WELCOME TO
SOFTWARE ENGINEERING TESTING!

(DARK MODE)

Lecture 3: Graphs I

We will be discussing:

- Graph and testing terminology... a lot of it
- Structural Coverage for Graphs

We'll get started shortly after 10am to give folks a chance to join.

Curtin University

LECTURE 3: GRAPHS (PART 1)

Software Engineering Testing (CMPE3008/CMPE4001/CMPE5000)

Created by Arlen Brower

OUTCOMES

You should be able to do the following:

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
  - Node & Edge Coverage

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
  - Node & Edge Coverage
  - Prime Path Coverage

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
  - Node & Edge Coverage
  - Prime Path Coverage
  - All-DU-Paths, All-Uses, All-defs coverage

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
  - Node & Edge Coverage
  - Prime Path Coverage
  - All-DU-Paths, All-Uses, All-defs coverage
- Identify test paths for the above

DEFINITION OF A GRAPH

Graphs are effectively made up of sets.

DEFINITION OF A GRAPH

Graphs are effectively made up of sets.

- A set of nodes

DEFINITION OF A GRAPH

Graphs are effectively made up of sets.

- A set of nodes
- A set of initial nodes

DEFINITION OF A GRAPH

Graphs are effectively made up of sets.

- A set of nodes
- A set of initial nodes
- A set of final nodes

DEFINITION OF A GRAPH

Graphs are effectively made up of sets.

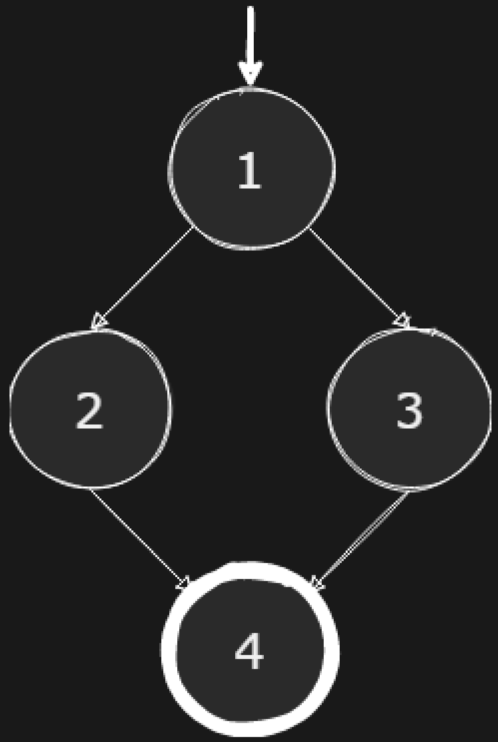
- A set of nodes
- A set of initial nodes
- A set of final nodes
- A set of edges

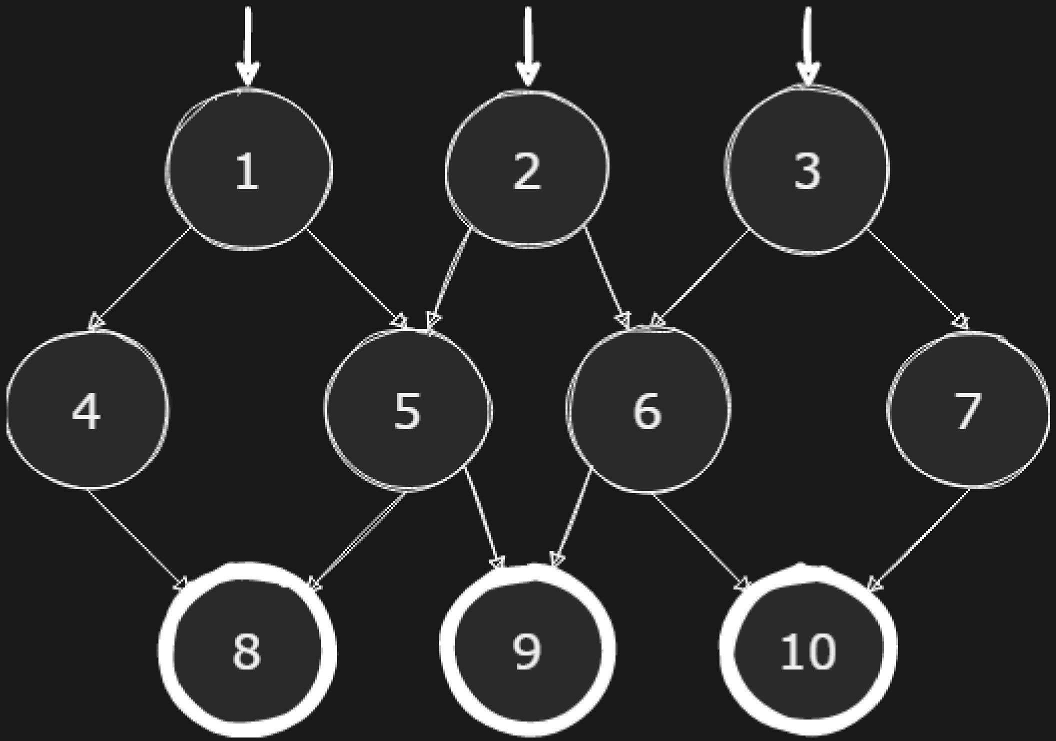
DEFINITION OF A GRAPH

Graphs are effectively made up of sets.

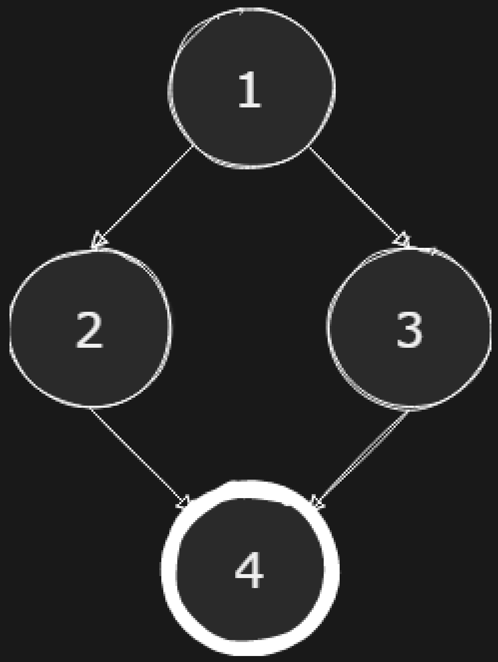
- A set of nodes
- A set of initial nodes
- A set of final nodes
- A set of edges

These sets must not be empty.


1
2
3
4

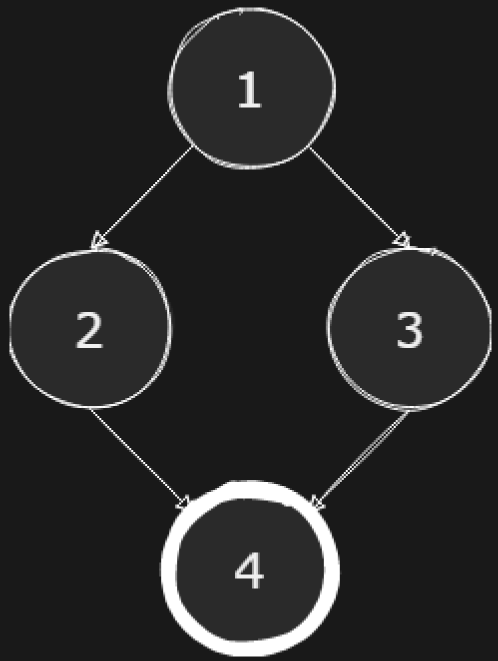

1
2
3
4
5
6
7
8
9
10

NOT A VALID GRAPH



Why?

