Graph Coverage Overview Part 2

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:

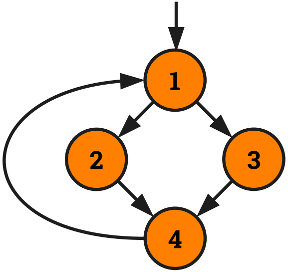
- Understand the basic concepts of graph coverage
- Understand def, use, and du pairs
- Evaluate a given graph for graph coverage criteria

Inspiration

"All software is a graph" – Anonymous

Simple Paths and Prime Paths

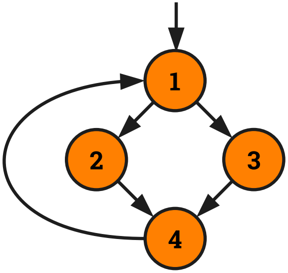
- **Simple Path**: A path from node n_i to n_j is simple if no node appears more than once, except possibly the first and last nodes are the same
  - No internal loops
  - A loop is a simple path
- **Prime Path**: A simple path that does not appear as a proper subpath of any other simple path



Write down the simple and prime paths for this graph

Simple Paths and Prime Paths

- **Simple Path**: A path from node n_i to n_j is simple if no node appears more than once, except possibly 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



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

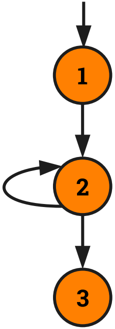
[2,4,1,2], [2,4,1,3], [1,3,4,1], [1,2,4,1], [3,4,1,2], [4,1,3,4], [4,1,2,4], [3,4,1,3]

Prime Path Coverage

- A simple, elegant and finite criterion that requires **loops** to be executed as well as skipped
  Prime Path Coverage (PPC): TR contains each prime path in G
- Will tour all paths of length 0, 1, ...
- That is, it **subsumes** node and edge coverage
- PPC almost, but **not quite**, subsumes **EPC** ...

PPC Does Not Subsume EPC

- If a node n has an edge to itself (self edge), **EPC** requires $[n, n, m]$ and $[m, n, n]$
- $[n, n, m]$ is not prime
- Neither $[n, n, m]$ nor $[m, n, n]$ are simple paths (not prime)



EPC Requirements:

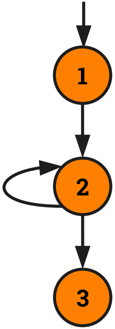
TR = ?

PPC Requirements:

TR = ?

PPC Does Not Subsume EPC

- If a node n has an edge to itself (self edge), **EPC** requires $[n, n, m]$ and $[m, n, n]$
- $[n, n, m]$ is not prime
- Neither $[n, n, m]$ nor $[m, n, n]$ are simple paths (not prime)



EPC Requirements:

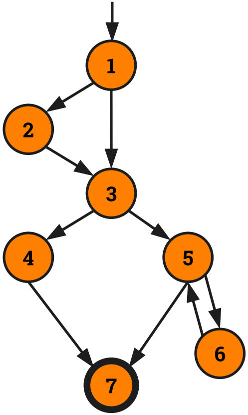
TR = { [1,2,3], [1,2,2], [2,2,3], [2,2,2]}

PPC Requirements:

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

Prime Path Example

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

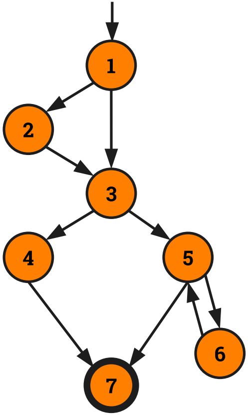


Prime Paths

Write down all 9 prime paths

Prime Path Example

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



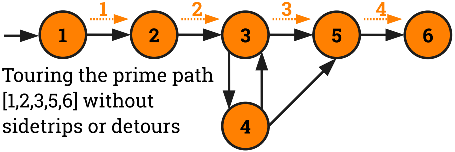
Prime Paths

[1, 2, 3, 4, 7]
[1, 2, 3, 5, 7]
[1, 2, 3, 5, 6]
[1, 3, 4, 7]
[1, 3, 5, 7] -> Execute Loop 0 times
[1, 3, 5, 6] -> Execute loop once
[6, 5, 7]
[6, 5, 6] -> Execute loop more than once
[5, 6, 5]

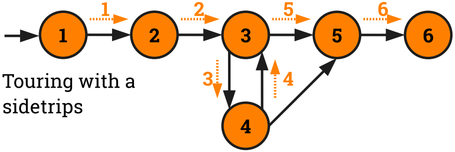
Touring, Sidetrips, and Detours

- 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 n_i, as long as it comes back to the prime path at a successor of n_i

