

# Graph: Data Flow Coverage Criteria

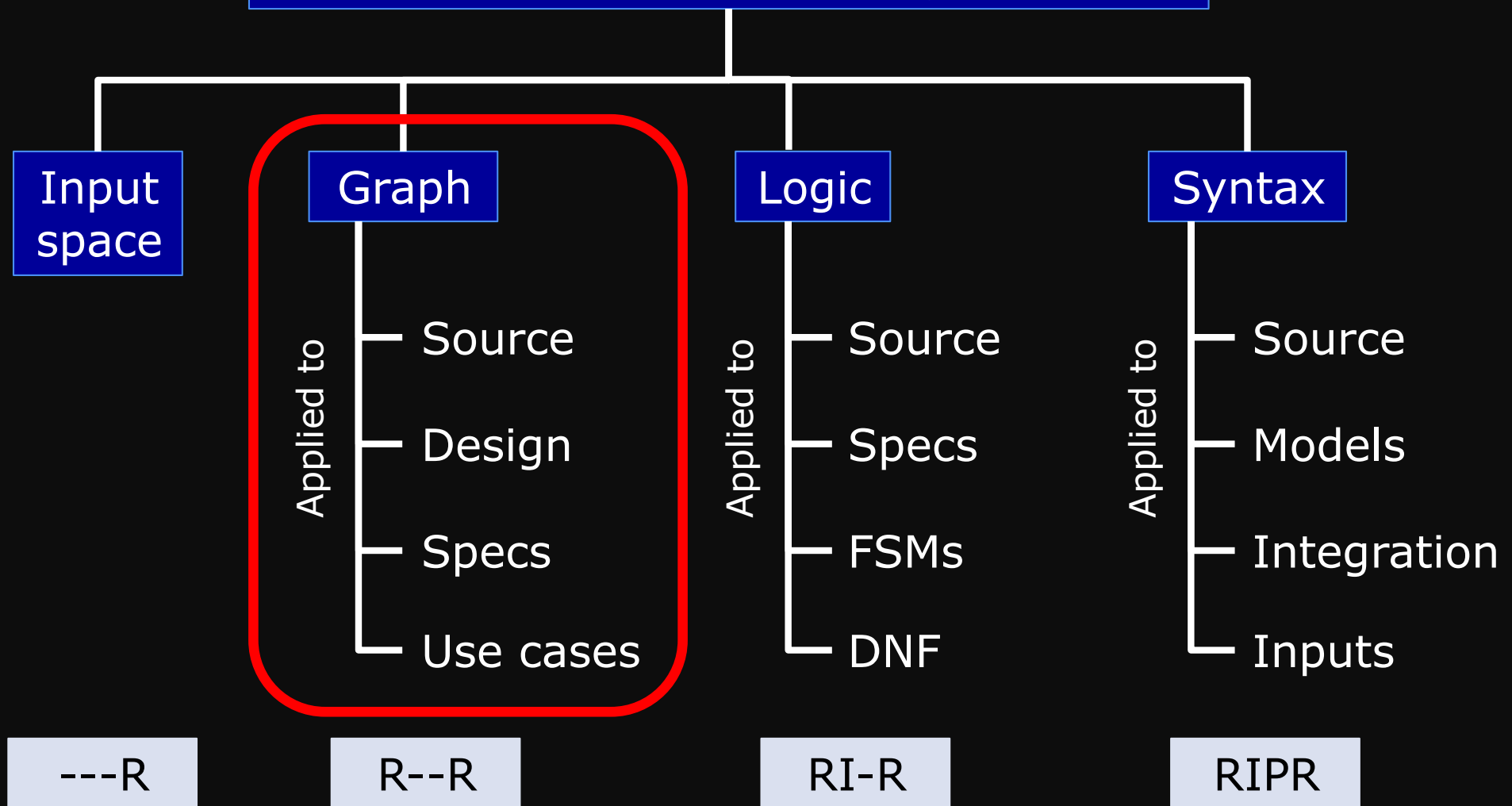
---

## CS 3250 Software Testing

[Ammann and Offutt, “Introduction to Software Testing,” Ch. 7]

