# CS 5154: Software Testing

# Graph Coverage

Instructor: Owolabi Legunsen

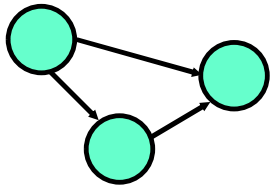
Fall 2021

# Recall the four software models in this course

**Input Domains**

A: {0, 1, >1}
B: {600, 700, 800}
C: {cs, ece, is, sds}

**Graphs**



**Logic Expressions**

(!x | !y) & a & b

**Syntax**

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

# Why learn about graph coverage?

- Some of the most widely-used coverage criteria
- The “R” in the RIPR model
- Graph coverage criteria help create tests that reach different parts of code

# Roadmap on Graph-based MDTD

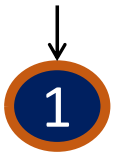
- Today: establish a vocabulary for talking about graph coverage
- Next: apply graph coverage to source code

# What is a graph?

Graph G is a tuple ($N$, $N_0$, $N_f$, $E$), where

- N is a non-empty set of nodes
- $N_0 \subset N$ is a non-empty set of initial nodes
- $N_f \subset N$ is a non-empty set of final nodes
- E is a set of pairs ($n_i$, $n_j$) where an edge exists from node $n_i$ to node $n_j$ in G
  - $n_i$ : predecessor, $n_j$ : successor