NOT A VALID GRAPH



Why?

$$N_0 = \{\}$$

GRAPH TERMINOLOGY

GRAPH TERMINOLOGY

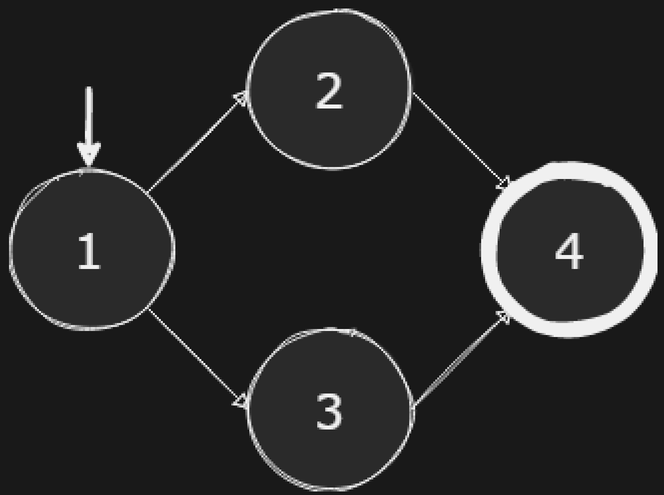
- Path: A sequence of nodes
- Length: The number of edges in a path
- Subpath: A path that makes up part of a larger path

GRAPH TERMINOLOGY

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

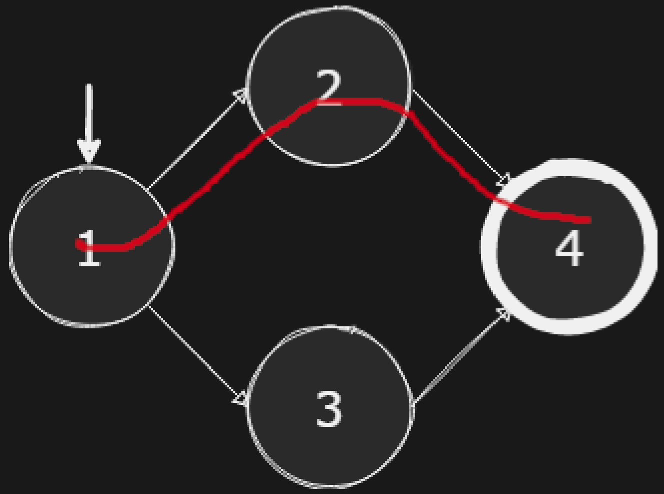
GRAPH TERMINOLOGY

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



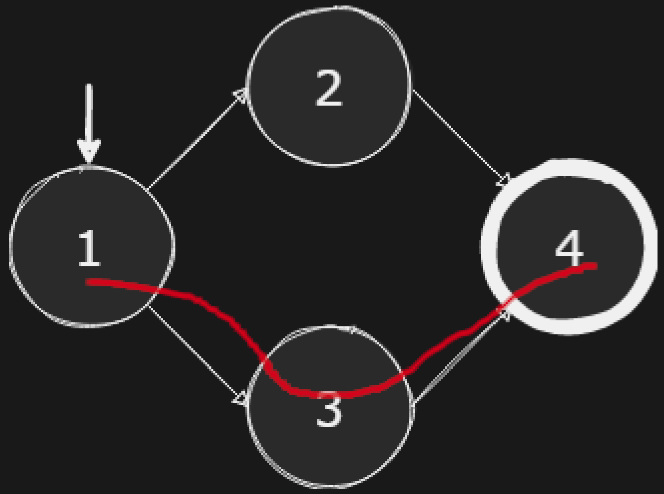
GRAPH TERMINOLOGY

- Test Path: A path that starts at an initial node and ends at a final node
- (1,2,4) is a valid test path.



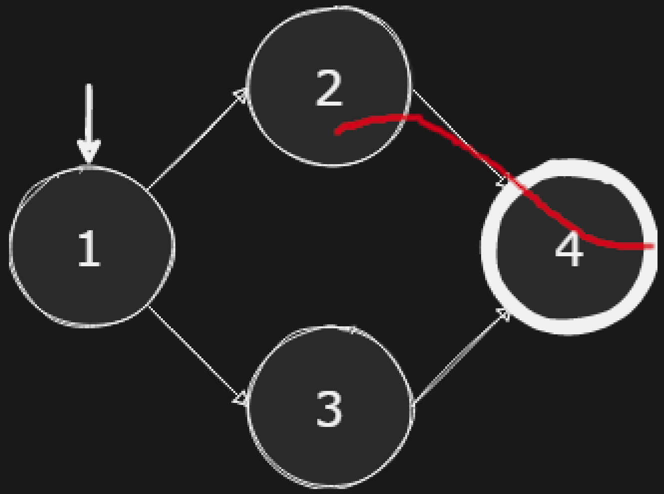
GRAPH TERMINOLOGY

- Test Path: A path that starts at an initial node and ends at a final node
- (1,3,4) is a valid test path.



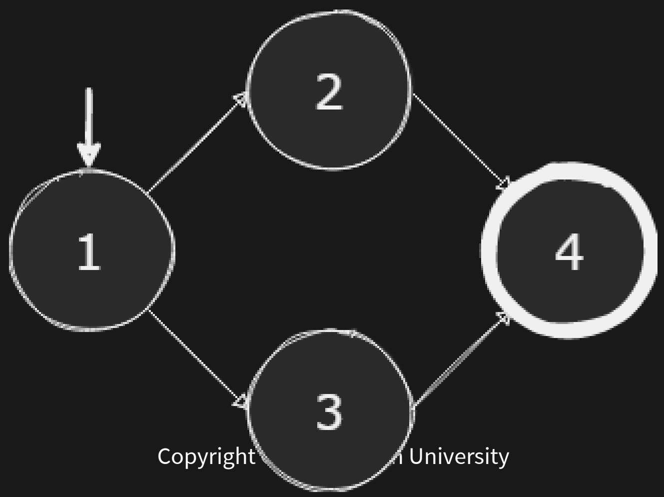
GRAPH TERMINOLOGY

- Test Path: A path that starts at an initial node and ends at a final node
- (2,4) is NOT a valid test path.



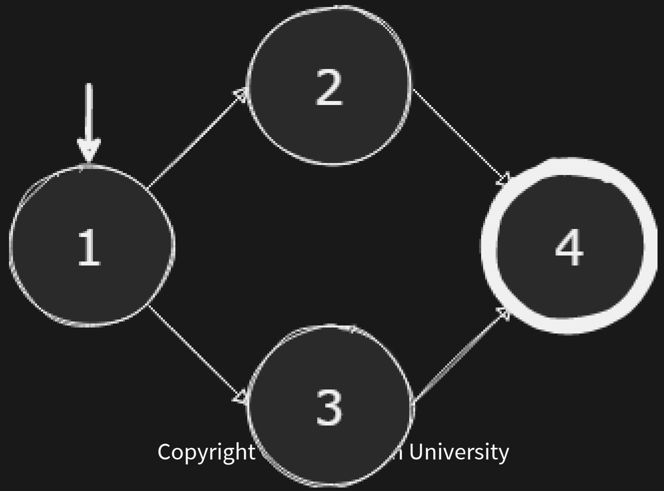
GRAPH TERMINOLOGY

- Test Path: A path that starts at an initial node and ends at a final node
- Whenever you execute your code, it will run through a test path.

