] ...

Handling Loops in Graphs

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

Thus CPC is **not feasible**.

SPC is not satisfactory because the results are **subjective** and vary with the tester.

Attempts to "deal with" **loops**:

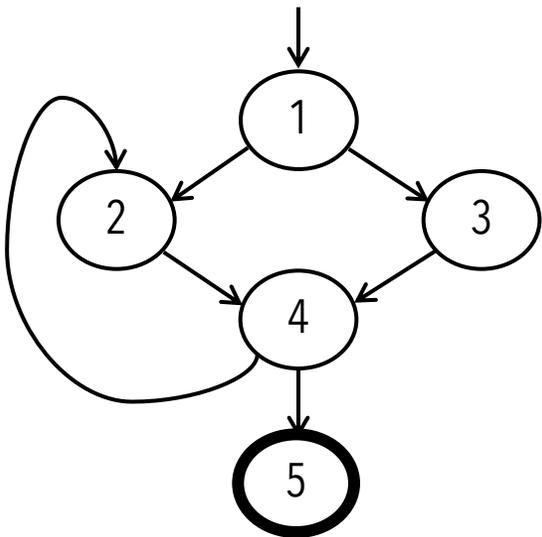
- 1970s**: Execute cycles once
- 1980s**: Execute each loop, exactly once
- 1990s**: Execute loops 0 times, once, more than once
- 2000s**: Prime paths (touring, sidetrips, detours)

Simple Paths & Prime Paths

Simple path: A path from node n_i to n_j is simple if no node appears more than once, except possible the first and last nodes are the same

- No internal loops
- A loop is a simple path

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



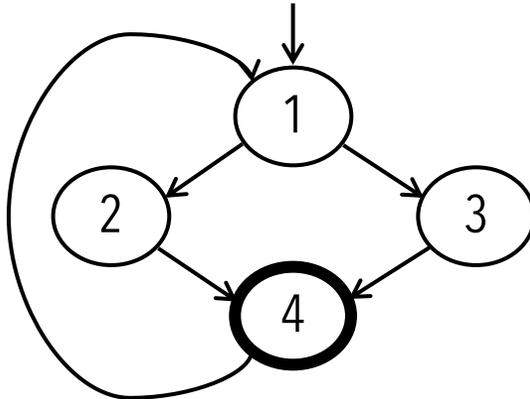
Write down the simple and prime paths for this graph

Simple Paths : [1,2,4,5], [1,3,4,2], [1,3,4,5], [1,2,4], [1,3,4], [2,4,2], [2,4,5], [3,4,2], [3,4,5], [4,2,4], [1,2], [1,3], [2,4], [3,4], [4,2], [4,5], [1], [2], [3], [4], [5]

Prime Paths : [1,2,4,5], [1,3,4,2], [1,3,4,5], [2,4,2], [4,2,4]

Simple Paths & Prime Paths

What if we change the graph?



Write down the simple and prime paths for this graph

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 : [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]

Prime Path Coverage

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

DEFINITION

Prime Path Coverage (PPC) – TR contains *each prime path* in G .

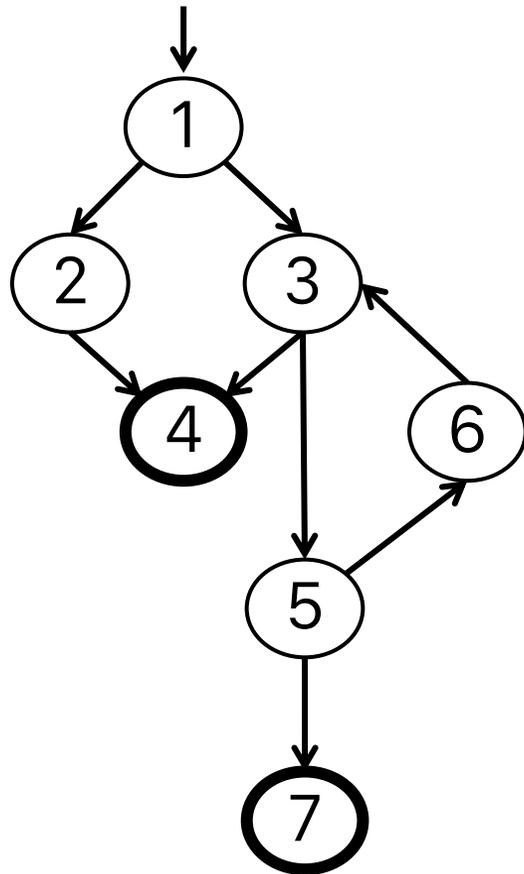
Will tour all paths of length $0, 1, \dots$

That is, it **subsumes** node and edge coverage

PPC almost, but **not quite**, subsumes **EPC**...