# Based on the definition, is this a graph?



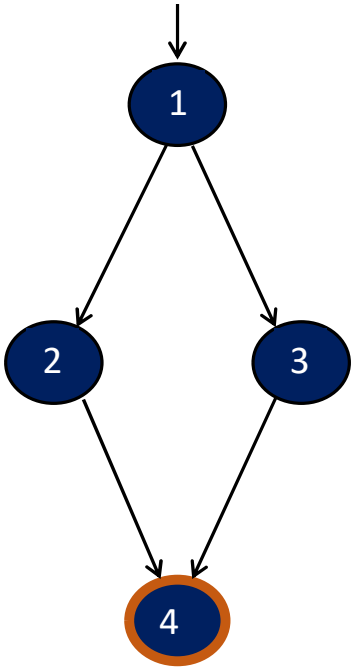

$N = \{ 1 \}$

$N_0 = \{ 1 \}$

$N_f = \{ 1 \}$

$E = \{ \}$

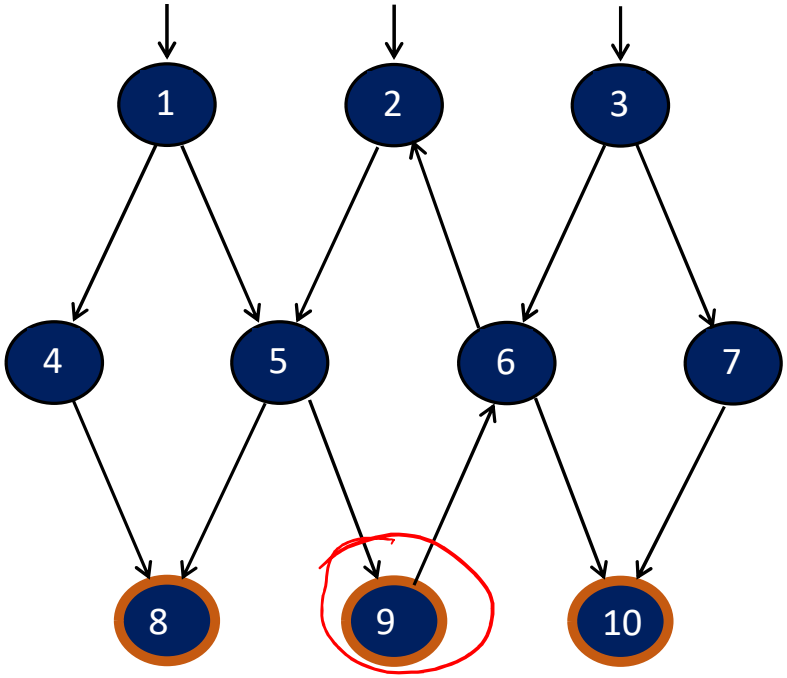
# Examples of graphs




$N_0 = \{ 1 \}$

$N_f = \{ 4 \}$

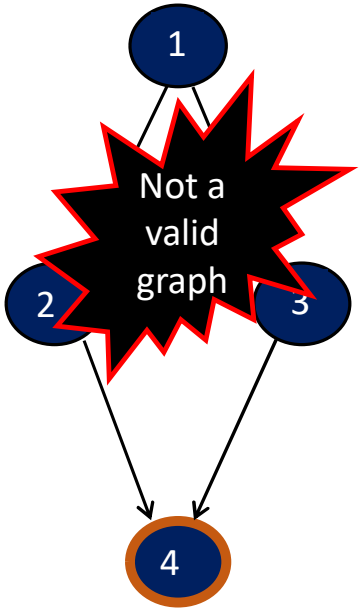
$E = \{ (1, 2), (1,3), (2,4), (3,4) \}$




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

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

$E = \{ (1,4), (1,5), (2,5), (3,6), (3, 7), (4, 8), (5,8), (5,9), (6,2), (6,10), (7,10) (9,6) \}$

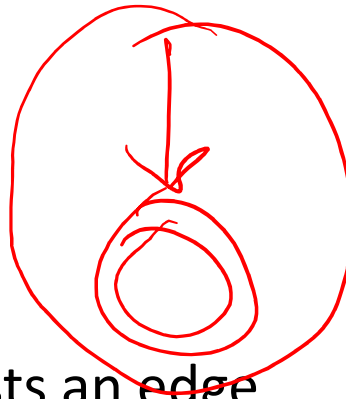



$N_0 = \{ \}$

$N_f = \{ 4 \}$

$E = \{ (1,2), (1,3), (2,4), (3,4) \}$

# Graph coverage criteria are usually about paths in G



- Path : A sequence, *p*, of nodes [$n_1$, $n_2$, …, $n_M$] such that there exists an edge between each pair of nodes in *p*

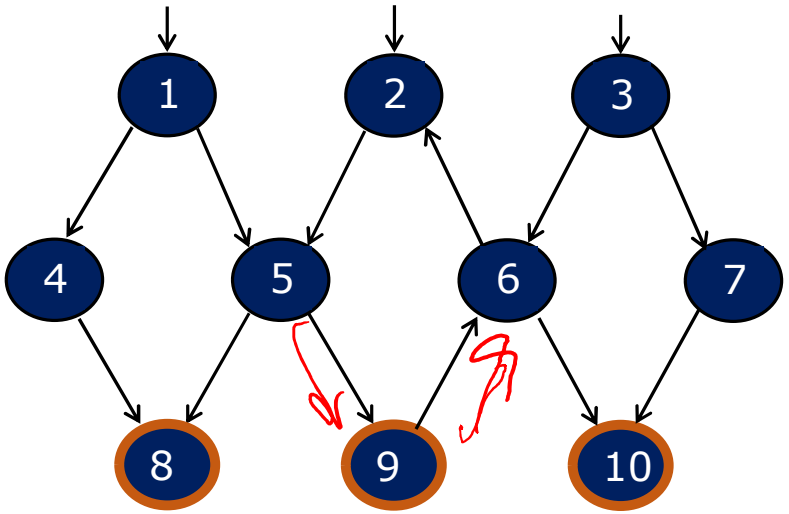
- Length of a path : The number of edges in *p*
  - A single node is a path of length 0

- Subpath : A subsequence of nodes in *p* is a subpath of *p*

# Graph coverage criteria are usually about paths in G

- Path : A sequence, *p*, of nodes $[n_1, n_2, \ldots, n_M]$ such that there exists an edge between each pair of adjacent nodes $(n_i, n_{i+1})$, $1 \leq i < M$
- Length of a path : The number of edges in *p*
  - A single node is a path of length 0
- Subpath : A subsequence of nodes in *p* is a subpath of *p*

# Identify some paths in this graph




**A Few Paths**

[ 1, 4, 8 ]

[ 2, 5, 9, 6, 2 ]

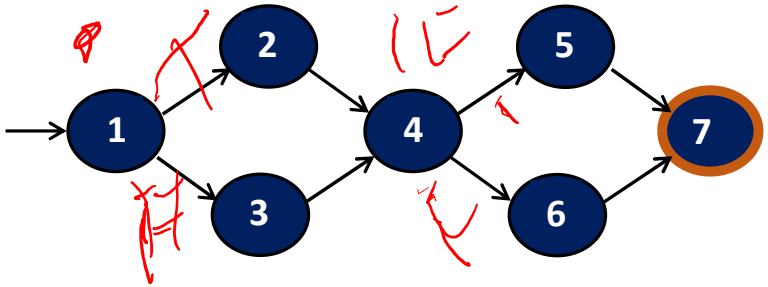
[ 3, 7, 10 ]

# Tests must start at $n_i \in N_0$ and end at $n_j \in N_f$

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

- Single entry, single exit (SESE) graphs :
  - $N_0$ and $N_f$ have exactly one node
  - All test paths start at $n \in N_0$ and end at $m \in N_f$

# Identify all the test paths in this graph




**Four test paths**

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

# What does it mean for tests to “cover” graphs?

- 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)