# Structures for Criteria-Based Testing

Four structures for modeling software



# 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

# Today's Objectives

- Analyze data flow of software artifacts
- Understand how to integrate data flow into a graph model of the program under test
- Focusing on the flow of data, understand how to define criteria and design tests
  - All-Defs Coverage (ADC)
  - All-Uses Coverage (AUC)
  - All-DU-Paths Coverage (ADUPC)

# Data Flow Criteria

- Goal: Ensure that the values are created and used correctly
- How: Focus on definitions and uses of values
- **Definition (def)**: A location where a value for a variable is stored in a memory
- **Use**: A location where a variable's value is accessed

Values are carried from defs to uses, refer to as  
***"du-pairs"***

- Also known *definition-use, def-use, du associations*

# Data Flow Criteria

Data flow coverage criteria define test requirements TR in terms of the flows of data values in a graph G

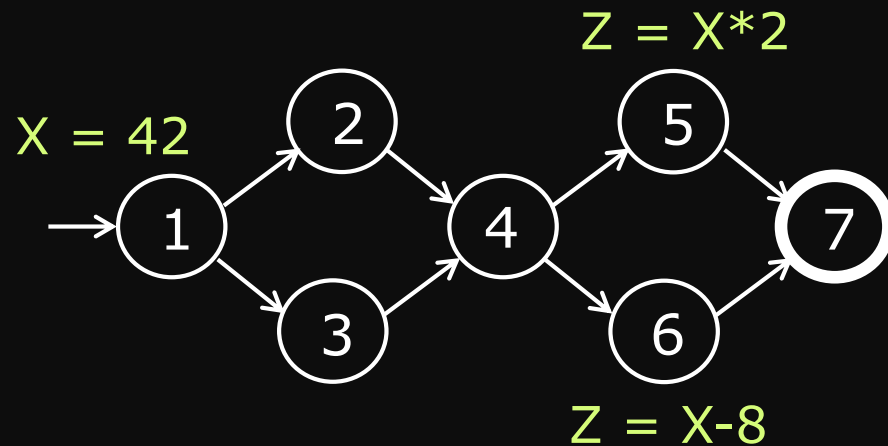
Steps:

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

# Def, Use, and DU Pairs

<b>def(<math>n</math>) or def(<math>e</math>)</b>	The set of variables defined by node $n$ or edge $e$
<b>use(<math>n</math>) or use(<math>e</math>)</b>	The set of variables used by node $n$ or edge $e$
<b>du-pair</b>	A pair of locations $(l_i, l_j)$ such that a variable $v$ is defined at $l_i$ and used at $l_j$

# Example: Defs, Uses, DU-Pairs



defs:

- $\text{def}(1) = \{ X \}$
- $\text{def}(5) = \{ Z \}$
- $\text{def}(6) = \{ Z \}$

uses:

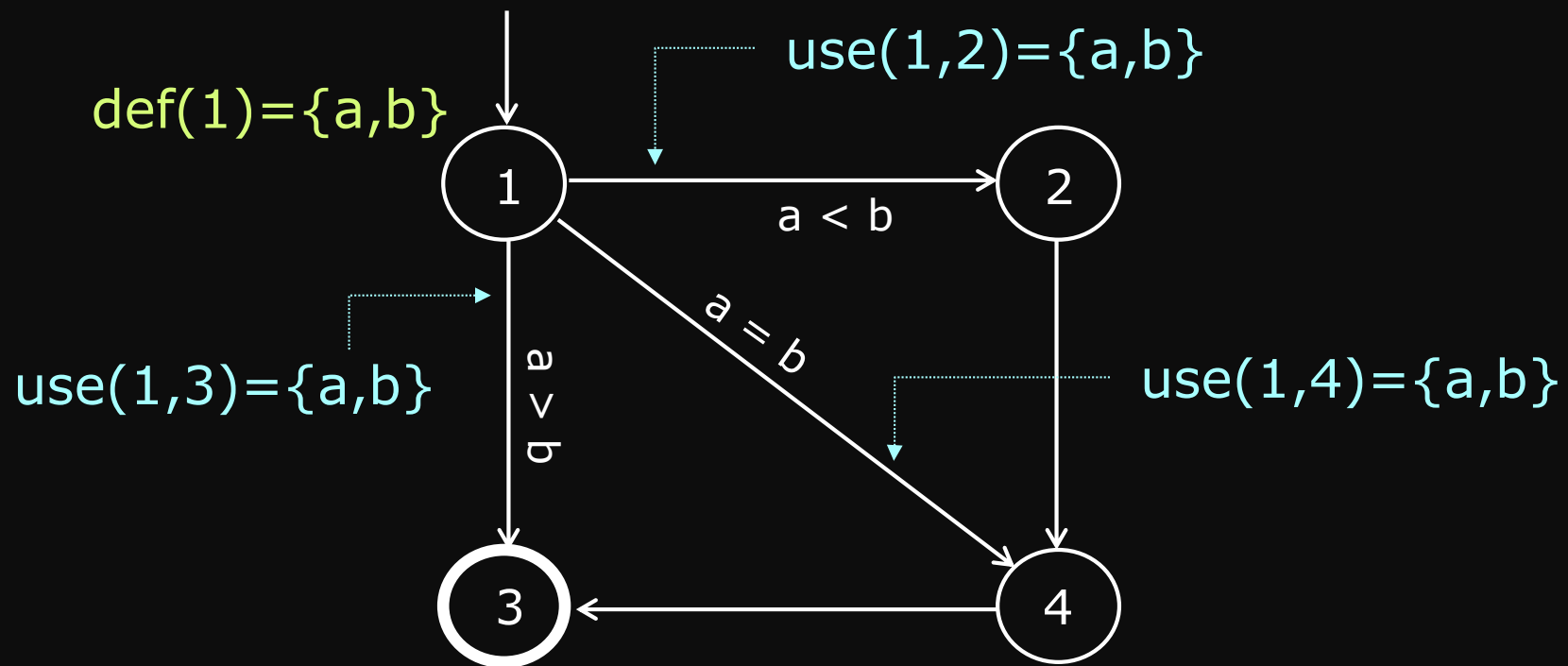
- $\text{use}(5) = \{ X \}$
- $\text{use}(6) = \{ X \}$

du-pairs: for variable X

- $(1, 5)$
- $(1, 6)$

# Example (2)

All variables involved in a decision are assumed to be used on the associated edges



# Def-clear and Reach

**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

- The values given in **defs** should reach at least one, some, or all possible **uses**.
- A def of a variable may or may not reach a particular use.
  - No path goes from a location where a variable is defined to a location where the variable is used
  - A variable's value may be changed by another def before it reaches the use

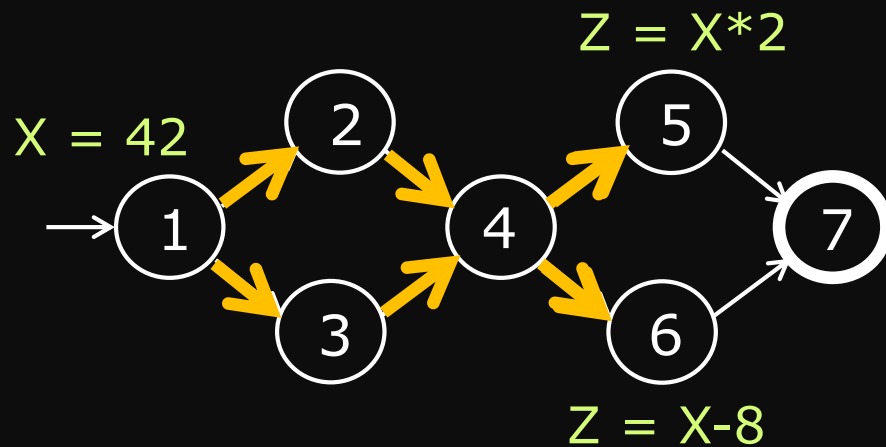
**Reach:** If there is a def-clear path from  $l_i$  to  $l_j$  with respect to  $v$ , the def of  $v$  at  $l_i$  reaches the use at  $l_j$

