Graph Coverage Overview

Isaac Griffith

CS 4422 and CS 5522
Department of Computer Science
Idaho State University

Outcomes

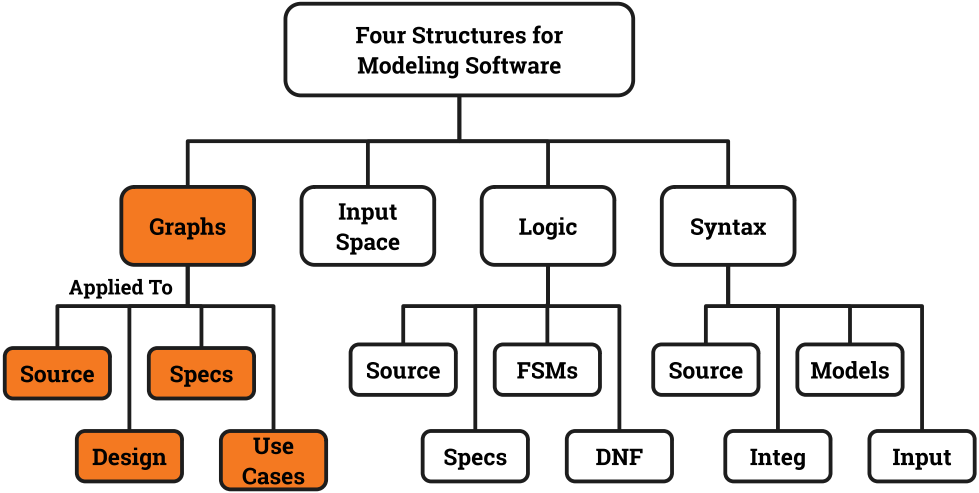
At the end of Today's Lecture you will be able to:

- Understand the basic concepts of graph coverage
- Understand def, use, and du pairs
- Evaluate a given graph for graph coverage criteria

Inspiration

“All software is a graph” – Anonymous

Graph Coverage



Covering Graphs

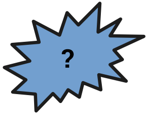
- Graphs are the most **commonly** used structure for testing
- Graphs can come from **many sources**
  - Control flow graphs
  - Design structure
  - FSMs and statecharts
  - Use cases
- Tests usually are intended to “**cover**” the graph in some way

Definition of 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?

$N_0 = \{1\}$
$N_f = \{1\}$
$E = \{\}$



ROAR

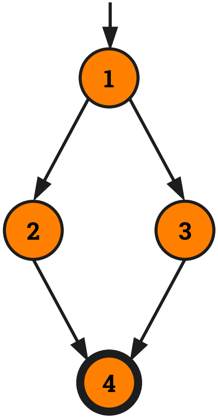
Definition of a Graph

- 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**

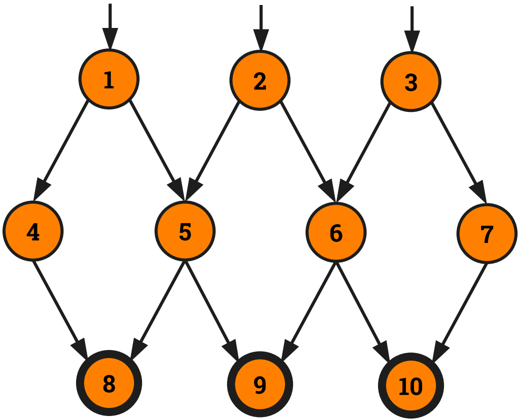
Is this a graph?

$N_0 = \{1\}$
$N_f = \{1\}$
$E = \{\}$

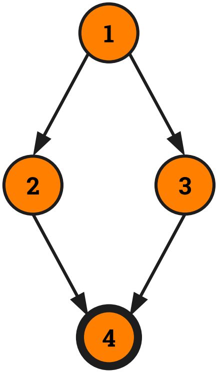
Example Graphs



$N_0 = \{ 1 \}$
$N_f = \{ 4 \}$
$E = \{ (1, 2), (1, 3), (2, 4), (3, 4) \}$

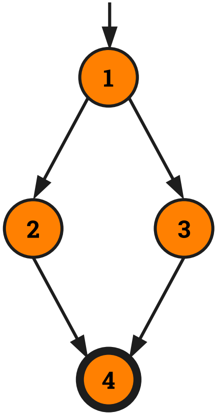


$N_0 = \{ 1, 2, 3 \}$
$N_f = \{ 8, 9, 10 \}$
$E = \{ (1,4), (1,5), (2, 5), (3, 6), (3, 7), (4, 8), (5, 8), (5, 9), (6, 2), (6, 10), (7, 10), (9, 6) \}$