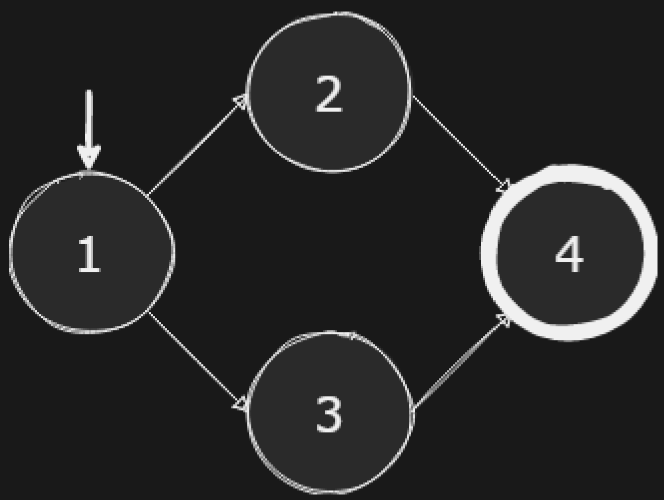


GRAPH TERMINOLOGY

- <u>Test Path</u>: A path that starts at an initial node and ends at a final node
- Whenever you execute your code, it will run through a test path.
- A single test path may satisfy multiple test requirements.

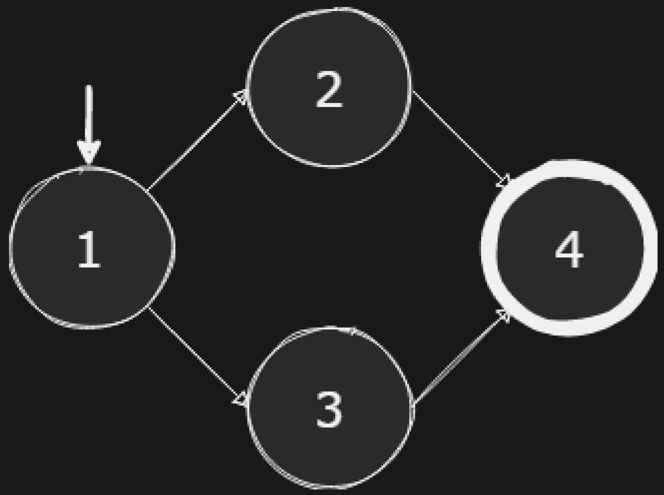


GRAPH TERMINOLOGY



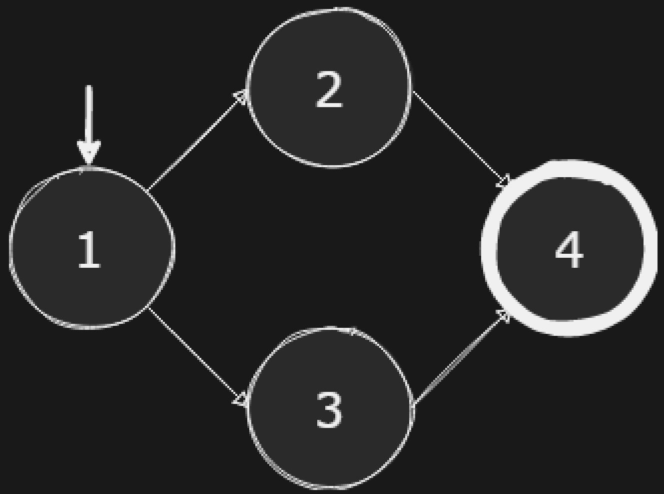
GRAPH TERMINOLOGY

- Visit: If node/edge N exists in a test path, that test visits that node/edge



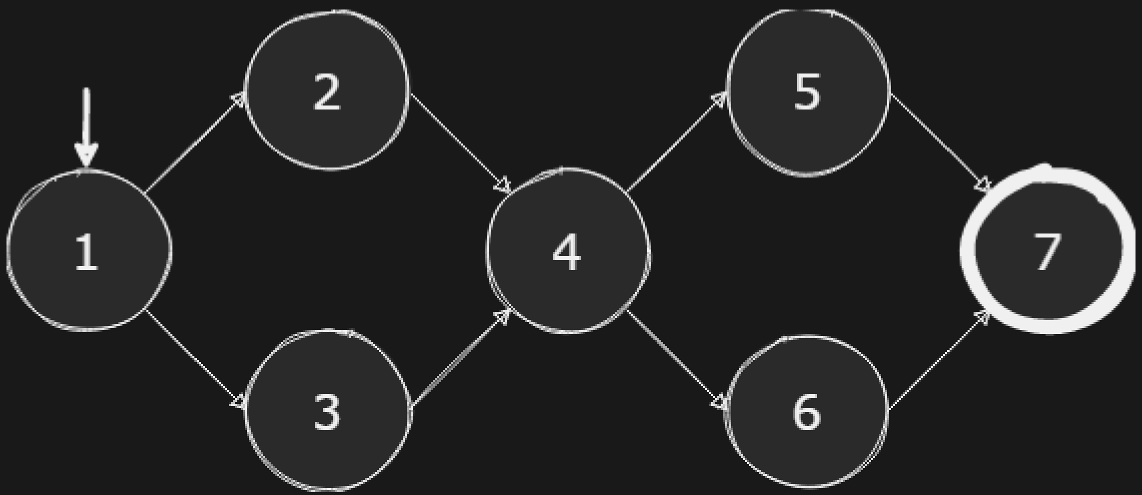
GRAPH TERMINOLOGY

- Visit: If node/edge N exists in a test path, that test visits that node/edge
- Tour:A subpath that is part of a test path is toured by that test path.



GRAPH TERMINOLOGY

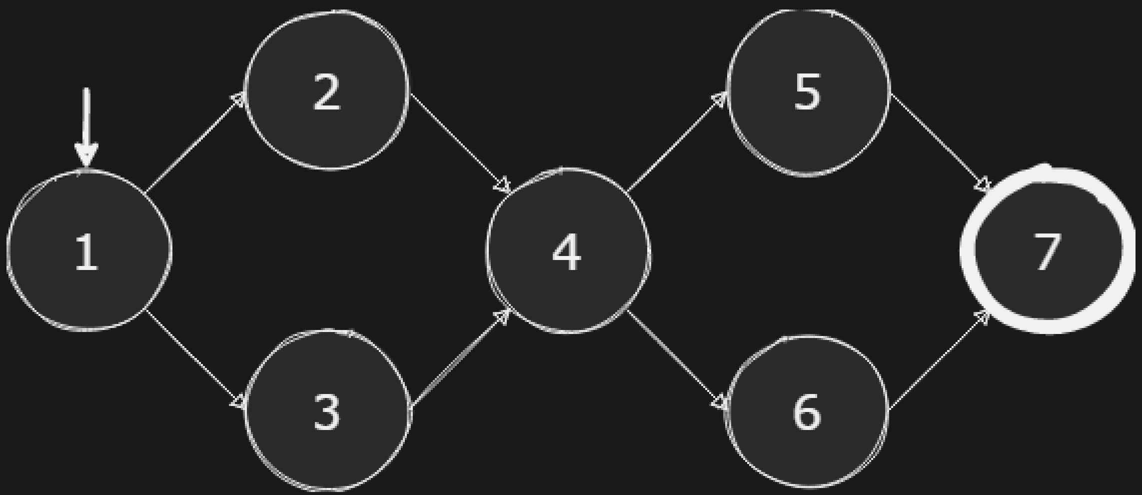
Test Path [1,2,4,5,7]



GRAPH TERMINOLOGY

Test Path [1,2,4,5,7]