In-class Exercise

Graph Criteria PPC



Answer the following questions for the graph on the left

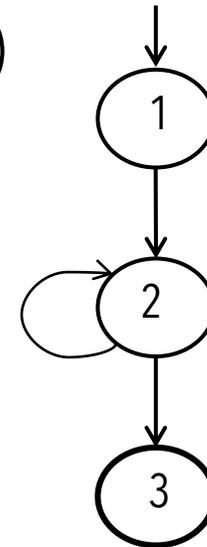
1. List test paths that satisfy edge coverage.
2. Write the set of test requirements for edge-pair coverage.
3. List test paths that satisfy edge-pair coverage.
4. Write the set of test requirements for prime path coverage.
5. List test paths that satisfy prime path coverage.

PPC does not subsume EPC

If a node has an edge to itself (*self edge*), **EPC** requires $[n, n, m]$ and $[m, n, n]$

$[n, n, m]$ is not prime

Neither $[n, n, m]$ nor $[m, n, n]$ are simple paths (not prime)



EPC Requirements : ?

TR = $\{ [1,2,3], [1,2,2], [2,2,3], [2,2,2] \}$

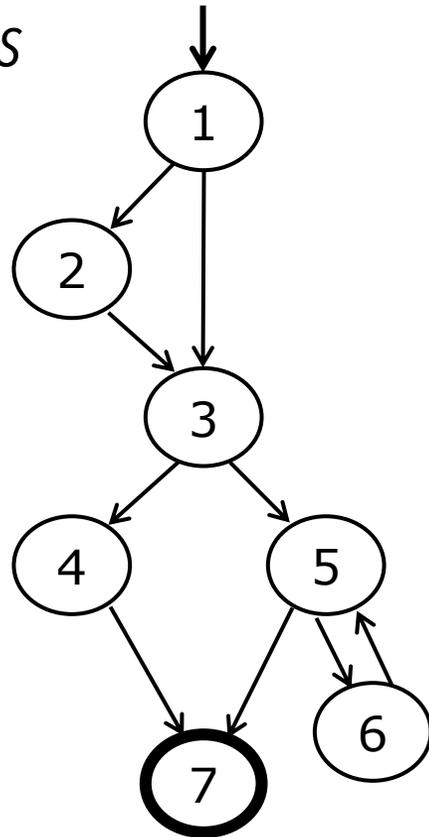
PPC Requirements : ?

TR = $\{ [1,2,3], [2,2] \}$

Prime Path Example

The previous example has 38 **simple** paths

Only **nine** prime paths



Write down all 9
prime paths

Prime Paths

[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

Touring, Sidetrips, and Detours

Prime paths do not have **internal loops** ... test paths might

Tour: *A test path p tours subpath q if q is a subpath of p*

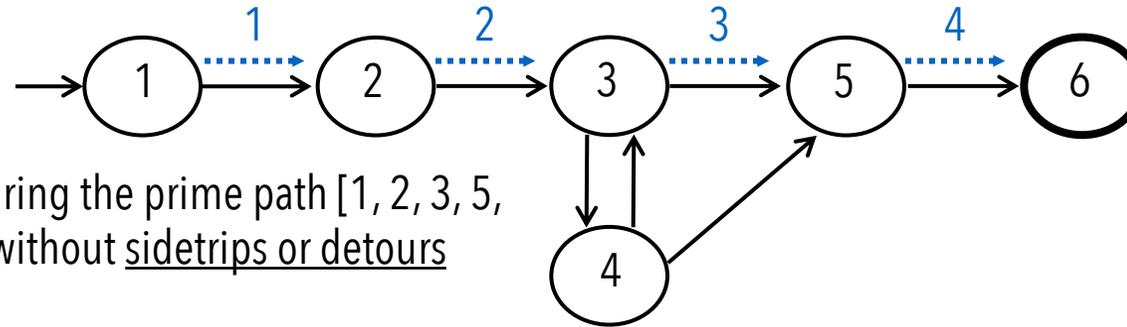
Tour with sidetrips: *A test path p tours subpath q with sidetrips iff every edge in q is also in p in the same order*