# DU-Paths

<b>du-path</b>	A simple subpath that is def-clear with respect to a variable $v$ from a def of $v$ to a use of $v$
<b><math>\text{du}(n_i, n_j, v)</math></b>	The set of du-paths from $n_i$ to $n_j$
<b><math>\text{du}(n_i, v)</math></b>	The set of du-paths that start at $n_i$

Keep the path simple to ensure a reasonably small number of paths

# Example: DU-Paths



**du-paths:** for variable  $X$

- $du(1, 5, X) = \{[1,2,4,5], [1,3,4,5]\}$
- $du(1, 6, X) = \{[1,2,4,6], [1,3,4,6]\}$

**defs:**

- $def(1) = \{ X \}$
- $def(5) = \{ Z \}$
- $def(6) = \{ Z \}$

**uses:**

- $use(5) = \{ X \}$
- $use(6) = \{ X \}$

**du-pairs:** for variable  $X$

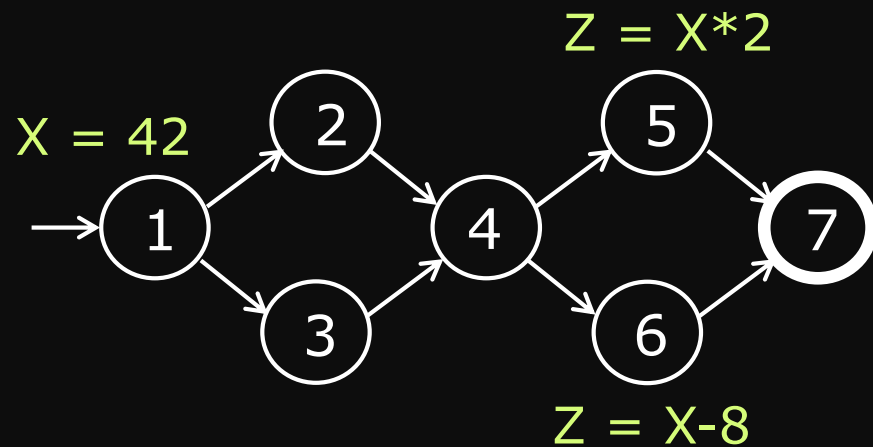
- $(1, 5)$
- $(1, 6)$

# Categorizing DU-Paths

- The core of data flow testing – allowing definitions to flow to uses
- The test criteria for data flow will be defined as sets of du-paths. Thus, we first categorize the du-paths according to:

<b>def-path set</b> $du(n_i, v)$	All simple paths w.r.t. a given variable $v$ defined in a given node
<b>def-pair set</b> $du(n_i, n_j, v)$	All simple paths w.r.t. a given variable $v$ from a given definition ( $n_i$ ) to a given use ( $n_j$ )