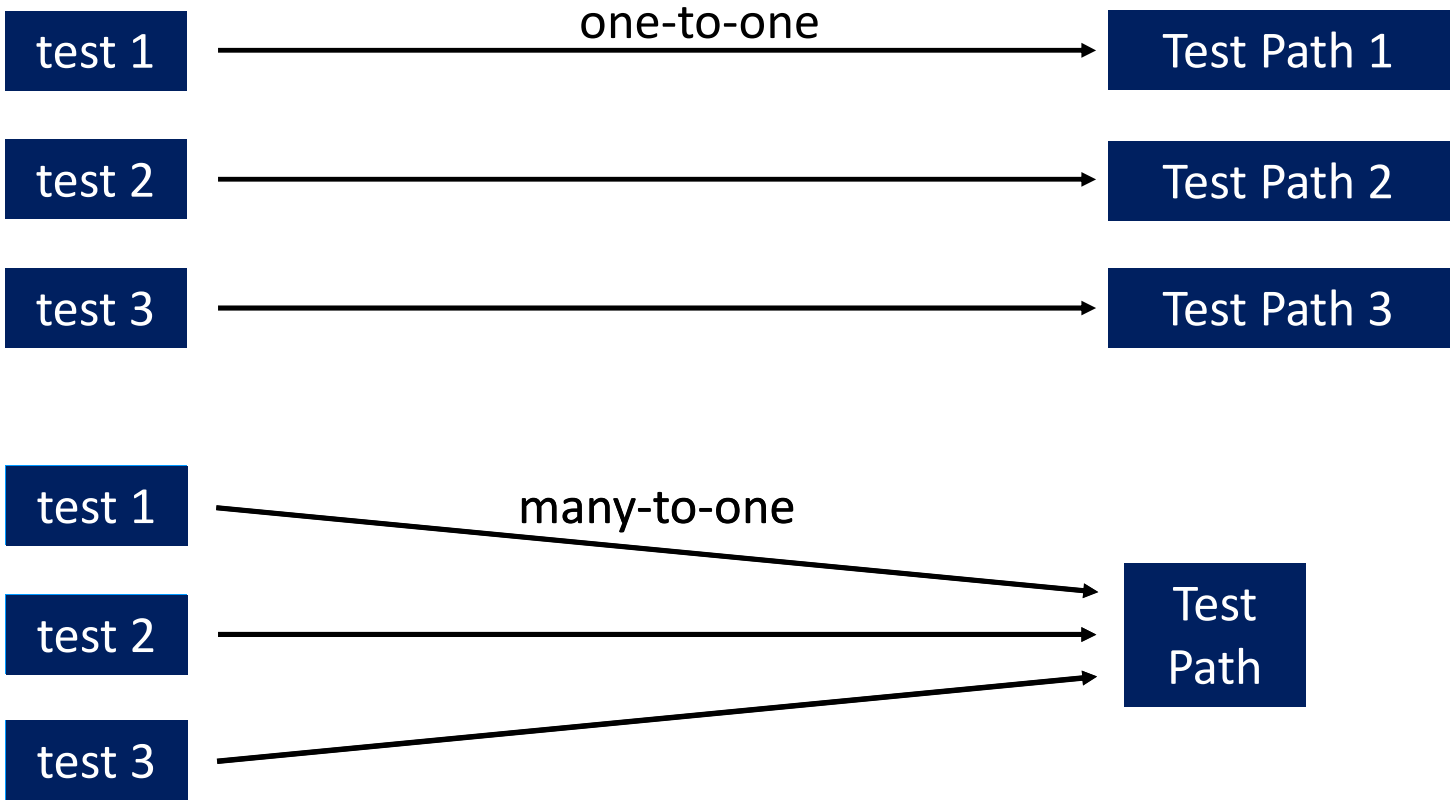
Tours subpaths ? [1,2,4], [2,4,5], [4,5,7], [1,2,4,5], [2,4,5,7], [1,2,4,5,7]

(*Also, each edge is technically a subpath*)

