CSc 179 – Graph Coverage Criteria

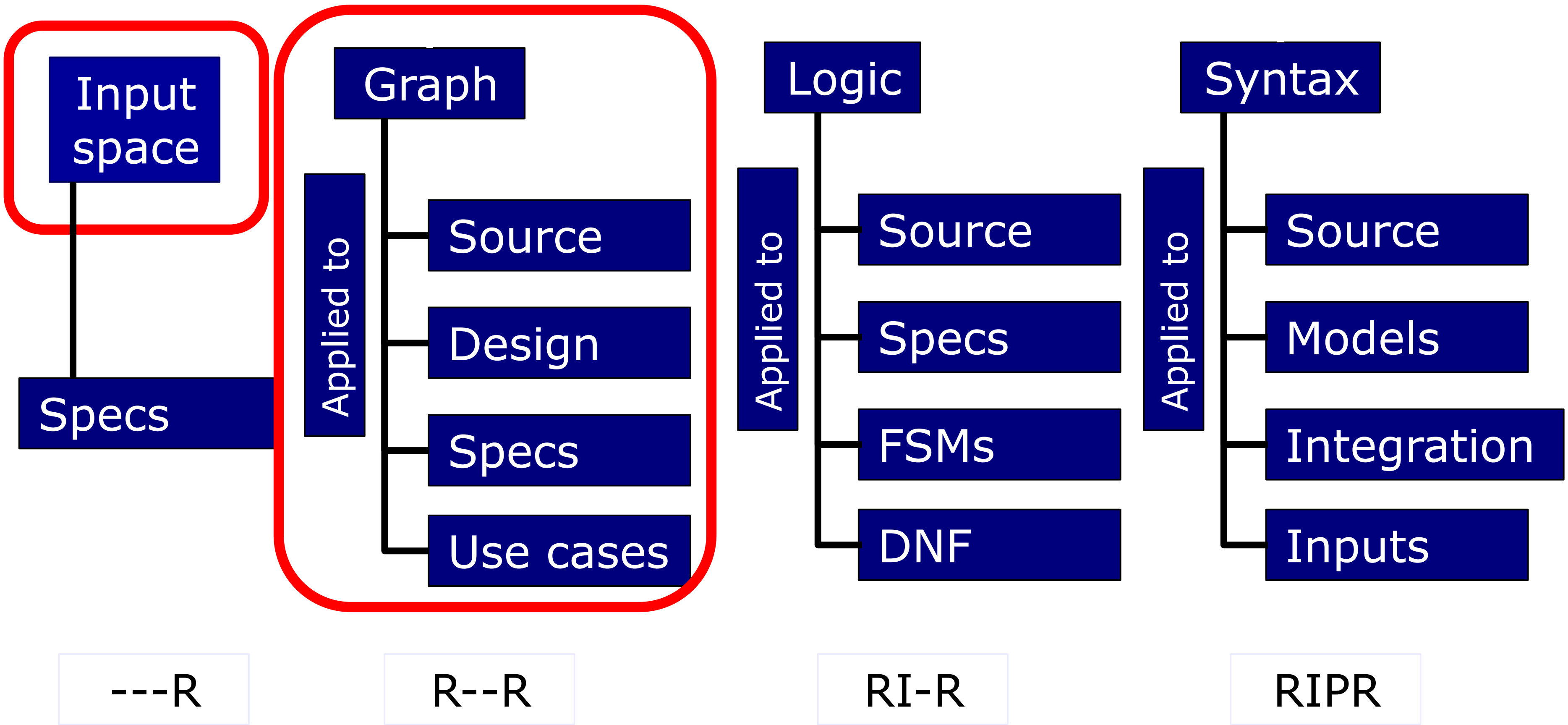
Credits:

AO – Ammann and Offutt, “Introduction to Software Testing,” Ch. 7.1 & 7.2

University of Virginia (CS 4501 / 6501)

Structures for Criteria-Based Testing

Four structures for modeling software



Today’s Objectives

- Investigate some of the most widely known test coverage criteria
- Understand basic theory of graph
  - Generic view of graph without regard to the graph’s source
- Understand how to use graph to define criteria and design tests
  - Node coverage (NC)
  - Edge coverage (EC)
  - Edge-pair coverage (EPC)
- Graph derived from various software artifacts (coming soon)

Overview

- Graphs are the most commonly used structure for testing
- Graphs can come from many sources
  - Control flow graphs from source
  - Design structures
  - Finite state machine (FSM)
  - Statecharts
  - Use cases
- The graph is not the same as the artifact under test, and usually omits certain details
- Tests must cover the graph in some way
  - Usually traversing specific portions of the graph

Graph: Nodes and Edges

- Node represents
  - Statement
  - State
  - Method
  - Basic block

- Edge represents
  - Branch
  - Transition
  - Method call

Basic Notion of a Graph

- Nodes:
  - N = a set of nodes, N must not be empty
- Initial nodes
  - N_0 = a set of initial nodes, must not be empty
  - Single entry vs. multiple entry
- Final nodes
  - N_f = a set of final nodes, must not be empty
  - Single exit vs. multiple exit

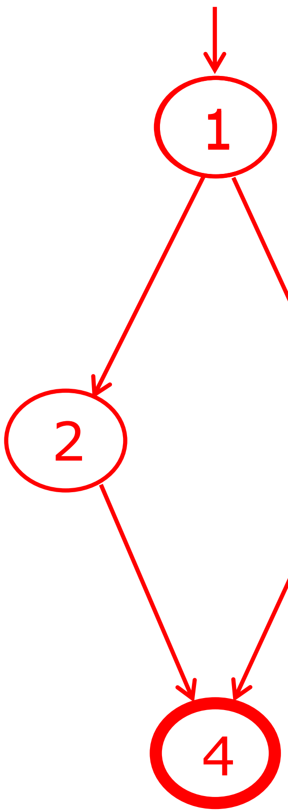
Every test must start in some initial node, and end in some final node

- Edges:
  - E = a set of edges, each edge from one node to another
  - An edge is written as (n_i, n_j)
    - n_i is predecessor, n_j is successor

Note on Graphs

- The concept of a final node depends on the kind of software artifact the graph represents
- Some test criteria require tests to end in a particular final node
- Some test criteria are satisfied with any node for a final node (i.e., the set N_f = the set N)

Example Graph



Single-Entry, Single-Exit (SESE)