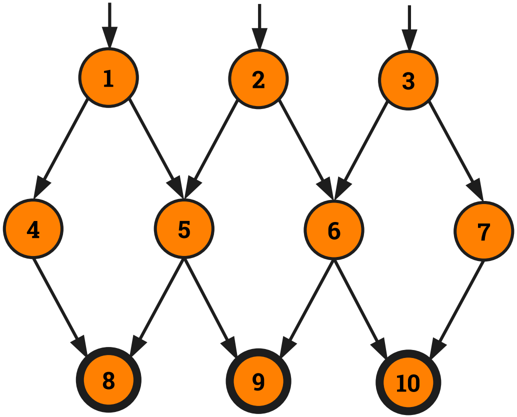


$N_0 = \{ \}$
$N_f = \{ 4 \}$
$E = \{ (1, 2), (2, 4), (3, 4)\}$

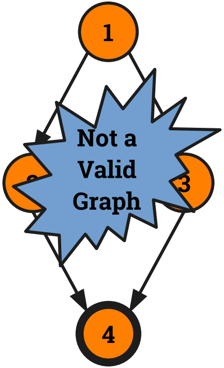
Example Graphs



$N_0 = \{ 1 \}$
$N_f = \{ 4 \}$
$E = \{ (1, 2), (1, 3), (2, 4), (3, 4) \}$



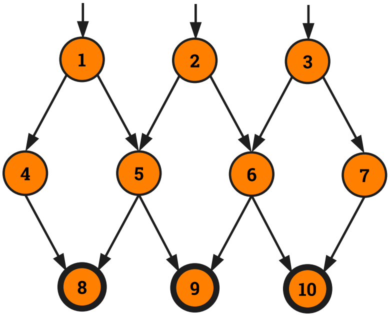
$N_0 = \{ 1, 2, 3 \}$
$N_f = \{ 8, 9, 10 \}$
$E = \{ (1,4), (1,5), (2, 5), (3, 6), (3, 7), (4, 8), (5, 8), (5, 9), (6, 2), (6, 10), (7, 10), (9, 6) \}$



$N_0 = \{ \}$
$N_f = \{ 4 \}$
$E = \{ (1, 2), (2, 4), (3, 4)\}$

Paths in Graphs

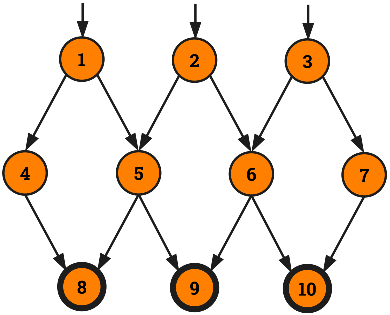
- **Path**: A sequence of nodes - $[n_1, n_2, \ldots, n_M]$
  - Each pair of nodes is an edge
- **Length**: The number of edges
  - A single node is a path of length 0
- **Subpath**: A subsequence of nodes in p is a subpath of p



Write down three paths in this graph

Paths in Graphs

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

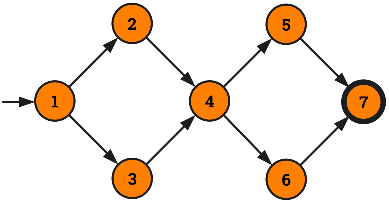


Write down three paths in this graph

- [1, 4, 8]
- [2, 5, 9]
- [3, 7, 10]

Test Paths and SESEs

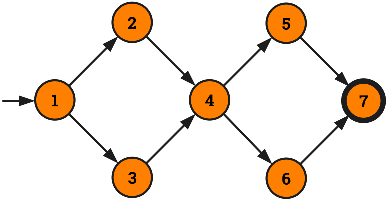
- **Test Path**: A path that starts at an initial node and ends at a final node
- Test paths represent execution of test cases
  - Some test paths can be executed by many tests
  - Some test paths cannot be executed by any tests
- **SESE graphs**: All test paths start at a single node and end at another node
  - Single-entry, Single-exit
  - N_0 and N_f have exactly one node



Write down all the test paths in this graph

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. [1, 2, 4, 5, 7]
2. [1, 2, 4, 6, 7]
3. [1, 3, 4, 5, 7]
4. [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**?
- **Visits Edges**?
- **Tours subpaths**?

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**?
- **Tours subpaths**?

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**?