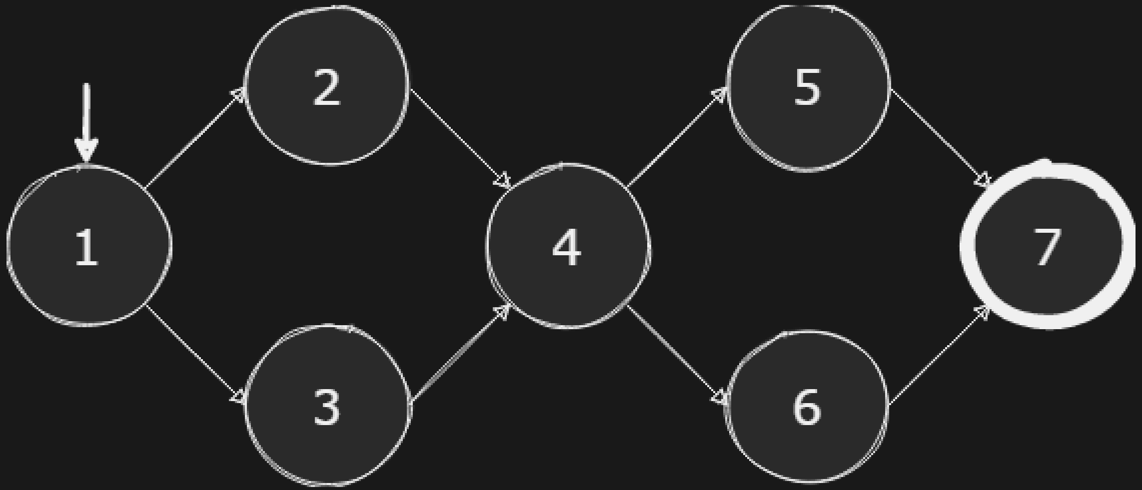
- Visits nodes 1,2,4,5,7



GRAPH TERMINOLOGY

Test Path [1,2,4,5,7]

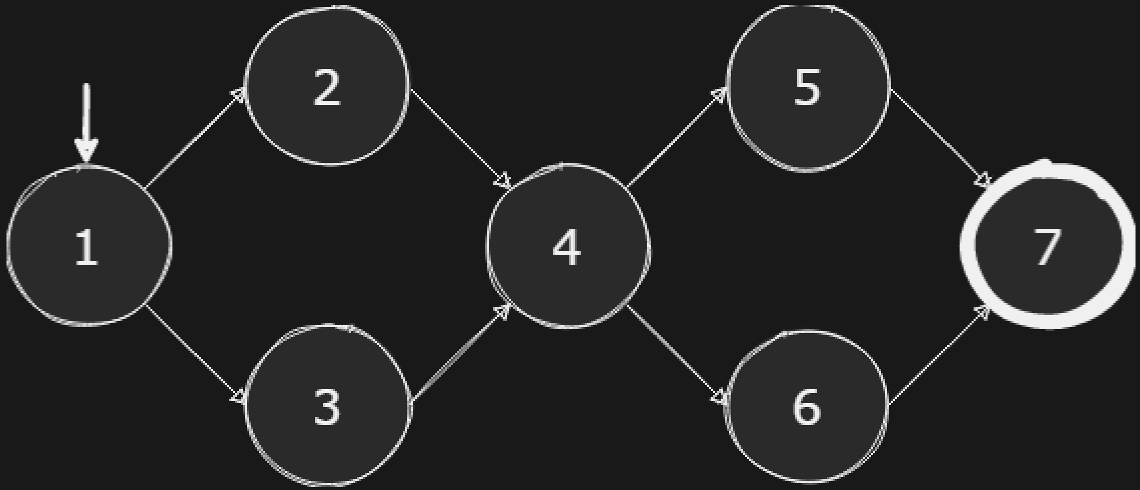
- Visits nodes 1,2,4,5,7
- Visits edges (1,2) (2,4) (4,5) (5,7)



GRAPH TERMINOLOGY

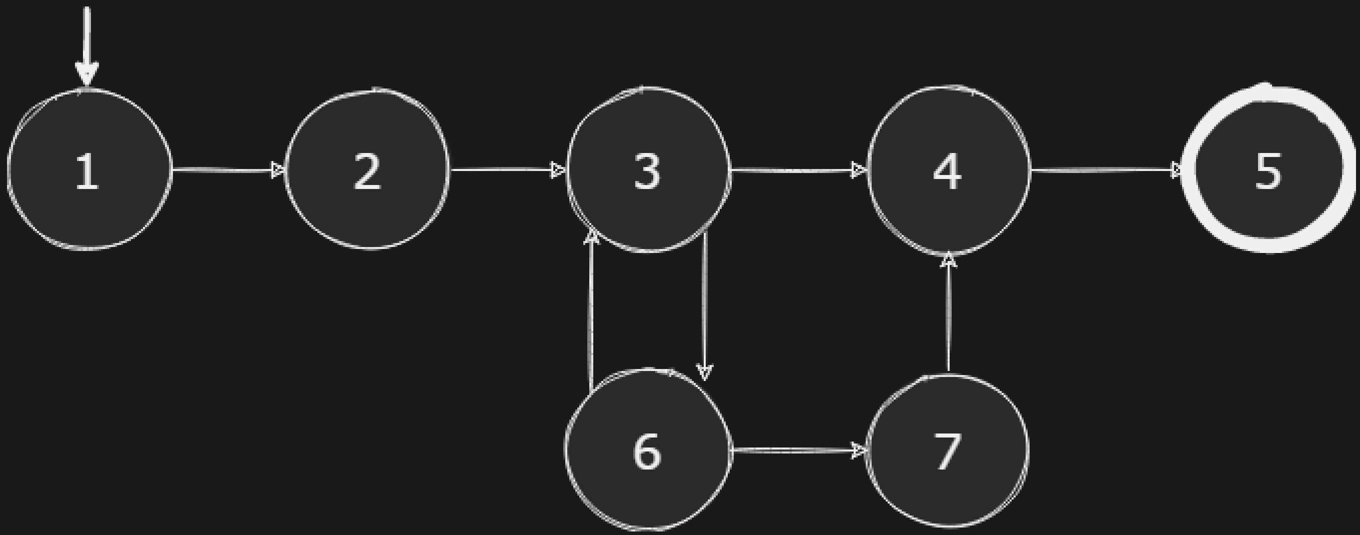
Test Path [1,2,4,5,7]

- 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) (2,4,5,7)