# Example: Def-Path and Def-Pair



## du-path sets

- $du(1, X) = \{[1,2,4,5], [1,3,4,5], [1,2,4,6], [1,3,4,6]\}$

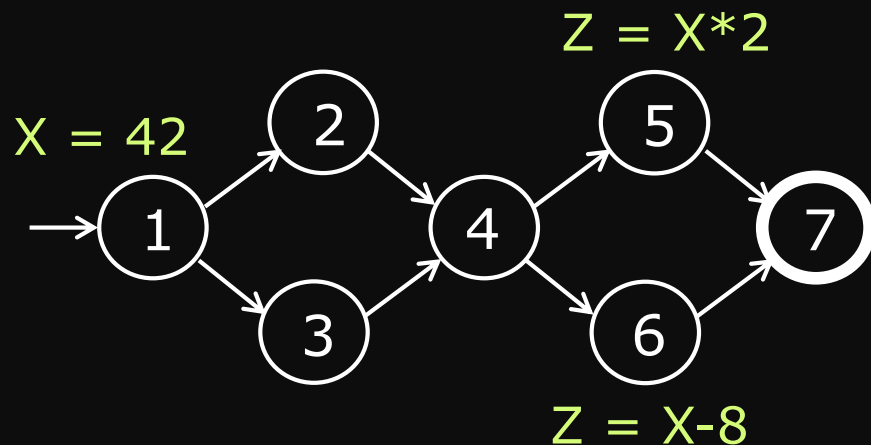
## du-pair sets

- $du(1, 5, X) = \{[1,2,4,5], [1,3,4,5]\}$
- $du(1, 6, X) = \{[1,2,4,6], [1,3,4,6]\}$

# 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



Test path  $[1,2,4,5,7]$  du-tours  
du-path  $[1,2,4,5]$

## du-path sets

- $du(1, X) = \{[1,2,4,5], [1,3,4,5], [1,2,4,6], [1,3,4,6]\}$

## du-pair sets

- $du(1, 5, X) = \{[1,2,4,5], [1,3,4,5]\}$
- $du(1, 6, X) = \{[1,2,4,6], [1,3,4,6]\}$

# Data Flow Coverage Criteria

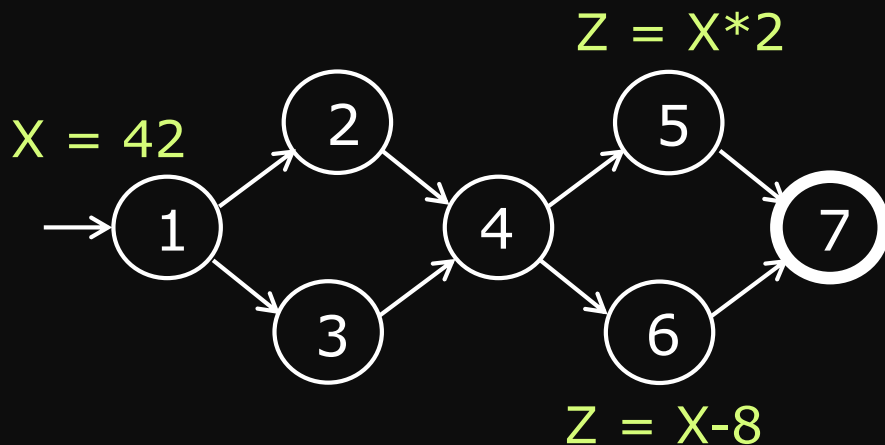
---

- All-Defs Coverage (ADC) – Cover every def
- All-Uses Coverage (AUC) – For each def, get to every use
- All-du-Paths Coverage (ADUPC) – Follow all du-paths