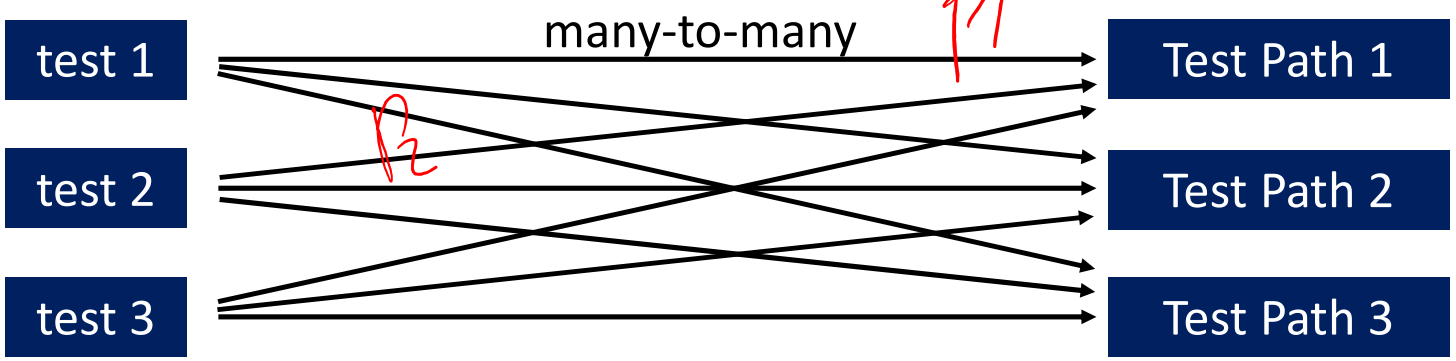
# Terminology for discussing test cases and test paths

- path (*t*) : The test path executed by test case *t*
- path (*T*) : The set of test paths executed by set of test cases *T*
- Each test case executes one and only one test path
  - Is the previous statement really true?

# Relationship among test cases and test paths



# Relationship among test cases and test paths



# Terminology for discussing test cases and test paths

- path (*t*) : The test path executed by test case *t*
- path (*T*) : The set of test paths executed by set of test cases *T*
- ~~Each test case executes one and only one test path~~
  - Is the previous statement really true? No
- Each test case executes one and only one test path at once

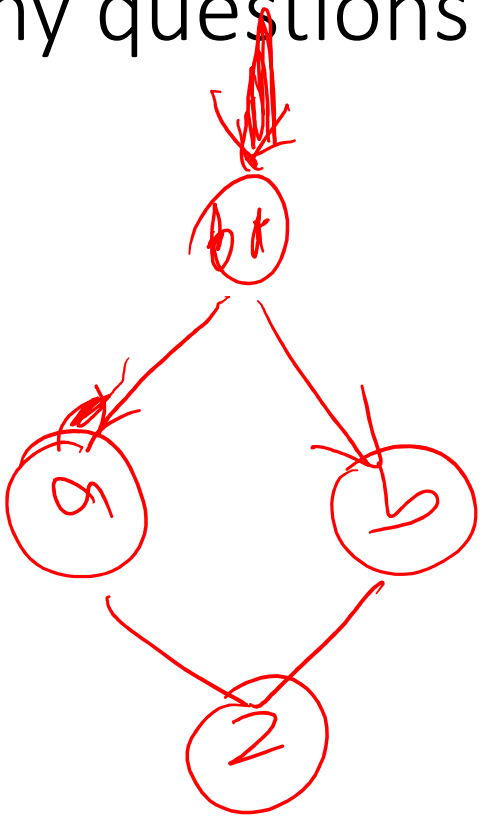
# More terminology on test cases and test paths

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

1. *Syntactic reach* : A subpath exists in the graph
2. *Semantic reach* : A test exists that can execute the subpath in (1)
3. Semantic vs syntactic reach is important when applied to source code

HW0: You all computed syntactic reachability on a given graph!

# Any questions so far




?

# Implementing Graph-based MDTD

- Develop a model of the software as a graph
- Require tests to visit/tour sets of nodes, edges, or sub-paths
- Choose inputs that satisfy the test requirements
- Implement and automate tests based on the inputs chosen

# Recall these three general concepts?

- **Test Requirement** : A software element that a test must satisfy or cover

- **Coverage Criterion** : A rule or collection of rules that impose test requirements on a set of tests

- **Coverage** : Given a set of test requirements *TR* for coverage criterion *C*, a test set *T* satisfies *C* if and only if for every test requirement *tr* in *TR*, there is at least one test *t* in *T* such that *t* satisfies *tr*

# Defining these three concepts on Graphs

- Test Requirements (TR) : Describe properties of test paths

- Coverage Criterion : Rules that define test requirements.
  - We discuss some of those next

- Coverage : 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 *tr* in *TR*, there is a test path in *path(T)* that meets the test requirement *tr*