- Node

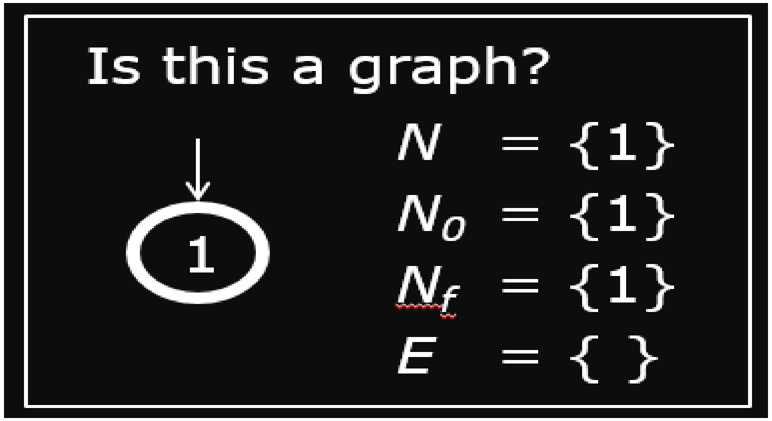
  $N = \{1, 2, 3, 4\}$

  $N_0 = \{1\}$

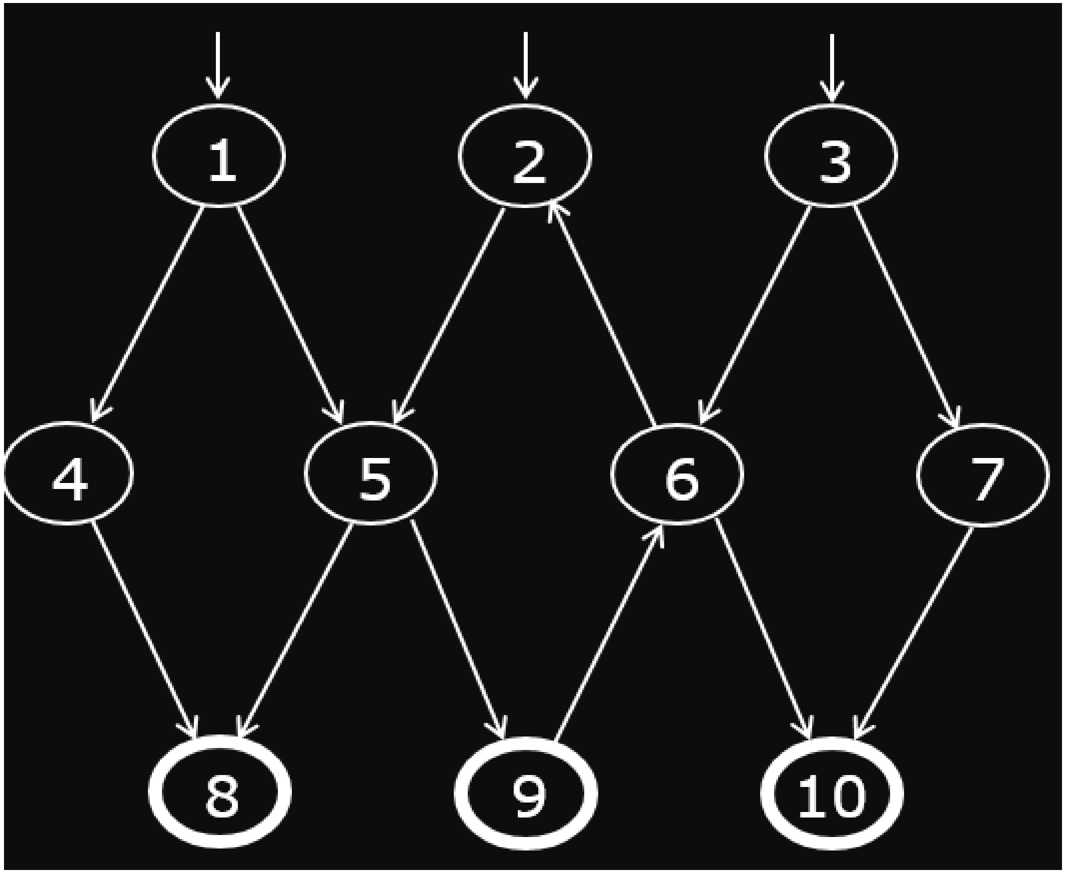
  $N_f = \{4\}$

- Edge

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



Example Graph



Multiple-entry, multiple-exit

- Node

  N = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

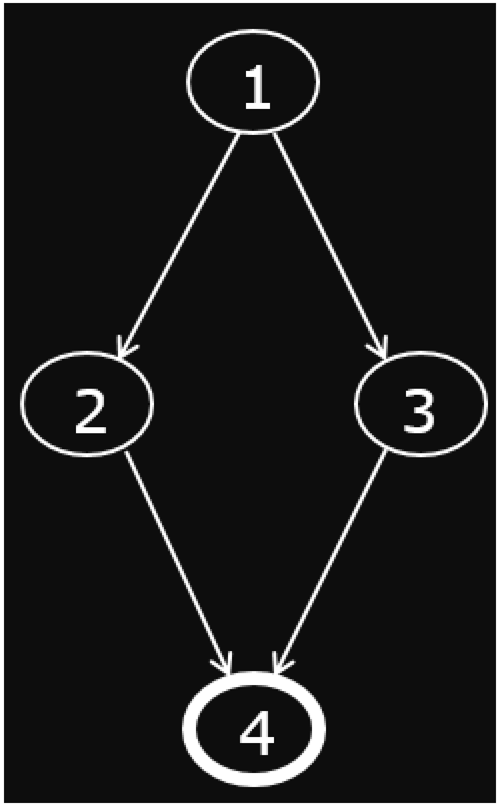
  N_0 = {1, 2, 3}

  N_f = {8, 9, 10}

- Edge

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

Example Graph



- Node

  $N = \{1, 2, 3, 4\}$

  $\boxed{N_0 = \{\}}$

  $N_f = \{4\}$

- Edge

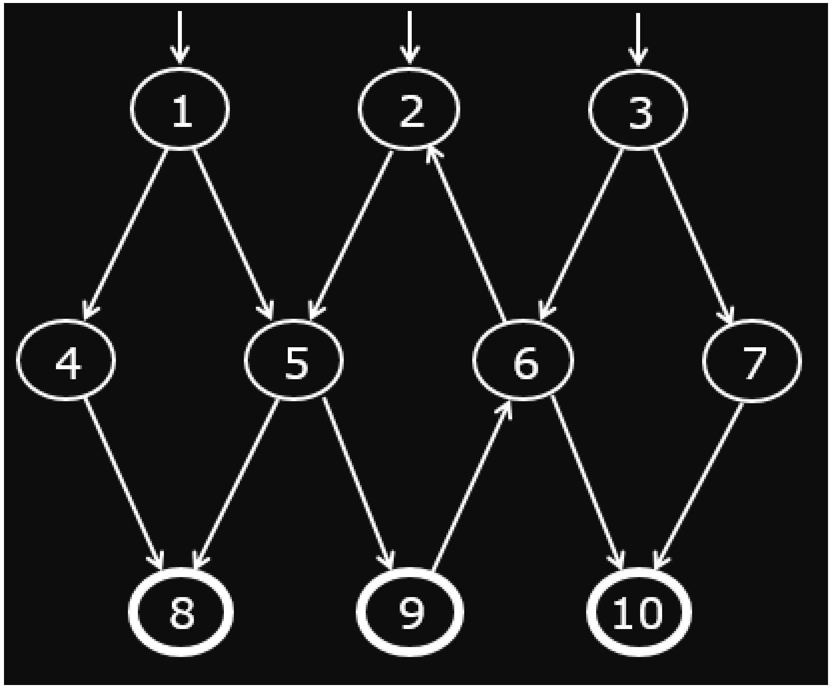
  $E = \{(1,2), (1,3), (2,4), (3,4)\}$

Not valid graph – no initial nodes
Not useful for generating test cases

Paths in Graphs

- **Path** *p*
  - A sequence of nodes, $[n_1, n_2, \ldots, n_M]$ State
  - 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 nodes in *p* (possibly *p* itself)