TOURING, SIDE-TRIPS, DETOURS

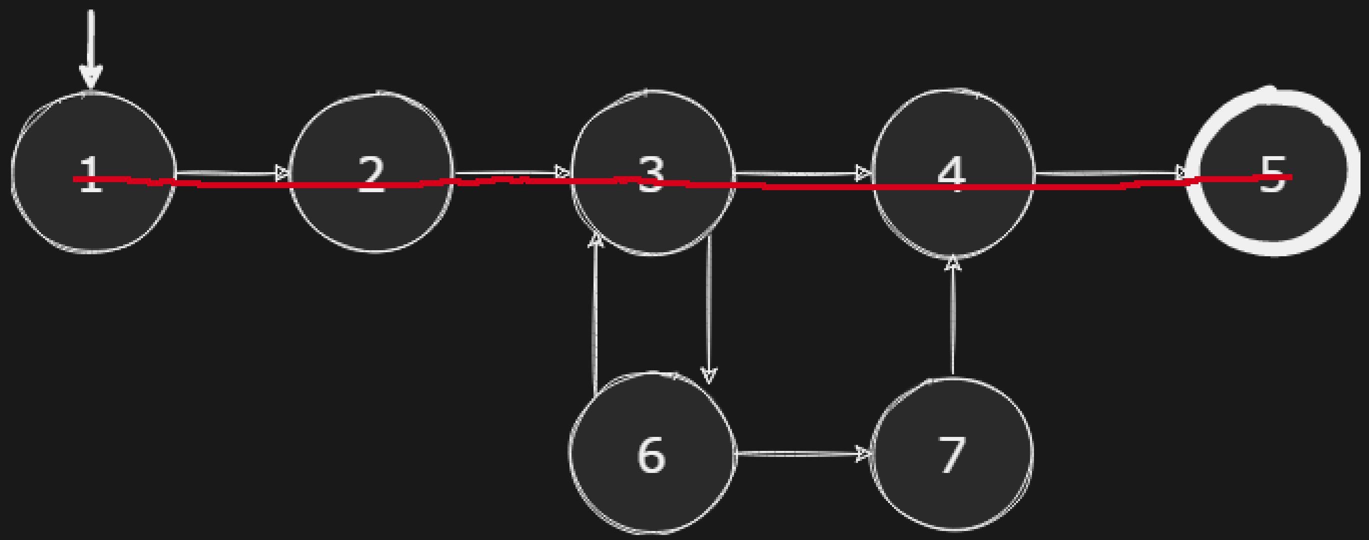
Let's say we have a test requirement: [1,2,3,4,5]



TOURING, SIDE-TRIPS, DETOURS

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

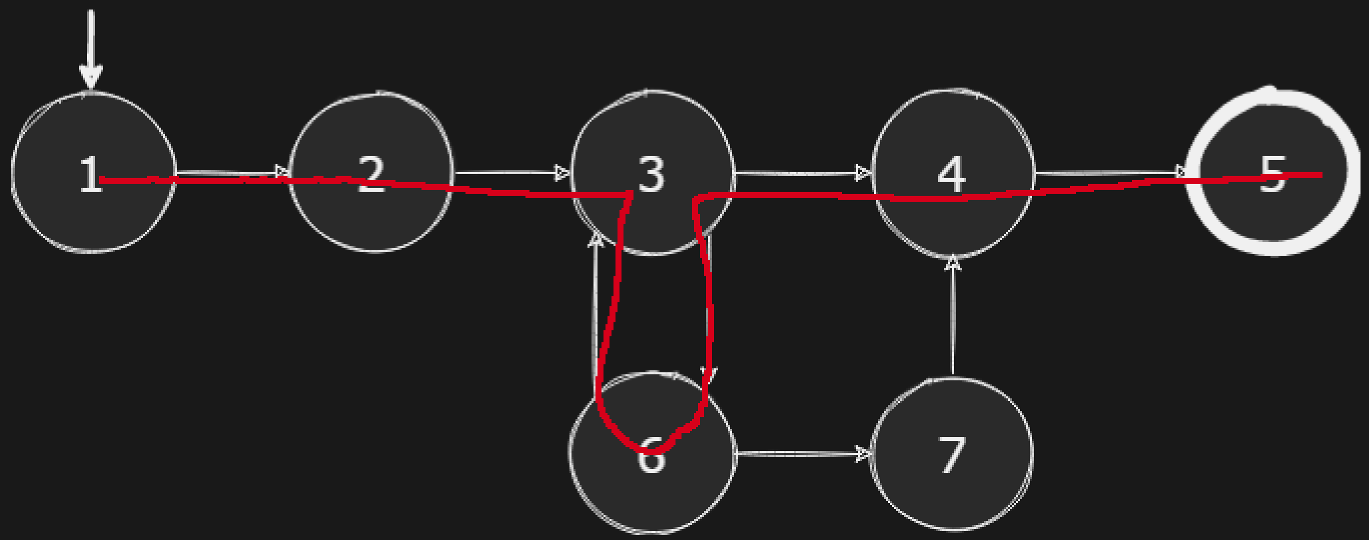
If our test path was also [1,2,3,4,5], it <u>tours</u> our requirement.



TOURING, SIDE-TRIPS, DETOURS

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

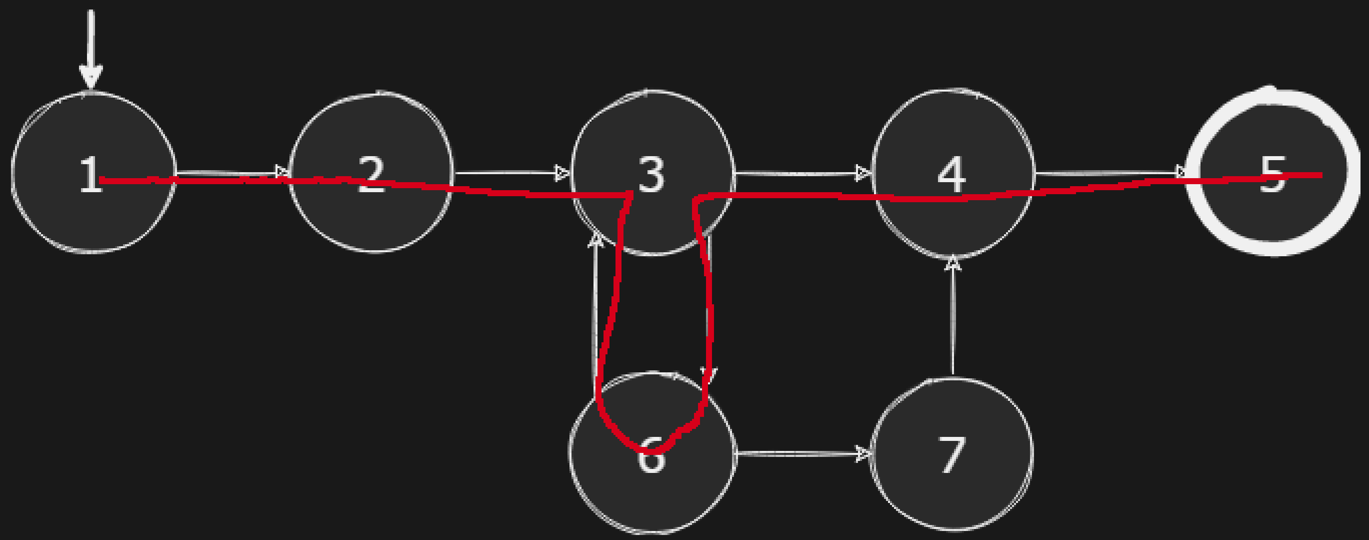
If our test path was also [1,2,3,6,3,4,5], it tours our requirement <u>with a side trip</u>.