# Two kinds of graph coverage criteria

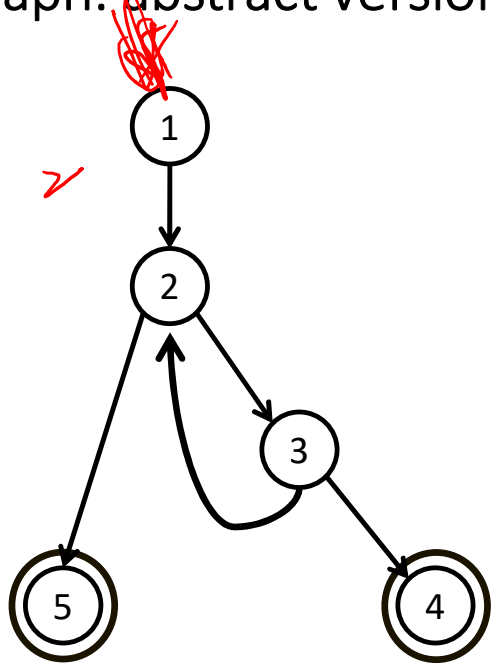
1. Structural Coverage Criteria : Defined on a graph just in terms of nodes and edges

2. Data Flow Coverage Criteria : Defined on a graph that is annotated with variable definitions and uses (i.e., *def-use* pairs)
   a. Do tests cover every use of each variable definition?
   b. Are there variable definitions that are not covered by any test?

We will not cover Data Flow Coverage Criteria this semester

# Recall: we saw structural coverage criteria before

Graph: abstract version




Edges
1 2
2 3
3 2
3 4
2 5
Initial Node: 1
Final Nodes: 4, 5

6 requirements for Edge-Pair Coverage
1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

Test Paths
[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]

convention : ([1,2],[2,3]) = [1,2,3]

# Structural Coverage Criteria

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

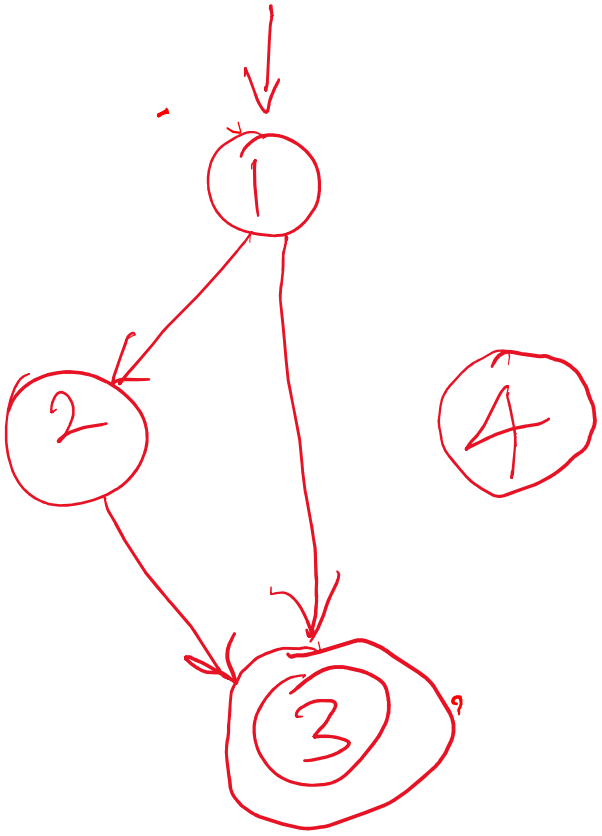
# Node Coverage

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

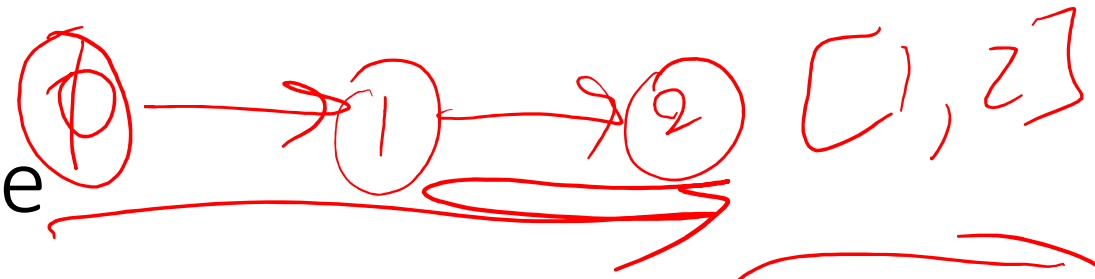
Node Coverage (NC) : TR contains each reachable node in G.

# Example on Node Coverage




How many TRs are there?