Example Paths



- Paths

  [1, 4, 8]

  [2, 5, 8]

  [2, 5, 9]

  [2, 5, 9, 6, 10]

  [3, 6, 10]

  [3, 7, 10]

  [3, 6, 2, 5, 9]

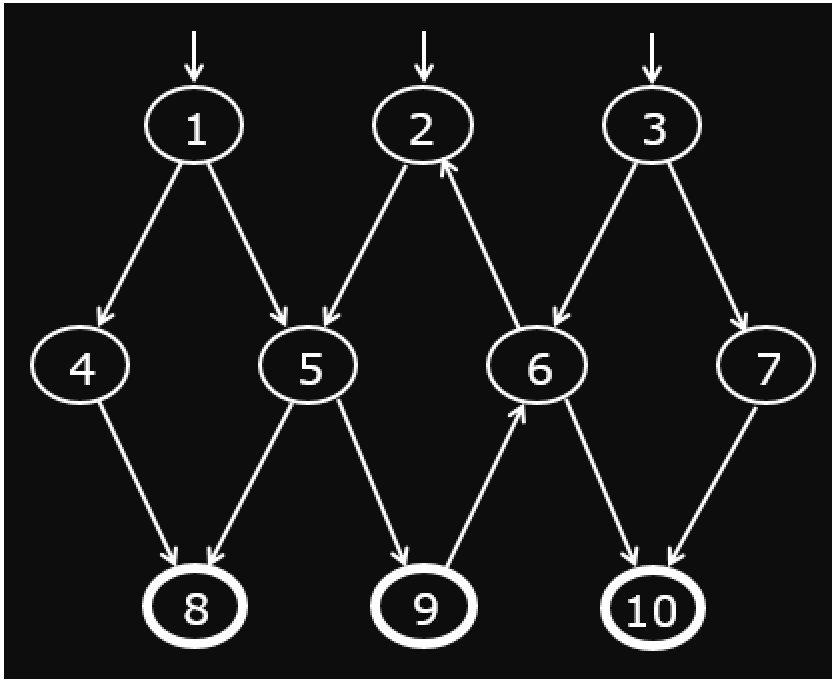
  …

  cycle

  [2, 5, 9, 6, 2]

Cycle – a path that begins and ends at the same node

Example Paths

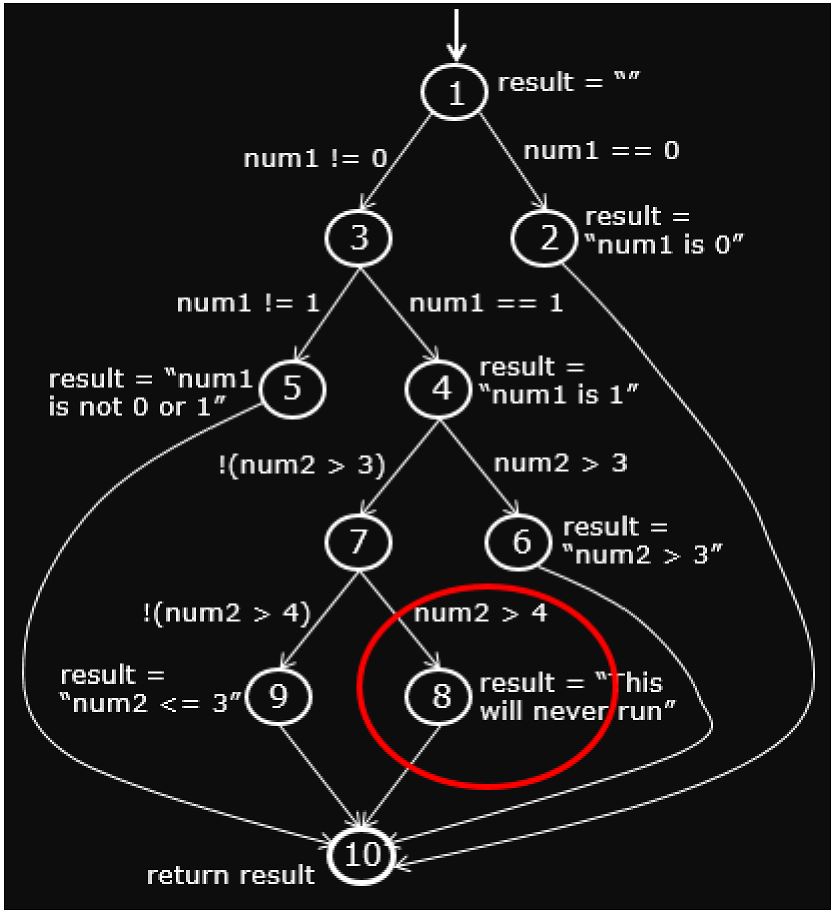


- Invalid paths
  - [1, 8]
  - [4, 5]
  - [3, 7, 9]

Invalid path – a path where the two nodes are not connected by an edge

Example Invalid Path

```
def template(num1, num2): (python)
    result = ""
    if num1 == 0:
        result = "num1 is 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result = " num2 > 3"
        elif num2 > 4:
            result = " This will never run"
        else:
            result = " num2 <= 3"
    else:
        result = "num1 is not 0 or 1"
    return result
```



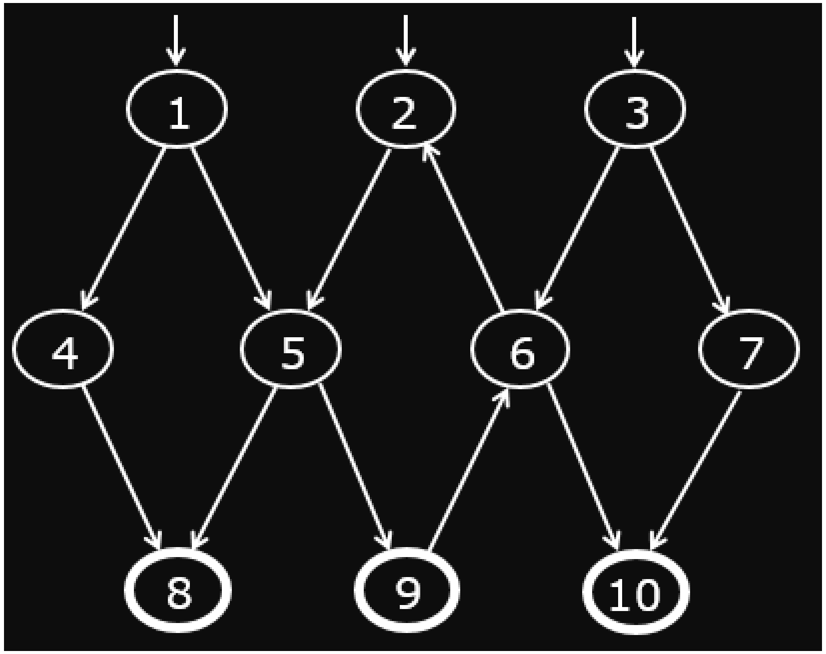
Invalid Paths

- Many test criteria require inputs that start at one node and end at another. – This is only possible if those nodes are **connected** by a path.
- When applying these criteria on specific graphs, we sometimes find that we have asked for a path that for some reason **cannot be executed**.
- Example: a path may demand that a loop be executed zero time, where the program always executed the loop at least once.
- This problem is based on the **semantics** of the software artifact that the graph represents.
- For now, let’s emphasize only the **syntax** of the graph