-The tour can include a sidetrip, as long as it comes back to the same node

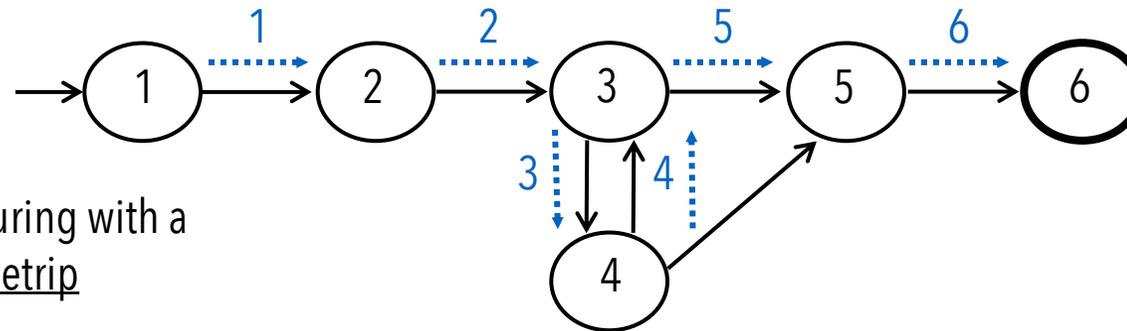
Tour with detours: *A test path p tours subpath q with detours iff every node in q is also in p in the same order*

-The tour can include a detour from node ni , as long as it comes back to the prime path at a successor of ni

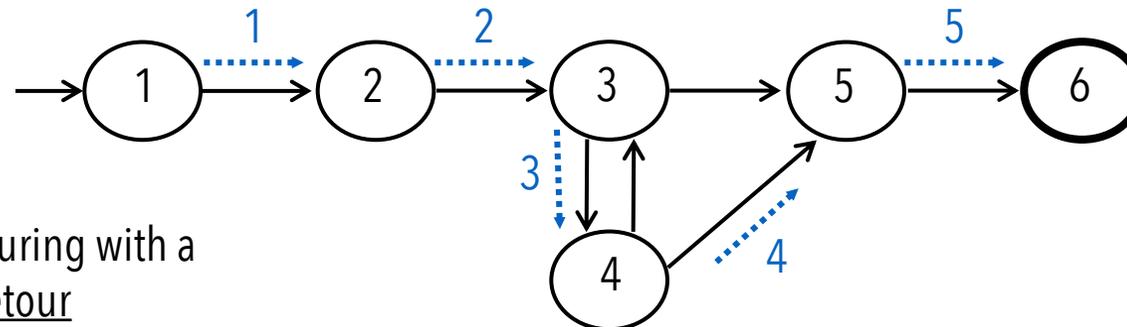
Sidetrips and Detours Example



Touring the prime path [1, 2, 3, 5, 6] without sidetrips or detours



Touring with a sidetrip



Touring with a detour

Infeasible Test Requirements

An **infeasible** test requirement cannot be satisfied.

- Unreachable statement (dead code)
- Subpath that can only be executed with a contradiction ($x > 0$ and $x < 0$)

Most test **criteria** have some infeasible test requirements

It is usually **undecidable** whether all test requirements are feasible

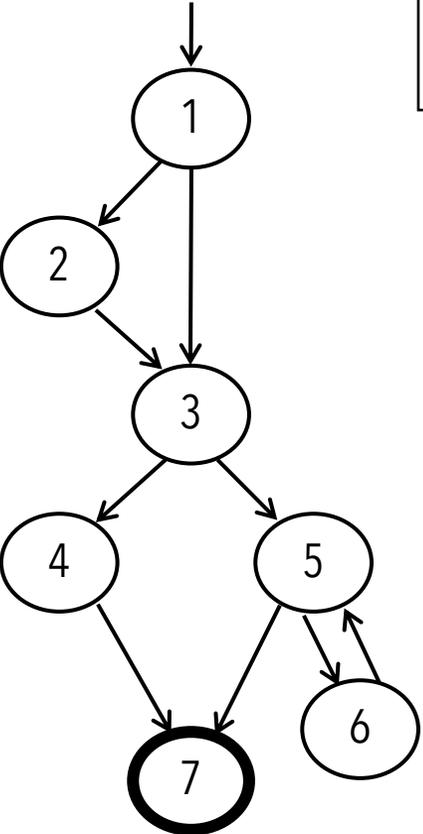
When sidetrips are not allowed, many structural criteria have **more infeasible test requirements**