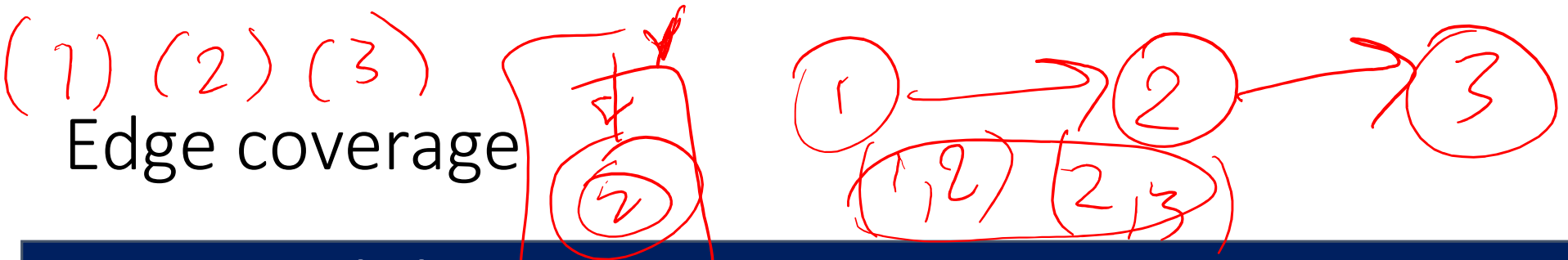
# Edge coverage




**Edge Coverage (EC) : TR contains each reachable path of length 2 in G.**

- What is wrong with this definition?

# Edge coverage




**Edge Coverage (EC) : TR contains each reachable path of length 1 in G.**

- In theory, should Edge Coverage subsume Node Coverage?
- Given the definition above, does Edge Coverage subsume Node Coverage?
- What is wrong with this definition of Edge Coverage?

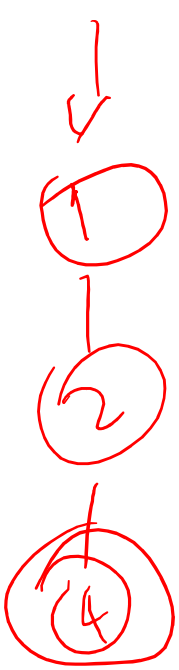
# Edge coverage

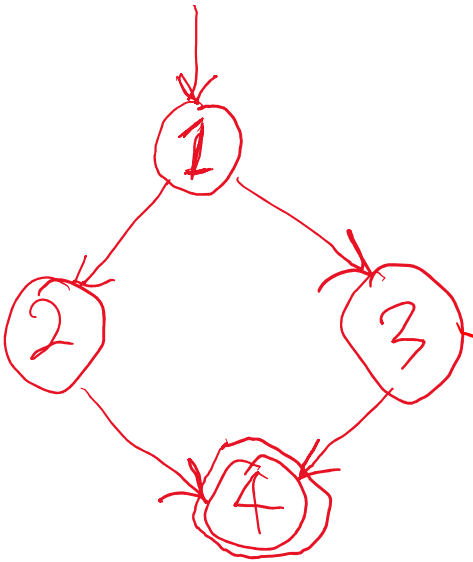
**Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.**

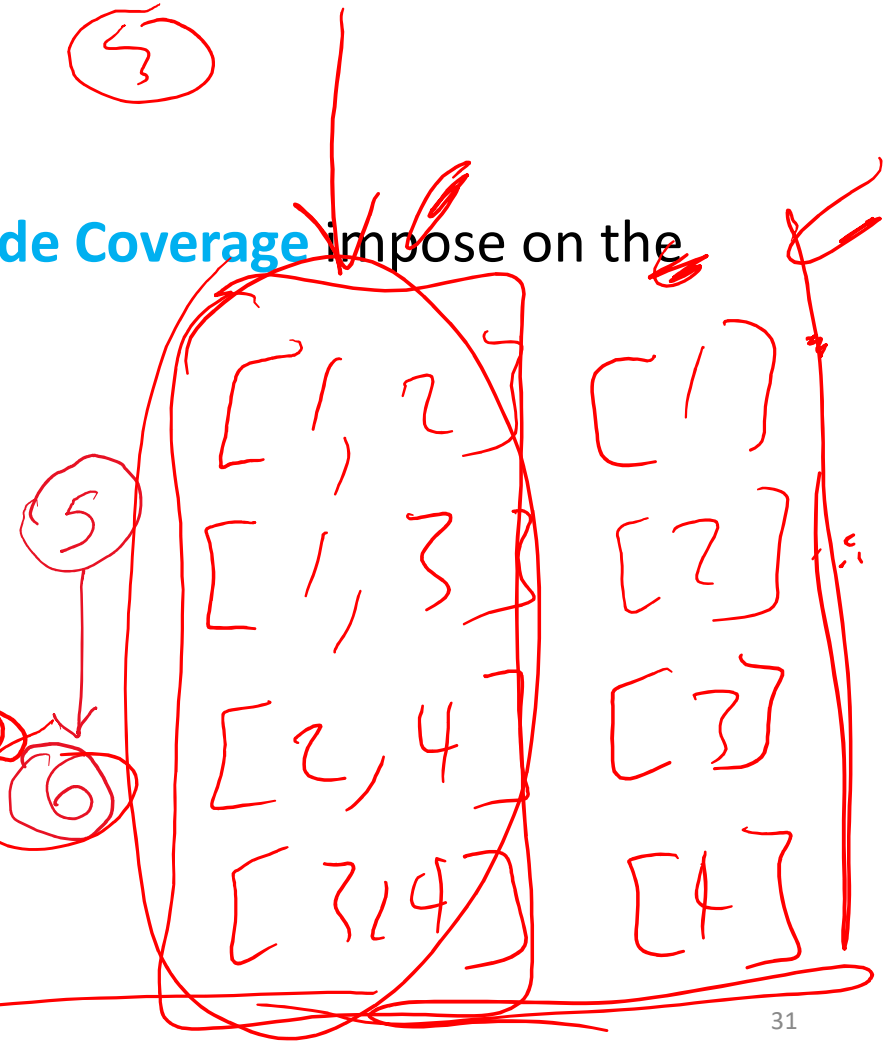
- The phrase "*length up to 1*" allows for graphs with one node and no edges