Graph and Reachability

- 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
- Syntactically reachable
  - There exists a subpath from node n_i to n (or to edge e)
- Semantically reachable
  - There exists a test that can execute that subpath

Example: Reachability



- From node 1
  - Possible to reach all nodes except nodes 3 and 7
- From node 5
  - Possible to reach all nodes except nodes 1, 3, 4, and 7
- From edge (7, 10)
  - Possible to reach nodes 7 and 10 and edge (7, 10)

Some graphs (such as finite state machines) have explicit edges from a node to itself, that is (n_i, n_i)

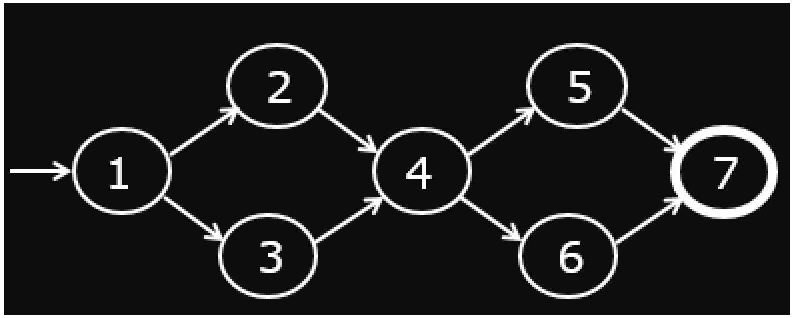
Test Paths

- A path that starts at an initial node and end at a final node

- A test path represents the execution test cases
  - Some test paths can be executed by many test cases
  - Some test paths cannot be executed by any test cases
  - Some test paths cannot be executed because they are infeasible

SESE Graphs

- SESE (Single-Entry-Single-Exit) graphs
  - The set N_0 has exactly one node (n_0)
  - The set N_f has exactly one node (n_f), n_f may be the same as n_0
  - n_f must be syntactically reachable from every node in N
  - No node in N (except n_f) be syntactically reachable from n_f (unless n_0 and n_f are the same node)

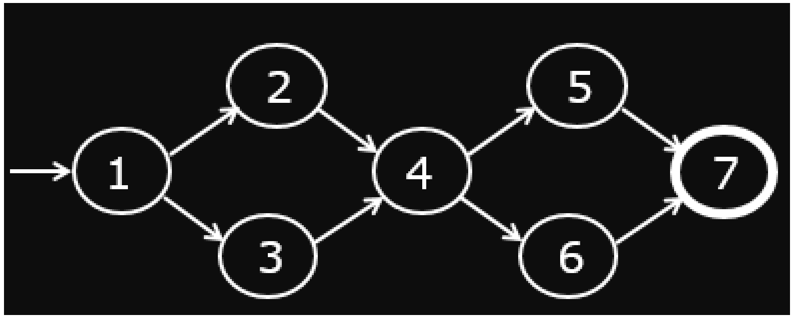


Double-diamonded graph
(two if-then-else statements)
4 test paths

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

Visiting

- A test path *p* visits node *n* if *n* is in *p*
- A test path *p* visits edge *e* if *e* is in *p*



Node *N* = {1, 2, 3, 4, 5, 6, 7}

Edge *E* = {(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), (6,7)}

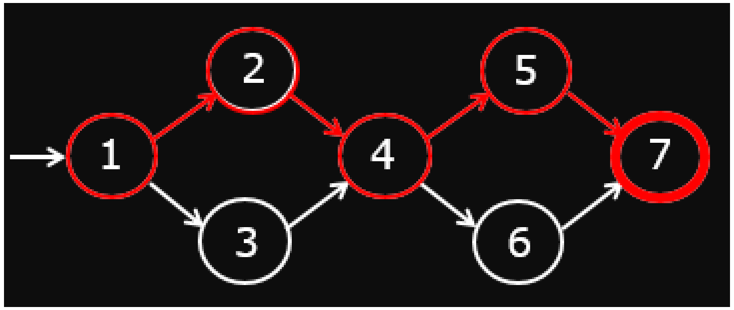
Consider path [1, 2, 4, 5, 7]

Visits node: 1, 2, 5, 4, 7

Visits edge: (1,2), (2,4), (4,5), (5,7)

Touring

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



Node *N* = {1, 2, 3, 4, 5, 6, 7}

Edge *E* = {(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), (6,7)}

(Each edge is technically a subpath)

Visit notes: 1, 2, 4, 5, 7

Visit edges: (1,2), (2,4), (4,5), (5,7)