TOURING, SIDE-TRIPS, DETOURS

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

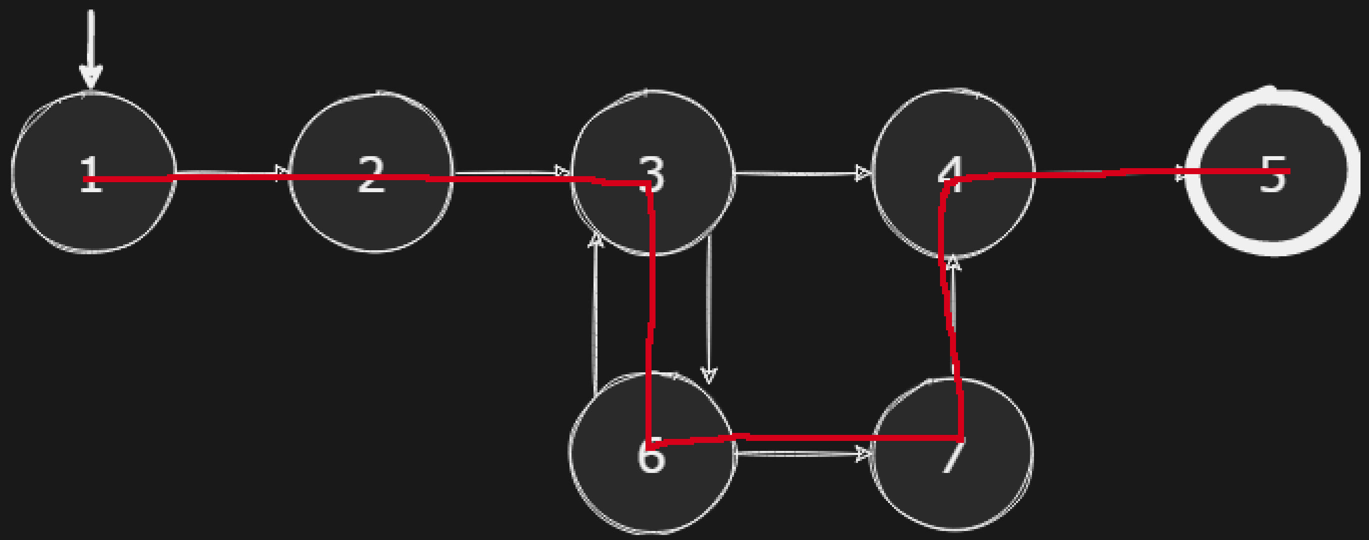
If our test path was also [1,2,3,6,3,4,5], it tours our requirement with a side trip.



TOURING, SIDE-TRIPS, DETOURS

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

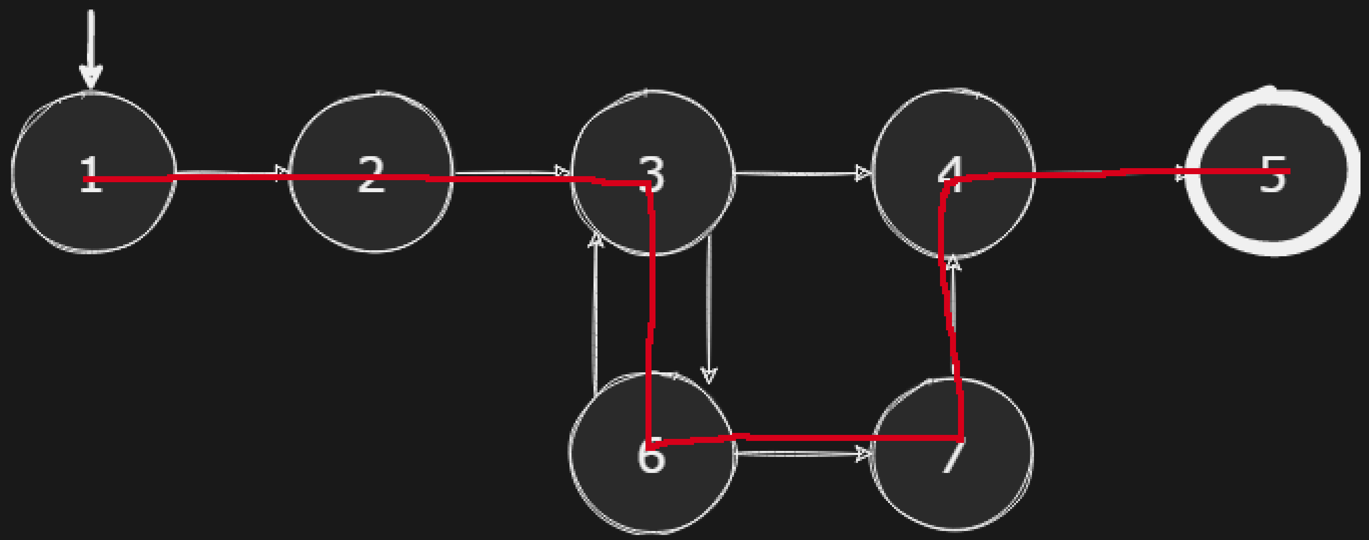
If our test path was also [1,2,3,6,7,4,5], it tours our requirement <u>with a detour</u>.



TOURING, SIDE-TRIPS, DETOURS

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

If our test path was also [1,2,3,6,7,4,5], it tours our requirement with a detour.



MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

- Lower case (i.e. 't') denotes an element of a set.

MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

- Lower case (i.e. 't') denotes an element of a set.
- Upper case (i.e. 'T') denotes an entire set.

MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

- Path (t): A test path executed by test t

MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

- Path (t): A test path executed by test t
- Path (T): The set of test paths executed by test set T

MORE GRAPH TERMINOLOGY

We tend to run into a lot of set notation when discussing graphs.

- Path (t): A test path executed by test t
- Path (T): The set of test paths executed by test set T
- Each test executes one and only one test path

SOME TESTING TERMINOLOGY

SOME TESTING TERMINOLOGY

- Test Criterion: The rules by which we design our tests

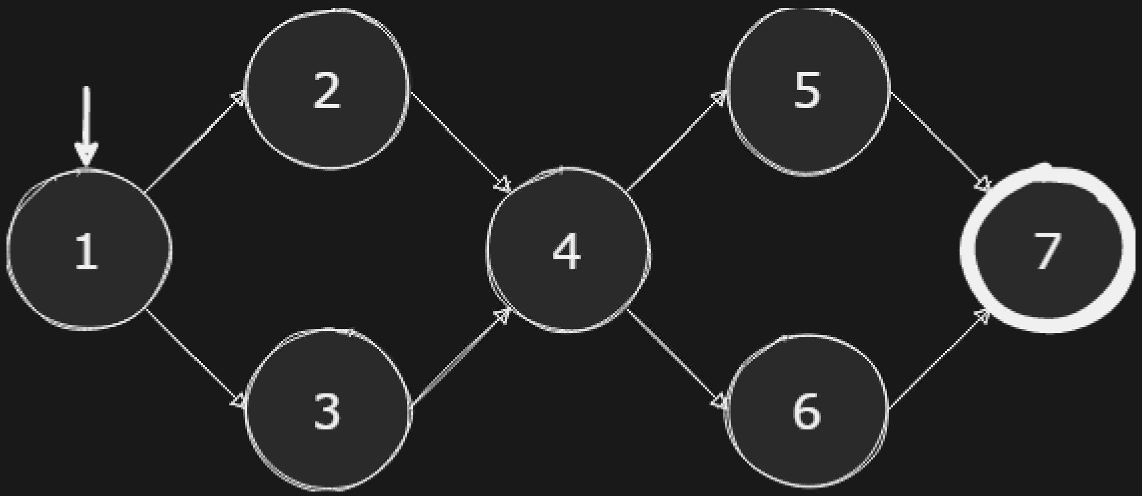
SOME TESTING TERMINOLOGY

- Test Criterion: The rules by which we design our tests
- Test Requirements (TR): A set of paths that test paths should ideally cover, based on a criterion

SOME TESTING TERMINOLOGY

Let's consider a simple criterion:

- Test Criterion: Node Coverage
- Test Requirements: [1],[2],[3],[4],[5],[6],[7]