However, always allowing **sidetrips weakens** the test criteria

Practical recommendation–Best Effort Touring

- Satisfy as many test requirements as possible without sidetrips
- Allow sidetrips to try to satisfy remaining test requirements

Simple & Prime Path Example



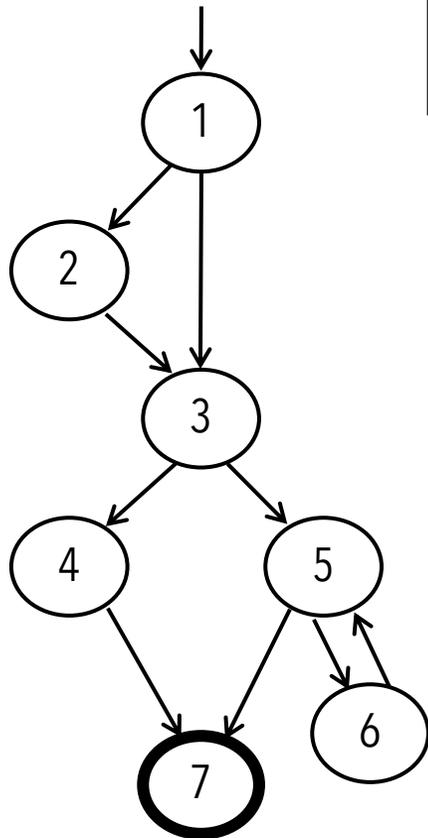
Simple paths

Write paths of length 0

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

'!' means path terminates

Simple & Prime Path Example



Simple paths

Write paths of length 1

Len 1

[1, 2]

[1, 3]

[2, 3]

[3, 4]

[3, 5]

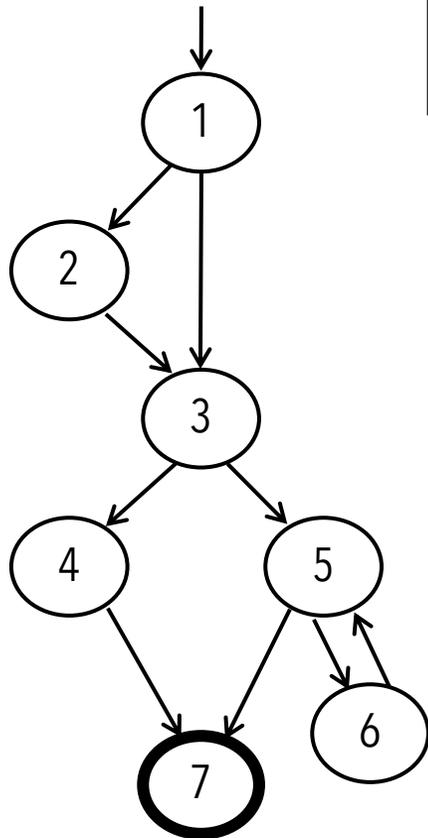
[4, 7]!

[5, 7]!

[5, 6]

[6, 5]