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

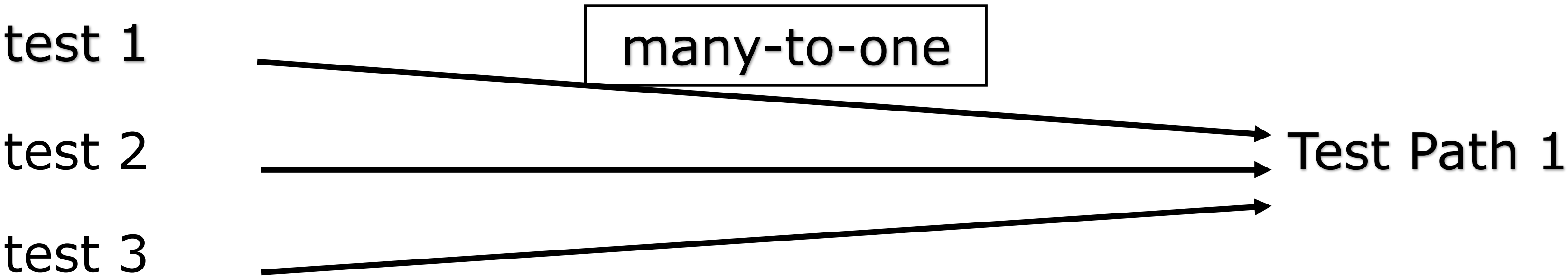
Any given path *p* always tours itself

Mapping: Test Cases – Test Paths

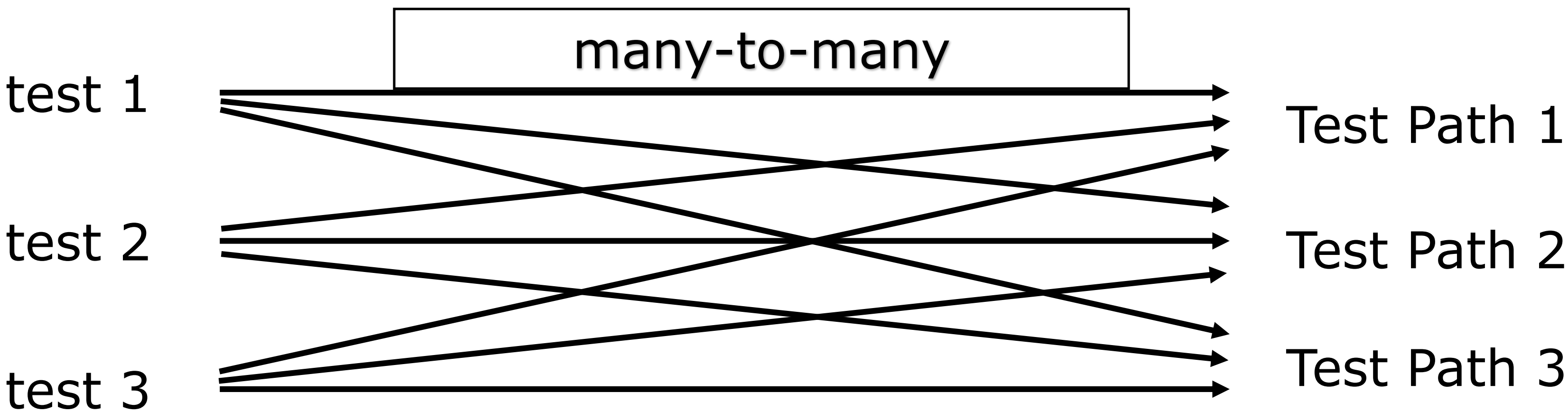
- **path(*t*) =** Test path executed by test case *t*
- **path(*T*) =** Set of test paths executed by set of tests *T*
- Test path is a complete execution from a start node to a final node

- **Minimal** set of test paths = the fewest test paths that will satisfy test requirements
  - Taking any test path out will no longer satisfy the criterion

Mapping: Test Cases – Test Paths

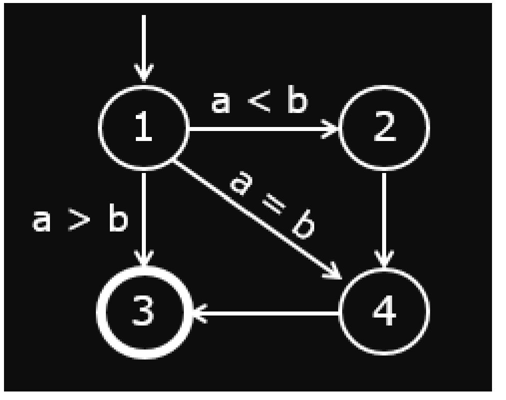


Deterministic software: test always executes the same test path



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

Example Mapping Test Cases – Test Paths



map to

Test case t1: (a=0, b=1) [Test path p1: 1, 2, 4, 3]

Test case t2: (a=1, b=1) [Test path p2: 1, 4, 3]

Test case t3: (a=2, b=1) [Test path p3: 1, 3]

Graph Coverage Criteria

Graph coverage criteria define test requirements TR in terms of properties of test paths in a graph G

Steps:

1. Develop a model of the software as a graph
2. A test requirement is met by visiting a particular node or edge or by touring a particular path

Test requirements (TR)

- Describe properties of test paths

Test criterion

- Rules that define test requirements

Graph Coverage Criteria

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 path(T) that meets the test requirement tr*

Two types

1. **Structural coverage criteria**
   - Define a graph just in terms of nodes and edges
2. **Data flow coverage criteria**
   - Requires a graph to be annotated with references to variables

Graph Coverage Criteria

Structural Coverage Criteria

- Node Coverage (NC)
  - Statement coverage
- Edge Coverage (EC)
  - Branch coverage
- Edge-Pair Coverage (EPC)
- Complete Path Coverage (CPC)
- Prime Path Coverage (PPC)

Data Flow Coverage Criteria

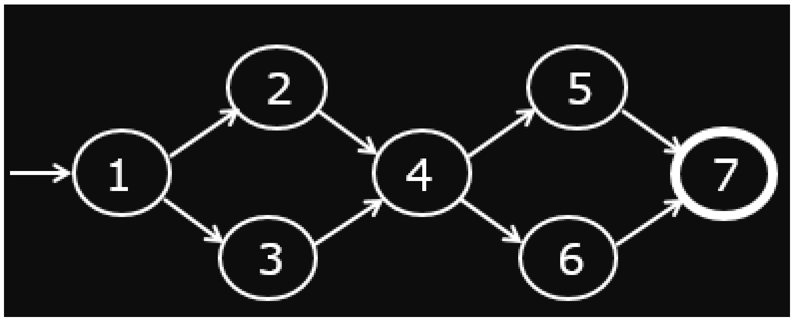
All-Defs Coverage (ADC)

All-Uses Coverage (AUC)

All-du-Paths Coverage (ADUPC)

Node Coverage (NC)

NC: TR contains each reachable node in G



Node *N* = {1, 2, 3, 4, 5, 6, 7}

Edge *E* = {(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), 6,7)}

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

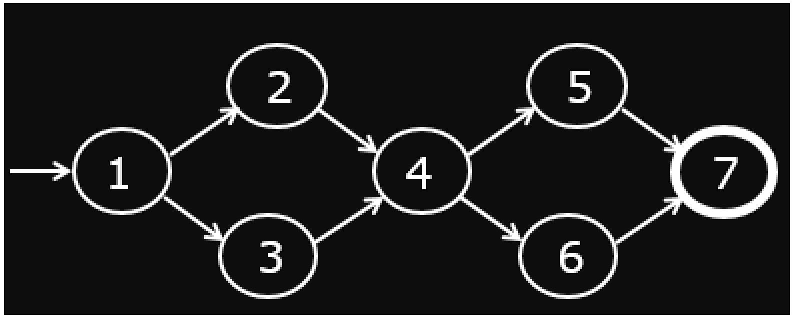
Test path *p1* = [1, 2, 4, 5, 7]

Test path *p2* = [1, 3, 4, 6, 7]

a test set *T* = {*t1*, *t2*},
where path(*t1*) = *p1* and path(*t2*) = *p2*,
Then *T* satisfies Node Coverage on G

Edge Coverage (EC)

EC: TR contains each reachable path of length up to 1, inclusive, in G



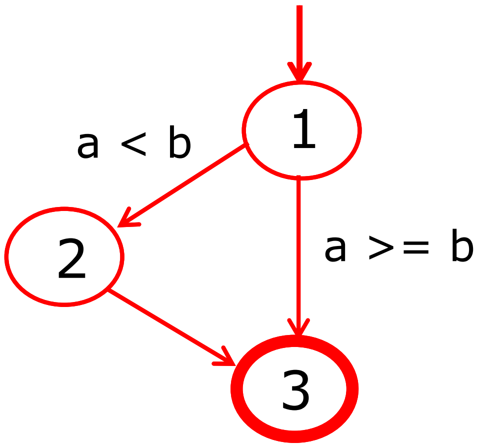
Node N = {1, 2, 3, 4, 5, 6, 7}

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

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

set T = {$t1$, $t2$},
where path($t1$) = $p1$ and path($t2$) = $p2$,
Then T satisfies Edge Coverage on G