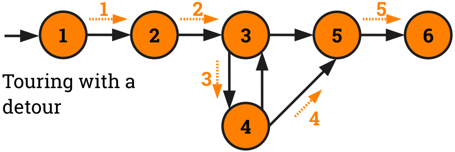
Sidetrips and Detours Example



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



Touring with a sidetrips



Touring with a detour

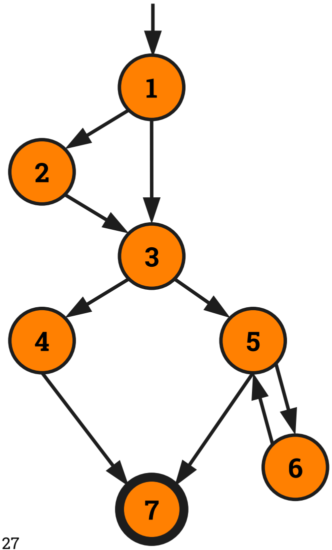
Infeasible Test Requirements

- An **infeasible** test requirement <u>cannot be satisfied</u>
  - 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

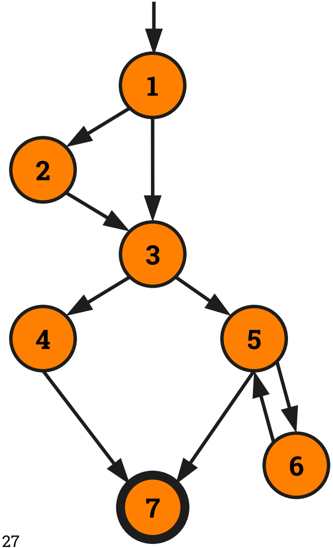
Simple & Prime Path Example



Simple Paths

Write all paths of length 0
Write Paths of length 1
Write paths of length 2
Write paths of length 3
Write paths of length 4

Simple & Prime Path Example



Simple Paths

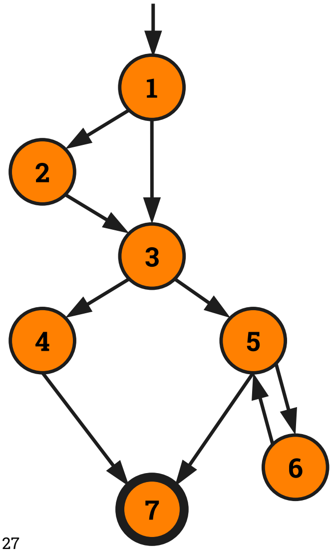
Length 0: [1] [2] [3] [4] [5] [6] [7]!
Length 1: [1,2] [1,3] [2,3] [3,4] [3,5] [4,7]! [5,7]! [5,6] [6,5]
Length 2: [1,2,3] [1,3,4] [1,3,5] [2,3,4] [2,3,5] [3,4,7]! [3,5,7]! [3,5,6] [5,6,5]* [6,5,7]! [6,5,6]*
Length 3: [1,2,3,4] [1,2,3,5] [1,3,4,7]! [1,3,5,7]! [1,3,5,6]! [2,3,4,7]! [2,3,5,6]! [2,3,5,7]!
Length 4: [1,2,3,4,7]! [1,2,3,5,7]! [1,2,3,5,6]!

Prime Paths?

! means path terminates

* means path is cyclic

Simple & Prime Path Example



Simple Paths

Length 0: [1] [2] [3] [4] [5] [6] [7]!
Length 1: [1,2] [1,3] [2,3] [3,4] [3,5] [4,7]! [5,7]! [5,6] [6,5]
Length 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]***
Length 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]!
Length 4: **[1,2,3,4,7]! [1,2,3,5,7]! [1,2,3,5,6]!**

! means path terminates
* means path is cyclic
Bold means prime path

ROAR

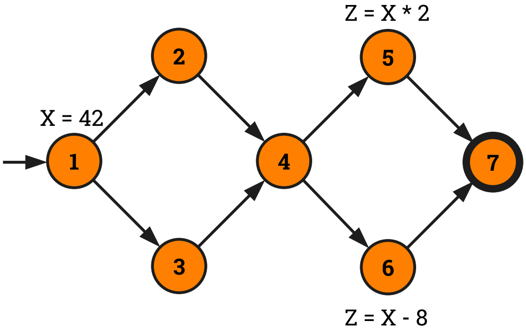
Round Trips

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

Data Flow Criteria

Goal: Ensure that values are computed and used correctly

- **Definition (def)**: A location where a value for a variable is stored into memory.
- **Use**: A location where a variable's value is accessed



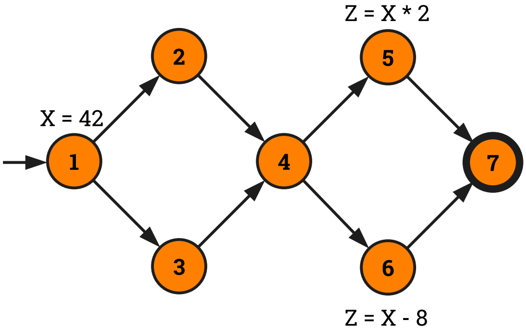
Fill in the Sets

Defs: { }

Uses: { }

The values given in **defs** should **reach** at least one, some, or all possible **uses**