# All-Defs Coverage (ADC)

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

- For each def, **at least one use** must be reached



## du-path sets

- $\text{du}(1, X) = \{[1, 2, 4, 5], [1, 3, 4, 5], [1, 2, 4, 6], [1, 3, 4, 6]\}$

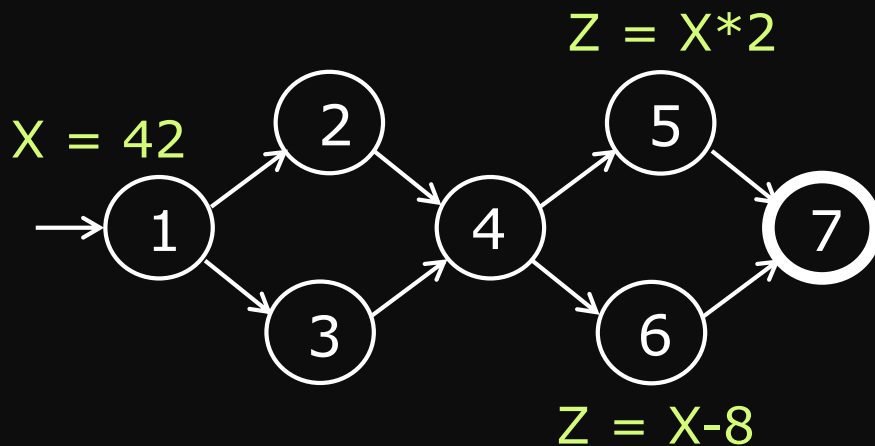
TR for  $X = \{[1, 2, 4, 5]\}$

Test paths =  $\{[1, 2, 4, 5, 7]\}$

# All-Uses Coverage (AUC)

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

- For each def, **all uses** must be reached



## du-pair sets

- $\text{du}(1, 5, X) = \{[1, 2, 4, 5], [1, 3, 4, 5]\}$
- $\text{du}(1, 6, X) = \{[1, 2, 4, 6], [1, 3, 4, 6]\}$

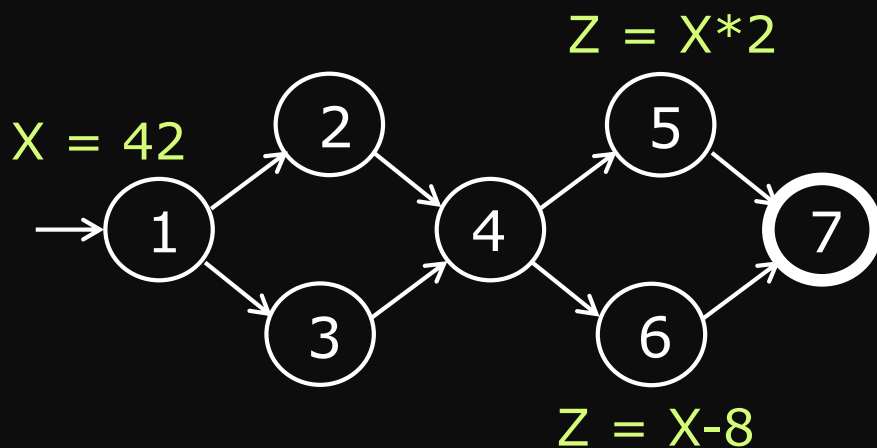
TR for  $X = \{[1, 2, 4, 5], [1, 2, 4, 6]\}$

Test paths =  $\{[1, 2, 4, 5, 7], [1, 2, 4, 6, 7]\}$

# All-DU-Paths Coverage (ADUPC)

For each set of du-paths  $S = \text{du}(n_i, n_j, v)$ , TR contains every path  $d$  in  $S$

- For each def-use pair, **all paths** between defs and uses must be covered



## du-pair sets

- $\text{du}(1, 5, X) = \{[1, 2, 4, 5], [1, 3, 4, 5]\}$
- $\text{du}(1, 6, X) = \{[1, 2, 4, 6], [1, 3, 4, 6]\}$

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

Test paths =  $\{[1, 2, 4, 5, 7], [1, 3, 4, 5, 7], [1, 2, 4, 6, 7], [1, 3, 4, 6, 7]\}$

# Summary

- Graphs are very powerful abstraction for designing tests
- Graphs appear in many situations in software
- Each criterion has its own cost/benefit tradeoffs
  - No silver bullet
  - When possible, choose the criterion that yields the smallest number of test requirements while maintaining fault detection capability

# Graph Coverage Criteria Subsumption