Difference between NC and EC



Node N = {1, 2, 3}

Edge E = {(1,2), (1,3), (2,3)}

NC: TR = {1,2, 3}

Test path = [1, 2, 3]

EC: TR = {(1,2), (1,3), (2,3)}

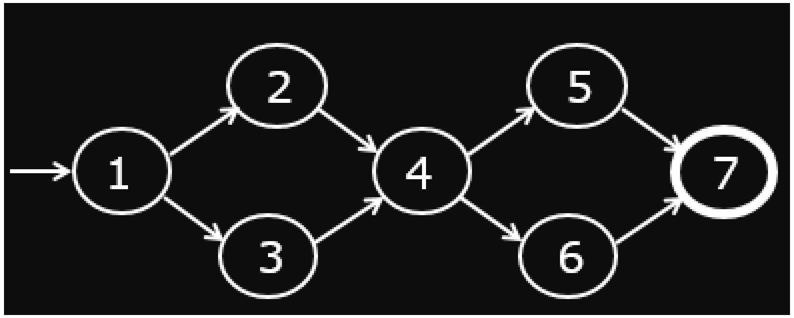
Test paths = [1, 2, 3], [1, 3]

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

Edge-Pair Coverage (EPC)

EPC: TR contains each reachable path of length up to 2, inclusive, in G

"length up to 2" – allows for graphs that have less than 2 edges



Node N = {1, 2, 3, 4, 5, 6, 7}

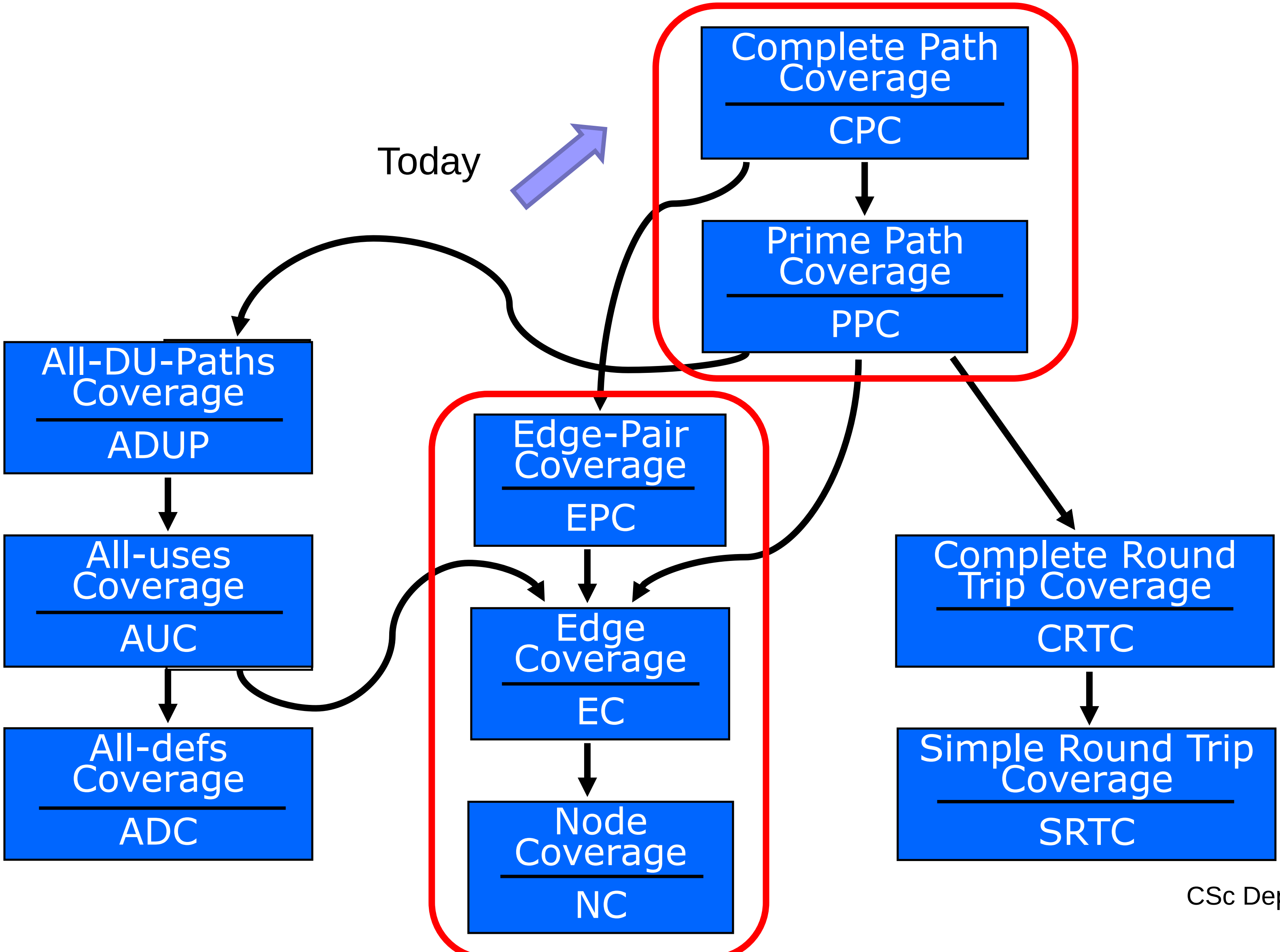
Edge E = {(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), (6,7)}

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

Test path *p1* = [1, 2, 4, 5, 7]
Test path *p2* = [1, 3, 4, 5, 7]
Test path *p3* = [1, 2, 4, 6, 7]
Test path *p4* = [1, 3, 4, 6, 7]