Simple & Prime Path Example



Simple paths

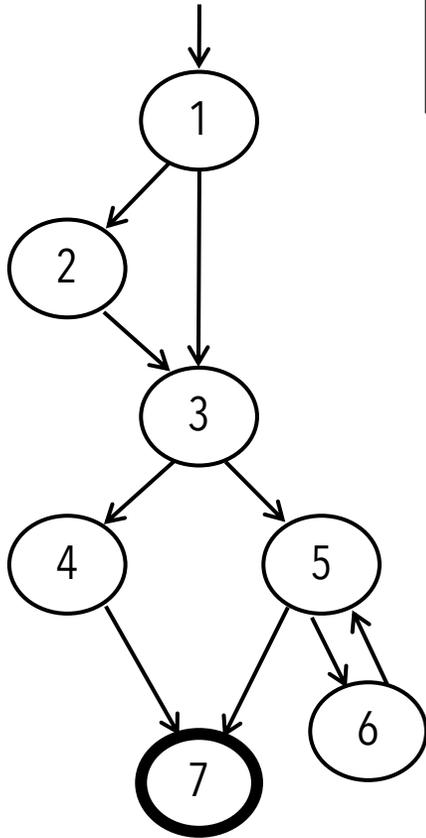
Write paths of length 2

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]*

'*' means path cycles

Simple & Prime Path Example

Simple paths

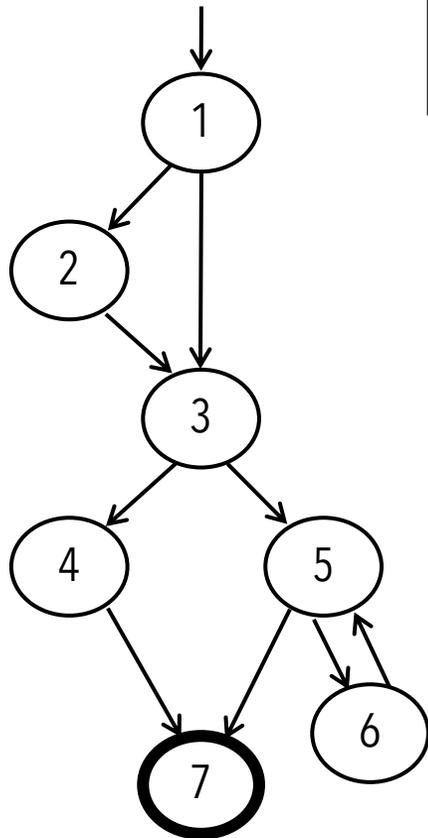


Write paths of length 3

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]!

Simple & Prime Path Example



Simple paths

Write paths of length 4

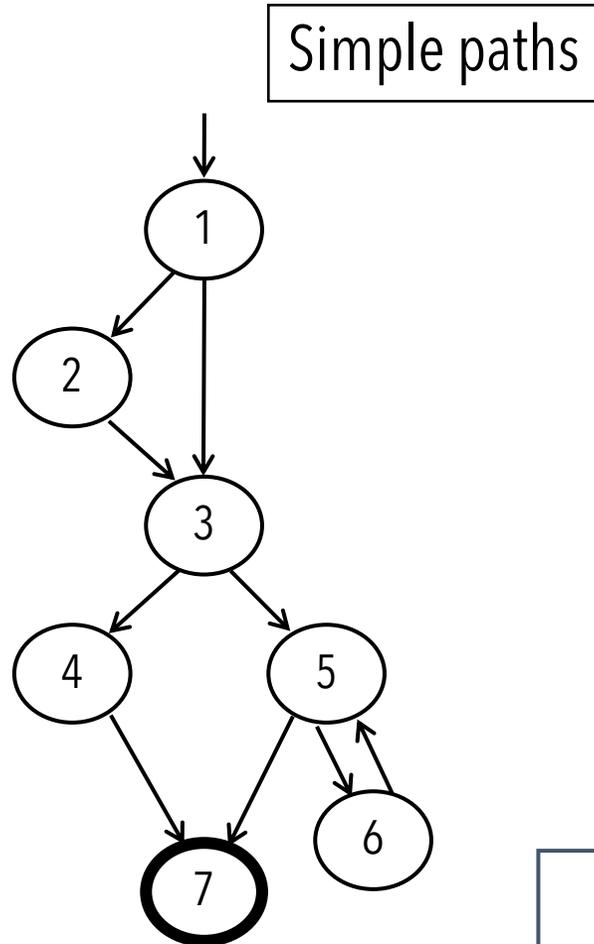
Len 4

[1, 2, 3, 4, 7]!

[1, 2, 3, 5, 7]!

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

Simple & Prime Path Example



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 ?