Data Flow Criteria



Fill in the Sets

```
Defs: {
  def (1) = { X }
  def (5) = { Z }
  def (6) = { Z }
}

Uses: {
  use (5) = { X }
  use (6) = { X }
}
```

The values given in **defs** should **reach** at least one, some, or all possible **uses**

DU Pairs and DU Paths

- **def(n) or def (e)**: The set of variables that are defined by node n or edge e
- **use (n) or use (e)**: The set of variables that are used by node n or edge e
- **DU pair**: A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j
- **Def-clear**: A path from l_i to l_j is def-clear with respect to variable v if v is not given another value on any of the nodes or edges in the path
- **Reach**: If there is a def-clear path from l_i to l_j with respect to v, the def of v at l_i reaches the use at l_j
- **du-path**: A simple subpath that is def-clear with respect to v from a def of v to a use of v
- **du**(n_i, n_j, v) - the set of du-paths from n_i to n_j
- **du**(n_i, v) - the set of du-paths that start at n_i

Touring DU-Paths

- A test path p **du-tours** subpath d with respect to v if p tours d and the subpath taken is def-clear with respect to v
- **Sidetrips** can be used, just as with previous touring
- Three criteria
  - Use every def
  - Get to every use
  - Follow all du-paths

Data Flow Test Criteria

- First, we make sure **every def** reaches **a use**

All-defs coverage (ADC):

For each set of du-paths $S = du(n, v)$, TR contains at least one path $d \in S$.

- Then we make sure that **every def** reaches **all** possible **uses**

All-uses coverage (AUC):

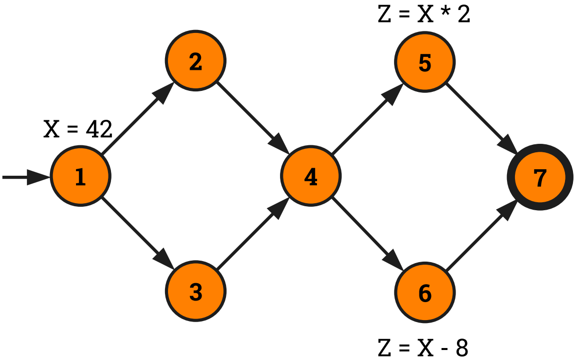
For each set of du-paths to uses $S = du(n_i, n_j, v)$, TR contains at least one path $d \in S$

- Finally, we cover **all the paths** between defs and uses

All-du-paths coverage (ADUPC):

For each set $S = du(n_i, n_j, v)$, TR contains every path $d \in S$.

Data Flow Example



All-defs for X

Write down paths to satisfy ADC

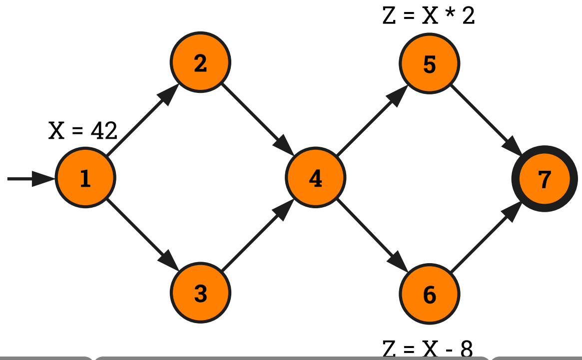
All-uses for X

Write down paths to satisfy AUC

All-du-paths for X

Write down paths to satisfy ADUPC

Data Flow Example



All-defs for X

[1, 2, 4, 5]

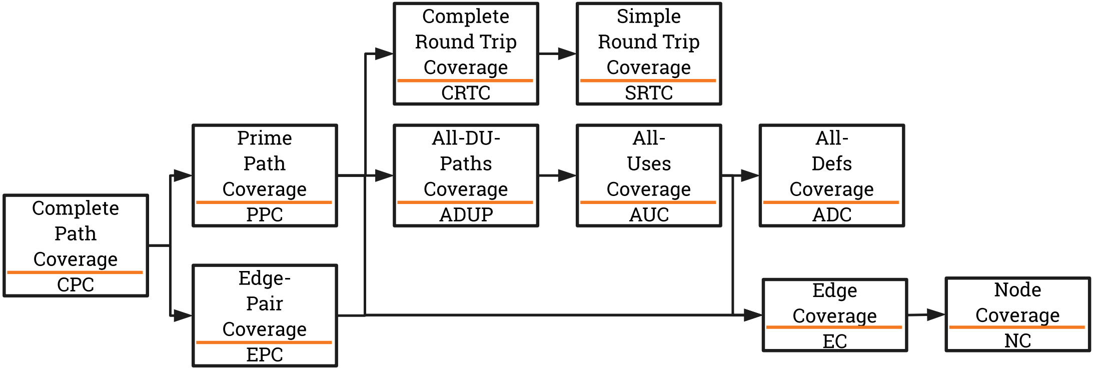
All-uses for X

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

All-du-paths for X

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

Graph Coverage Criteria Subsumption



Summary

- 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 remainder of chapter 7

Are there any questions?