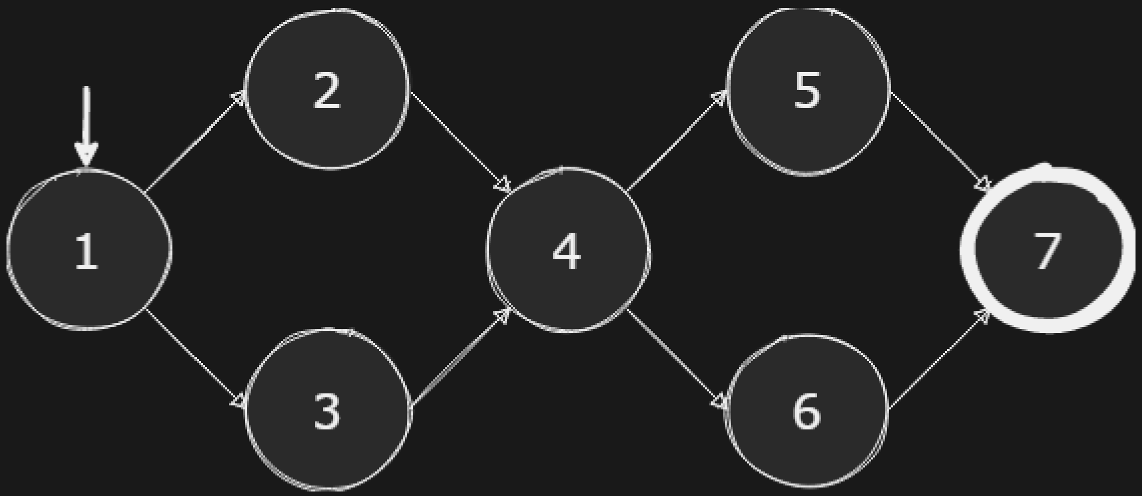
SOME TESTING TERMINOLOGY

Formal...

SOME TESTING TERMINOLOGY

Less formal

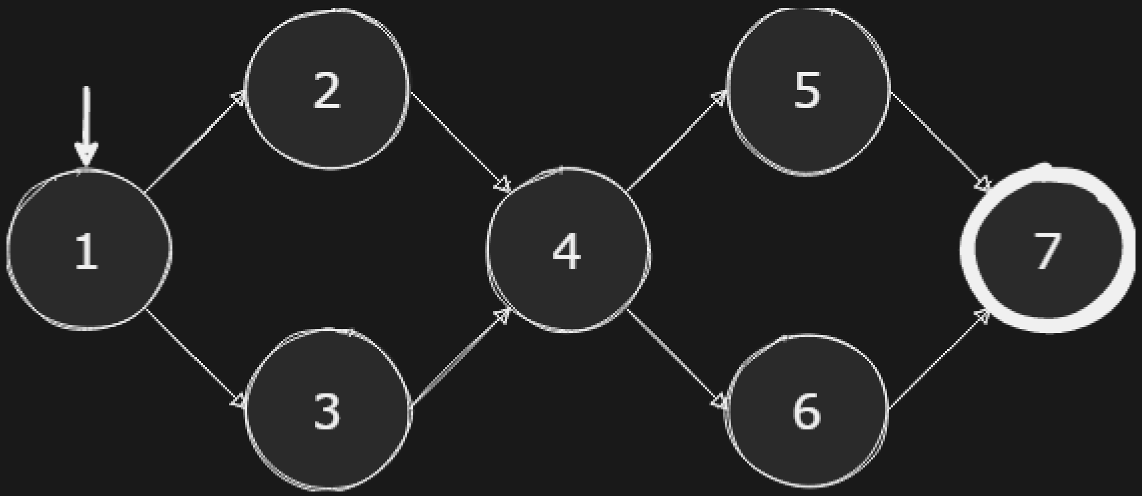
- Satisfaction: Make sure you have test paths that cover all the test requirements for a given criteria.



SOME TESTING TERMINOLOGY

Node Coverage Satisfaction

- Test Requirements: [1],[2],[3],[4],[5],[6],[7]
- Path(T): [1,2,4,6,7], [1,3,4,5,7]



TYPES OF CRITERION FOR GRAPHS

In general, we have two categories for Graph Coverage:

- Structural: criteria that investigate the structure of the graph and the paths
- Data Flow: criteria that investigate the variables and how data is actually used

TYPES OF CRITERION FOR GRAPHS

In general, we have two categories for Graph Coverage:

- Structural: criteria that investigate the structure of the graph and the paths
- Data Flow: criteria that investigate the variables and how data is actually used

Today we're looking at Structural Criteria.

NODE COVERAGE

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*.

NODE COVERAGE

... Make sure that your Test Requirements contain all reachable nodes for a given graph.

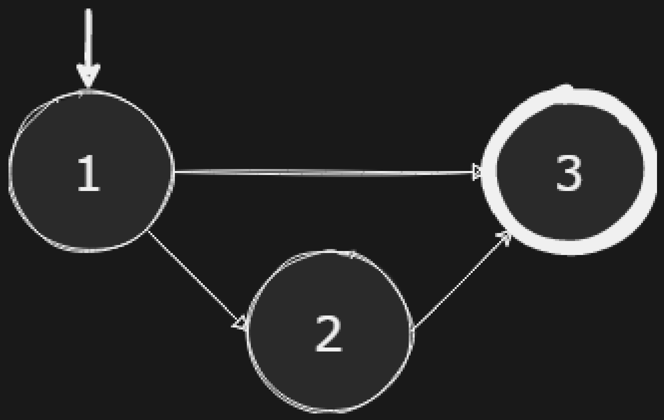
EDGE COVERAGE

TR contains each reachable path of length up to 1, inclusive, in a given graph.

We say "up to 1" to account for graphs with only one node.

EDGE COVERAGE

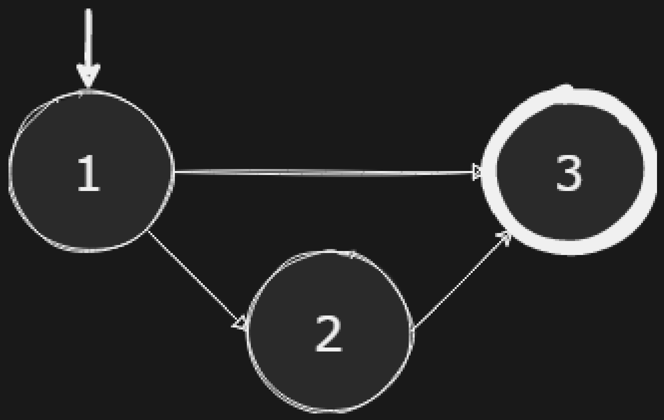
TR contains each reachable path of length up to 1, inclusive, in a given graph.



- TR: [1,3], [1,2], [2,3]
- Path(T): [1,3], [1,2,3]

EDGE-PAIR COVERAGE

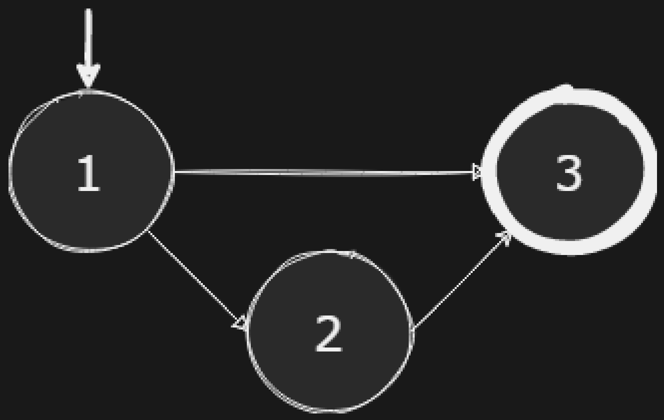
TR contains each reachable path of length up to 2, inclusive, in G.