EPC requires pairs of edges, or subpaths of length 2 – covering multiple edges

Graph Coverage Criteria Subsumption

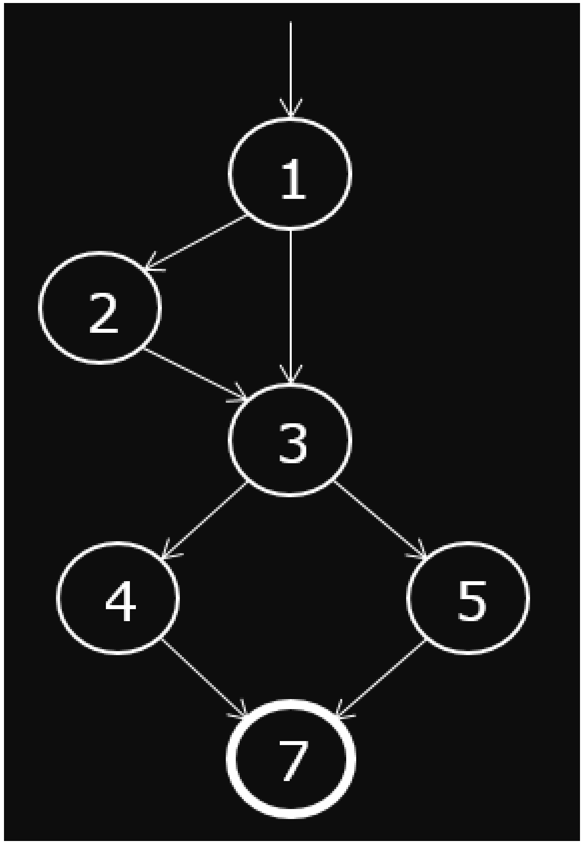


Today's Objectives

- Understand the concepts of simple paths and prime paths
- Understand how to use graph to define criteria and design tests
  - Complete Path Coverage (CPC)
  - Prime Path Coverage (PPC)
- Touring, sidetrips, and detours
- Dealing with infeasible test requirements

Complete Path Coverage (CPC)

CPC: TR contains all paths in G



Node N = {1, 2, 3, 4, 5, 6, 7}

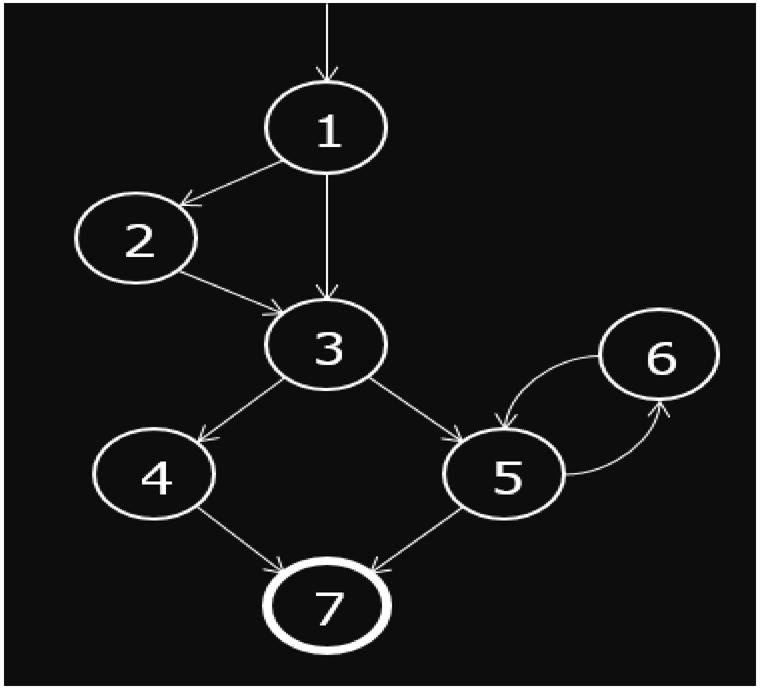
Edge E = {(1,2), (1,3), (2,3), (3,4), (3,5), (4,7), (5,7)}

List all test paths:

Test path *p1* = [1, 2, 3, 4, 7]
Test path *p2* = [1, 2, 3, 5, 7]
Test path *p3* = [1, 3, 4, 7]
Test path *p4* = [1, 3, 5, 7]

TR = {p1,p2,p3,p4}

CPC: Graph with Loop



Node N = {1, 2, 3, 4, 5, 6, 7}

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

List all test paths:

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

Impossible if a graph has a loop
≈ infinite number of paths
≈ infinite number of test requirements

Handling Loops in Graphs

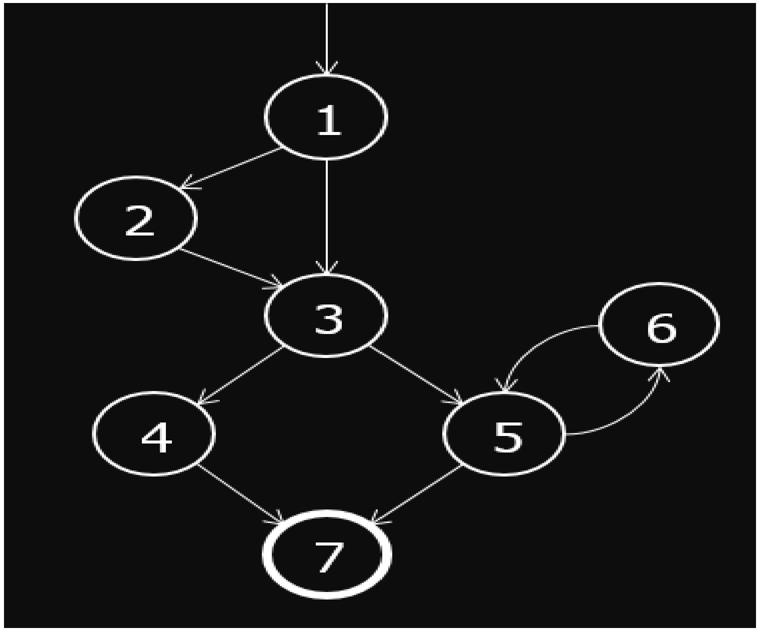
Attempts to deal with loops:

- 1970s: Execute cycles once ([5, 6, 5] in previous example)
- 1980s: Execute each loop, exactly once
- 1990s: Execute loops 0 times, once, more than once
- 2000s: Prime paths (touring, sidetrips, and detours)

Simple Paths

Path from node n_i to n_j that is **no internal loops**

- A path from ni to nj is simple if no node appears more than once in the path (first and last nodes may be identical)
- A loop is a simple path



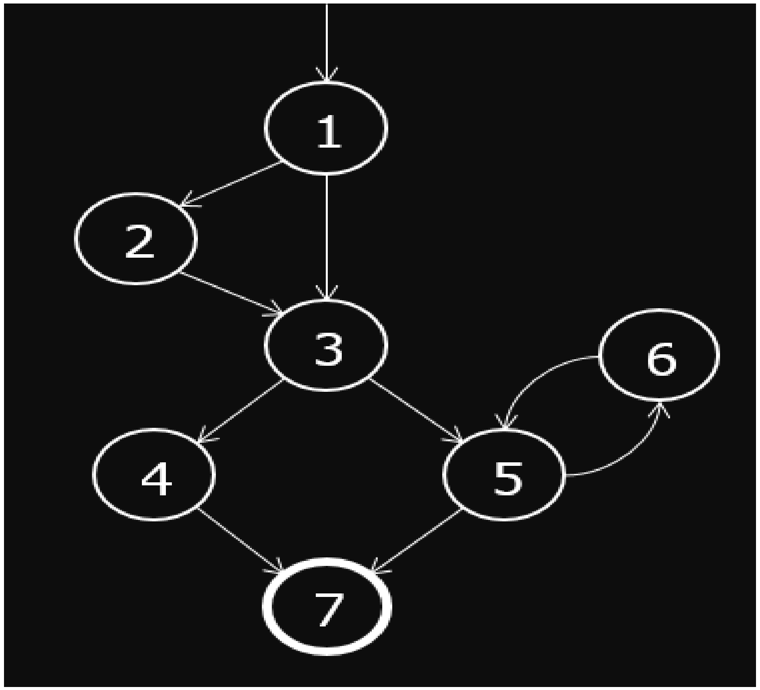
List simple paths: 31 simple paths

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

Subpaths of other simple paths → avoid these

Prime Paths

Simple path that is **not subpath** of any other simple path



List prime paths: 9 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],

[5,6,5],

[6,5,6], [6,5,7]