# Node Coverage review

- How many test requirements does **Node Coverage** impose on the tests for this graph:

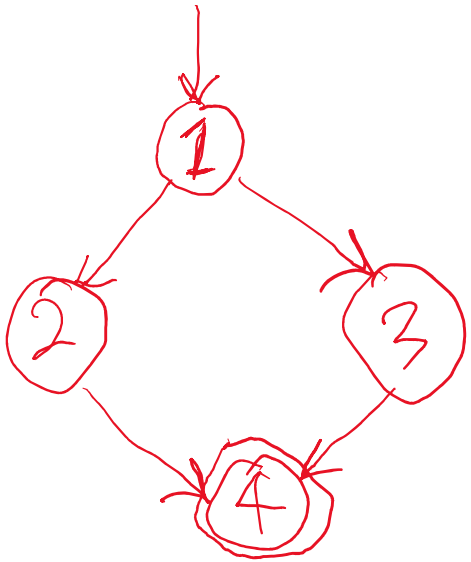


# Edge Coverage review

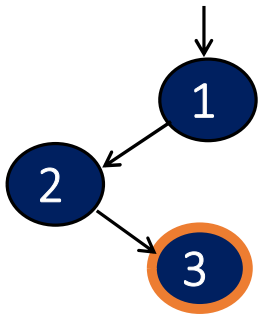
- How many test requirements does **Edge Coverage** impose on the tests for this graph:





[1,3,4]

# Examples on Node and Edge Coverage

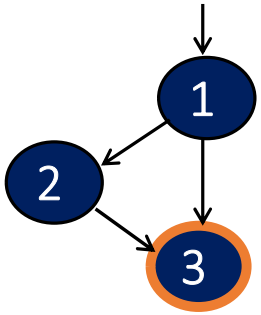



TRs for Node Coverage:

Test Paths for Node Coverage:

TRs for Edge Coverage:

Test Paths for Edge Coverage:



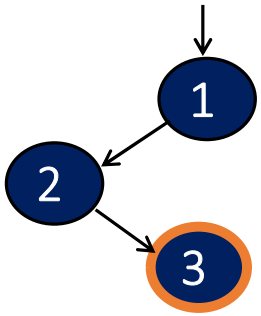

TRs for Node Coverage:

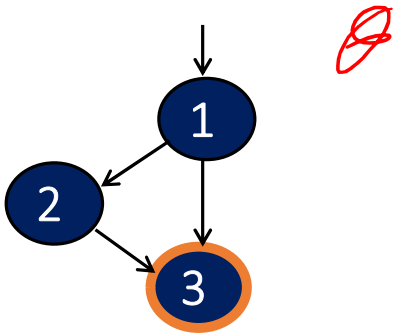
Test Paths for Node Coverage:

TRs for Edge Coverage:

Test Paths for Edge Coverage:

# When do Node Coverage and Edge Coverage differ?




Node Coverage and Edge Coverage are only different when there is more than one path between a pair of nodes (as in an “if-else” statement)

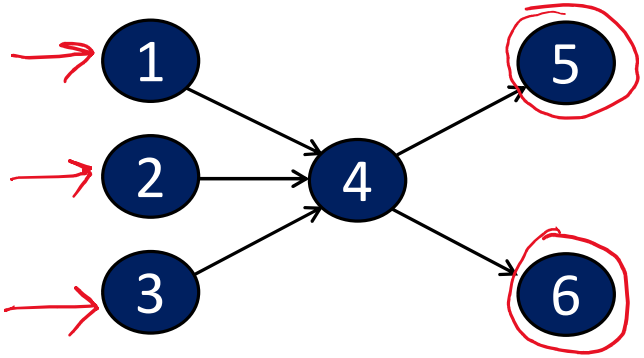
# What if we want tests to cover multiple edges?