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]

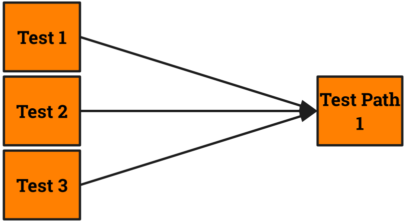
- **Visits Nodes**? 1, 2, 4, 5, 7
- **Visits Edges**? (1,2), (2,4), (4,5), (5,7)
- **Tours subpaths**? [1,2,4], [2,4,5], [4,5,7], [1,2,4,5], [2,4,5,7], [1,2,4,5,7]

(Also, each edge is technically a subpath)

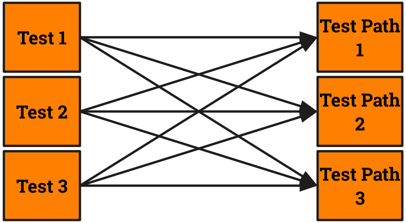
Tests and Test Paths

- **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 an 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

Testing and Covering Graphs

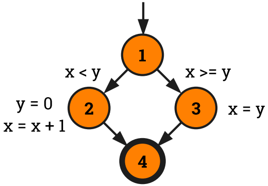
- We use graphs in testing as follows:
  - Develop a model of the software as a graph
  - Require tests to visit or tour specific sets of nodes, edges or subpaths
- **Test Requirements (TR)**: Describe properties of test paths
- **Test Criterion**: Rules that define test requirements
- **Satisfaction**: 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 path(T) that meets the test requirement tr
- **Structural Coverage Criteria**: Defined on a graph just in terms of nodes and edges.

Node and Edge Coverage

- The first (and simplest) two criteria require that each node and edge in a graph be executed
  Node Coverage (NC): Test set T satisfies node coverage on graph G iff 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.

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



Node Coverage

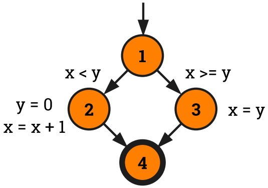
- TR?
- Test Paths

Edge Coverage

- TR?
- Test Paths

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



Node Coverage

- TR? = {1,2,3,4}
- Test Paths = [1, 2, 4] [1, 3, 4]

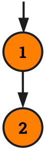
Edge Coverage

- TR? = {(1,2), (1,3), (2,4), (3,4)}
- Test Paths = [1, 2, 4] [1, 3, 4]

Paths of Length 1 and 0

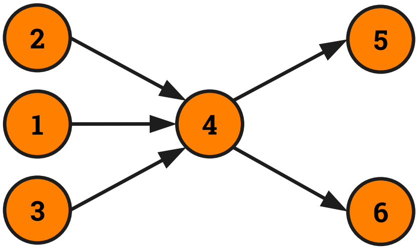
- A graph with **only one node** will not have any edges

- 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 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 phrase "**length up to 2**" is used to include graphs that have less than 2 edges



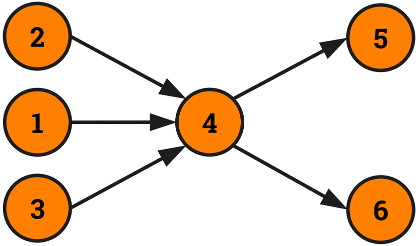
Edge-Pair Coverage:

- TR = ?

- The logical extension is to require **all paths** ...

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 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** ...

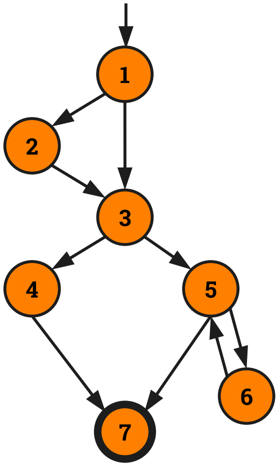
Covering Multiple Edges

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

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

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

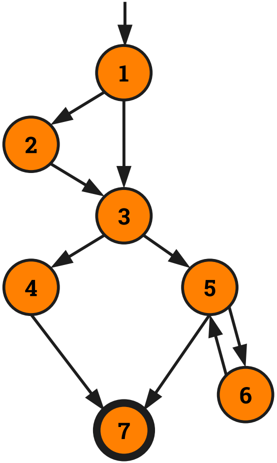
Structural Coverage Example



Write down the TRs and
Test paths for these criteria:

- Node Coverage
- Edge Coverage
- Edge-Pair Coverage
- Complete Path Coverage

Structural Coverage Example



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] [1,2,3,5,7] [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 ([4,5,4] in previous example, informal)
  - **1980s**: Execute each loop, exactly once (formalized)
  - **1990s**: Execute loops 0 times, once, more than once (informal description)
  - **2000s**: Prime paths (touring, sidetrips, and detours)

Are there any questions?