Execute loop 0 time

Execute loop once

Execute loop more than once

Prime Path Coverage (PPC)

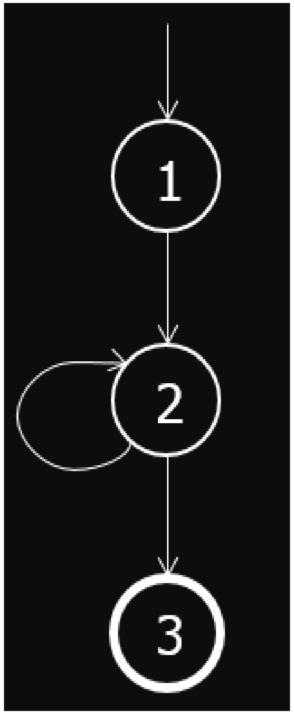
PPC: TR contains each prime path in graph G

- Keep the number of test requirements down
- For a given infeasible prime path that consists of some feasible simple paths, replace the infeasible prime path with relevant feasible subpaths

Note on PPC

- PPC does not subsume EPC
- If a node *n* has an edge to itself ("self edge"), EPC requires [*n*, *n*, *m*] and [*m*, *n*, *n*]
- [*n*, *n*, *m*] and [*m*, *n*, *n*] are not simple paths (prime paths)

Recall Simple Path => Path from node n_i to n_j that is **no internal loops**



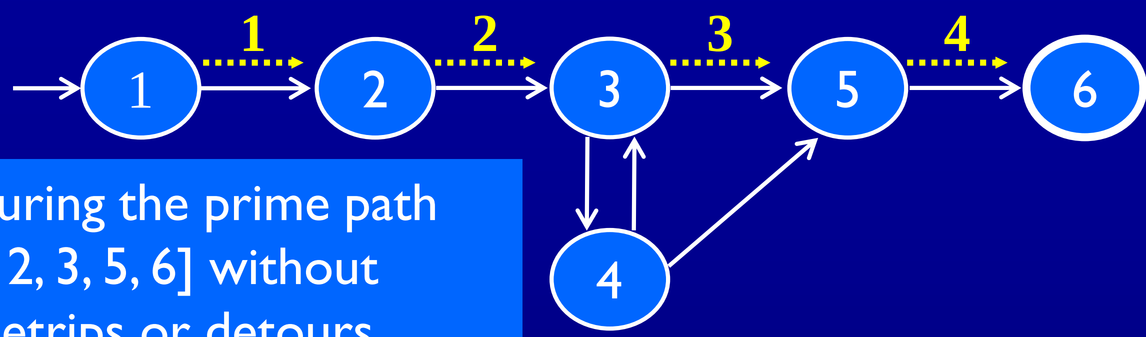
List EPC requirements:

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

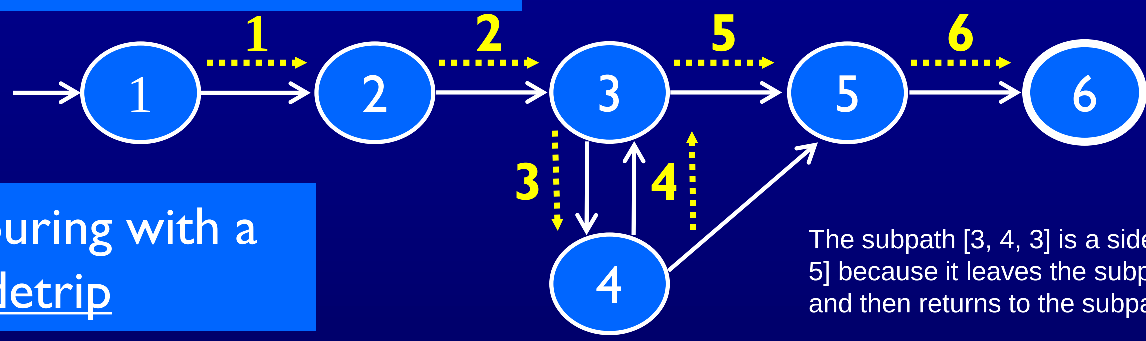
List PPC requirements:

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

Sidetrips and Detours Example

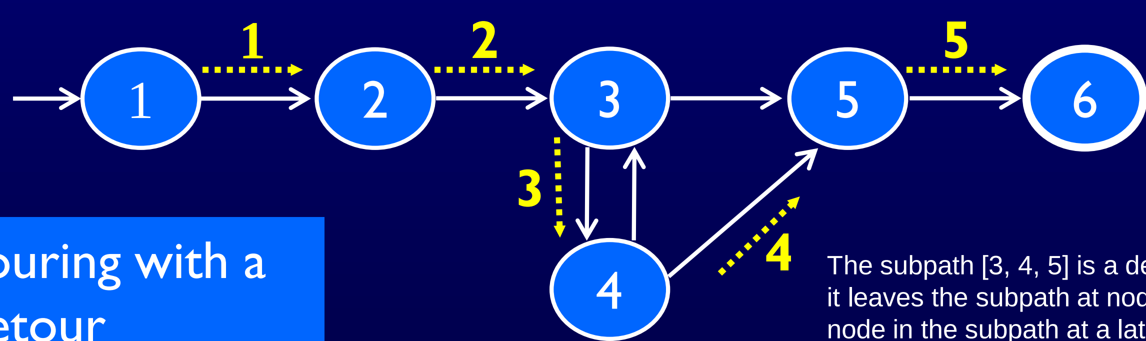


Touring the prime path [1, 2, 3, 5, 6] without <u>sidetrips or detours</u>



Touring with a <u>sidetrip</u>

The subpath [3, 4, 3] is a sidetrip to [2, 3, 5] because it leaves the subpath at node 3 and then returns to the subpath at node 3



Touring with a <u>detour</u>

The subpath [3, 4, 5] is a detour to [2, 3, 5] because it leaves the subpath at node 3 and then returns to a node in the subpath at a later point

[AO, Figures 7.8, 7.9]

Touring with SideTrips and Detours

- **Tour**: Test path p is said to tour subpath q if and only if q is a subpath of p.

- **Tour With Sidetrips**: Test path p is said to tour subpath q with sidetrips if and only if every edge in q is also in p in the same order.

- **Tour With Detours**: Test path p is said to tour subpath q with detours if and only if every node in q is also in p in the same order.

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 to know whether all test requirements are feasible
- When sidetrips are not allowed, many structural criteria have more infeasible test requirements
- 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

Graph Coverage Criteria Subsumption