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

- Why do we need the phrase, "length up to 2"?

- "length up to 2" captures graphs that have less than 2 edges

# Example on Edge-Pair Coverage




Edge-Pair Coverage : ?

TR = { …}

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

What if we want tests to cover more than two edges?

# Should we play the same game as in ISP?

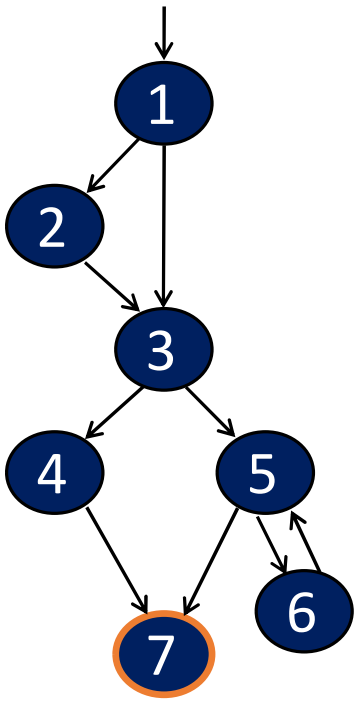
- **Pair-Wise Coverage (PWC) Criterion** : A value from each block for each characteristic must be combined with a value from every block for all other characteristics.

- **t-Wise Coverage (TWC) Criterion** : A value from each block for each group of t characteristics must be combined

# Covering all edges

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

# In-class exercise…




**Node Coverage**

TR =

Test Paths:

**Edge Coverage**

TR =

Test Paths:

*Write down the TRs and Test Paths for these criteria*

**Edge-Pair Coverage**

TR =

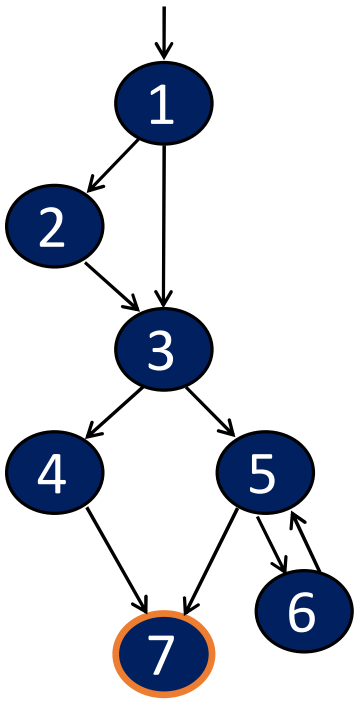
Test Paths:

**Complete Path Coverage**

TR:

Test Paths:

# In-class exercise solution




**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) }

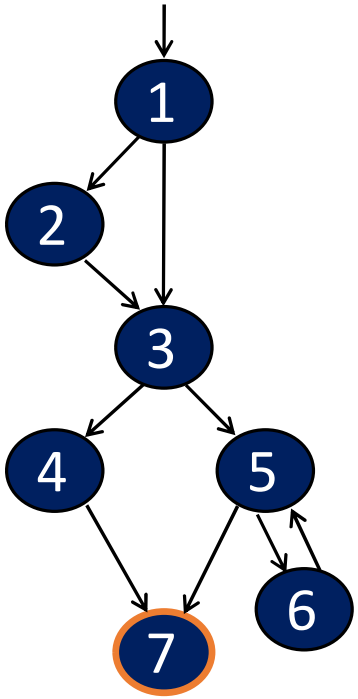
Test Paths: [ 1, 2, 3, 4, 7 ] [1, 3, 5, 6, 5, 7 ]

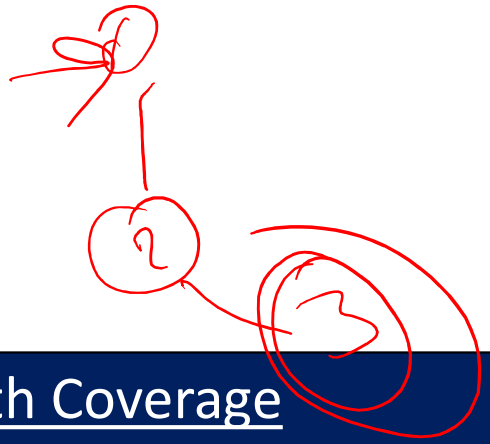
**Edge-Pair Coverage**

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 ]

# In-class exercise solution (2)



**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 ] …

Unfortunately, CPC is **impossible** to satisfy if G has a loop

# What we covered so far

- Basic definition of Graphs
- Terminology that we will use to talk about Graph Coverage
- Your first few Graph Coverage Criteria
- Complete Path Coverage is infeasible!

# Next

- How to handle loops in Graph Coverage