Round Trips

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

DEFINITION

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

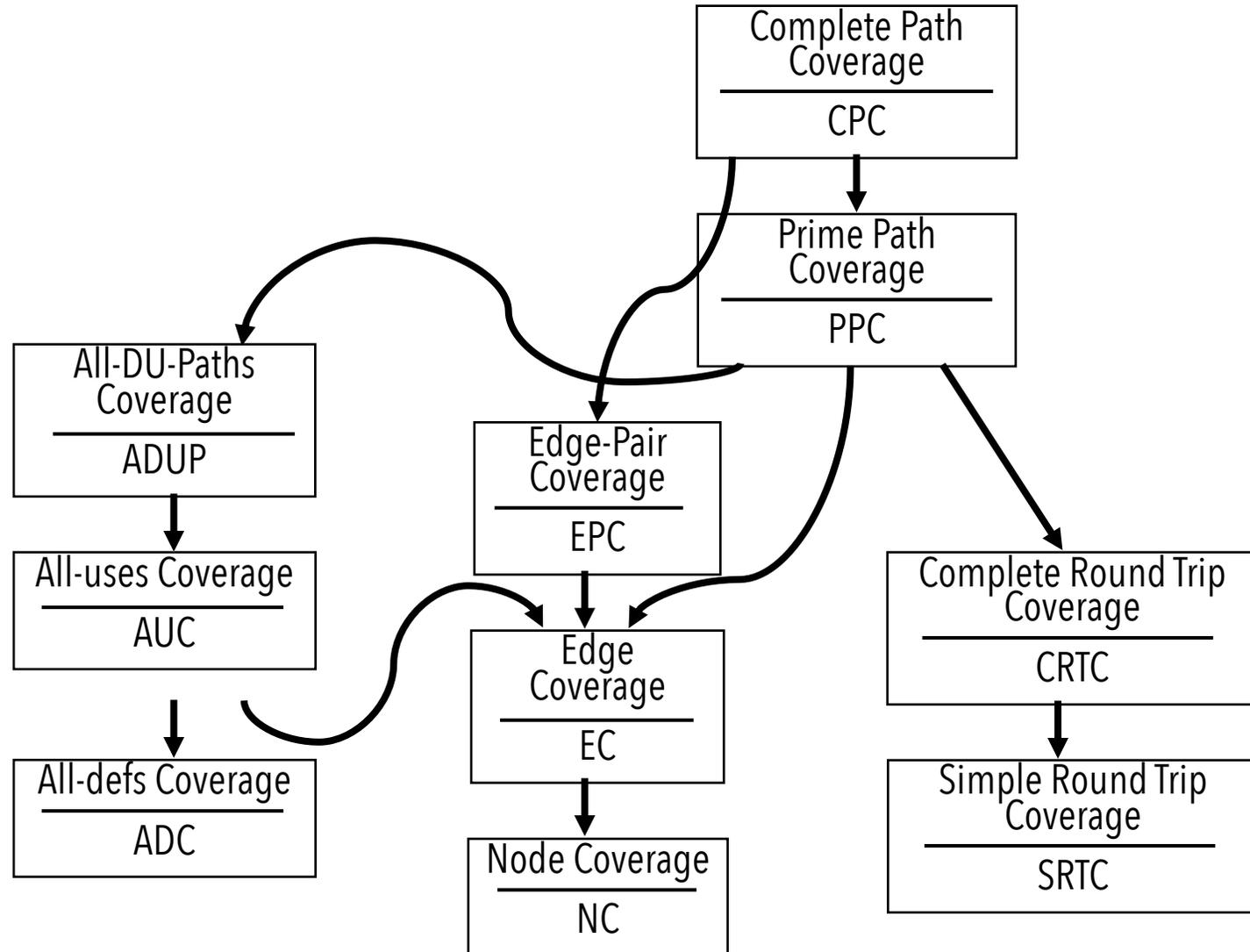
DEFINITION

Complete Round Trip Coverage (CRTC) – TR contains *all* round-trip paths for each reachable node in G .

These criteria **omit nodes and edges** that are not in round trips

Thus they do **not subsume** edge-pair, edge, or node coverage

Graph Coverage Criteria Subsumption



Graph Coverage Summary (7.1-7.2)

Graphs are a very **powerful abstraction** for designing tests

The various criteria allow lots of **cost/benefit** tradeoffs

These two sections are entirely at the "**design abstraction level**" from chapter 2

Graphs appear in **many situations** in software

-As discussed in the rest of chapter 7