- TR: [1,3], [1,2,3]
- Path(T): [1,3], [1,2,3]

EDGE-PAIR COVERAGE

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



- TR: [1,3], [1,2,3]
- Path(T): [1,3], [1,2,3]

COMPLETE PATH COVERAGE

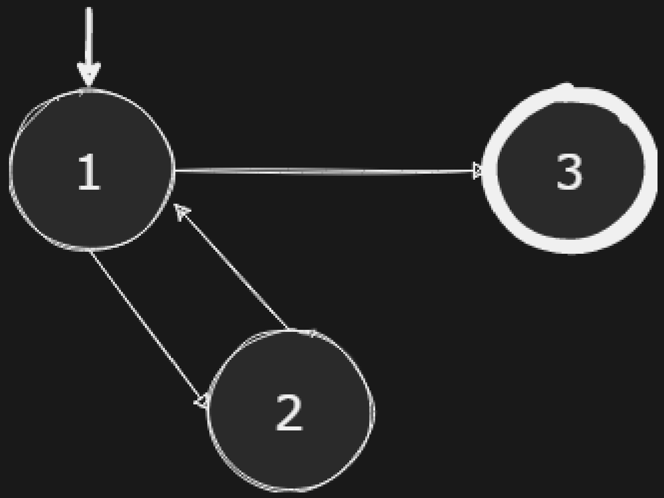
TR contains all paths in G.

COMPLETE PATH COVERAGE

TR contains all paths in G.

COMPLETE PATH COVERAGE

TR contains all paths in G.



SPECIFIED PATH COVERAGE

TR contains a set S of test paths, where S is supplied as a parameter.

SPECIFIED PATH COVERAGE

TR contains a set S of test paths, where S is supplied as a parameter.

- Unfortunately, this is very subjective.

SO... LOOPS.

How do we deal with them?

1. Specify a specific path? Not formal...
2. Go through each loop exactly once? More formal, but not so useful.
3. Go through a loop zero times, one time, more than once? Not formal...

SIMPLE PATHS & PRIME PATHS

SIMPLE PATHS & PRIME PATHS

- Simple Path: A path with *no duplicate nodes* except (maybe!) for the first and last nodes.

SIMPLE PATHS & PRIME PATHS

- Simple Path: A path with *no duplicate nodes* except (maybe!) for the first and last nodes.
- Valid simple paths: [1,2,3], [1,2,1], [1,2,3,4,5,1]

SIMPLE PATHS & PRIME PATHS

- <u>Simple Path</u>: A path with *no duplicate nodes* except (maybe!) for the first and last nodes.
- Valid simple paths: [1,2,3], [1,2,1], [1,2,3,4,5,1]
- Invalid simple paths: [1,2,1,3], [2,1,2,3]

SIMPLE PATHS & PRIME PATHS

- Simple Path: A path with *no duplicate nodes* except (maybe!) for the first and last nodes.
- Valid simple paths: [1,2,3], [1,2,1], [1,2,3,4,5,1]
- Invalid simple paths: [1,2,1,3], [2,1,2,3]

SIMPLE PATHS & PRIME PATHS

- Simple Path: A path with *no duplicate nodes* except (maybe!) for the first and last nodes.
- Valid simple paths: [1,2,3], [1,2,1], [1,2,3,4,5,1]
- Invalid simple paths: [1,2,1,3], [2,1,2,3]

SIMPLE PATHS & PRIME PATHS

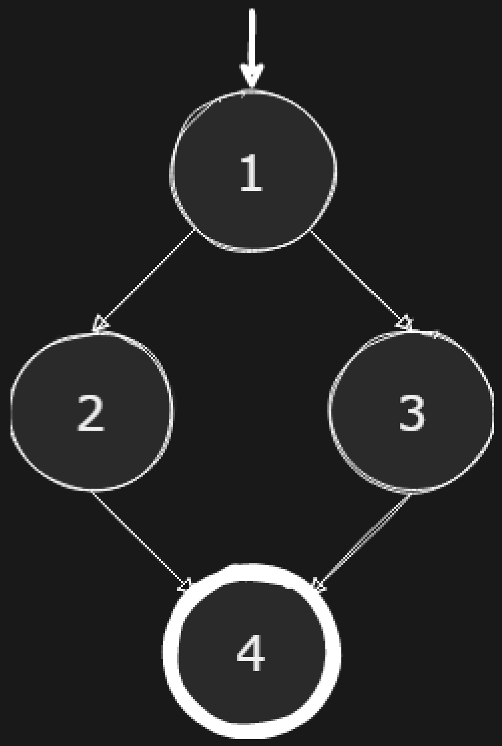
- Prime Path: a **simple path** that is not a subpath of any other simple path.
- Prime paths do not need to be test paths. They can start and end anywhere!

SIMPLE PATHS & PRIME PATHS

- Prime Path: a **simple path** that is not a subpath of any other simple path.
- If it's a subpath, it's not a prime path!
- Prime paths do not need to be test paths. They can start and end anywhere!

PRIME PATHS

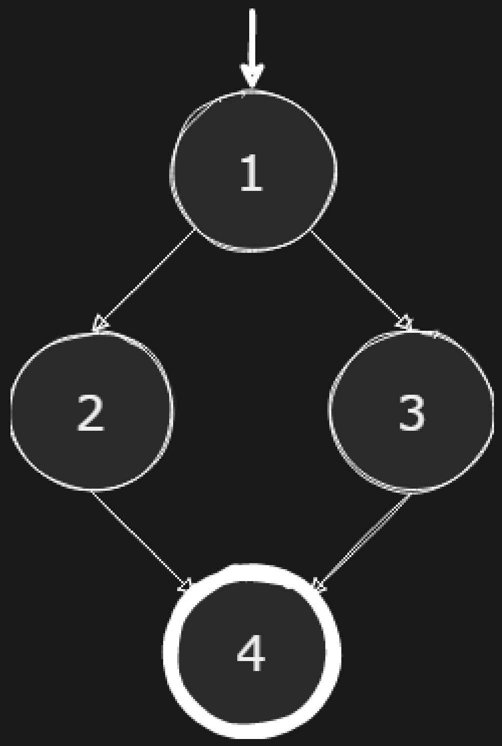
An example...



- Prime Paths: [1,2,4], [1,3,4]

PRIME PATHS

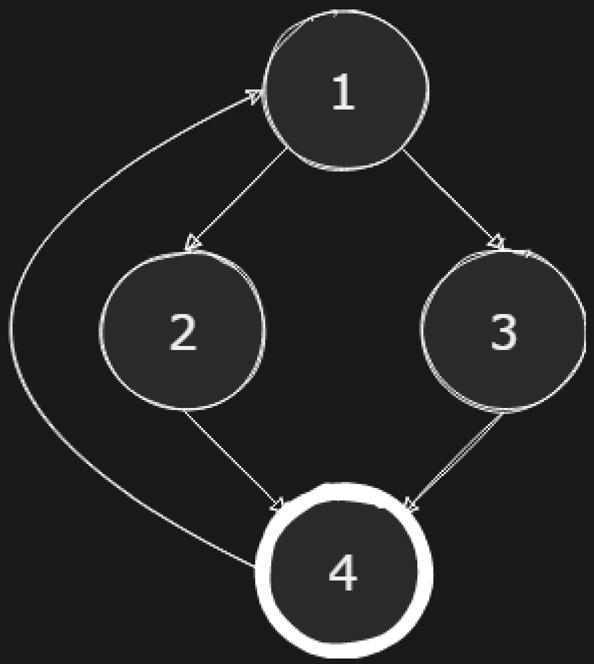
An example...



- Prime Paths: [1,2,4], [1,3,4]
- Not too bad, right?

PRIME PATHS

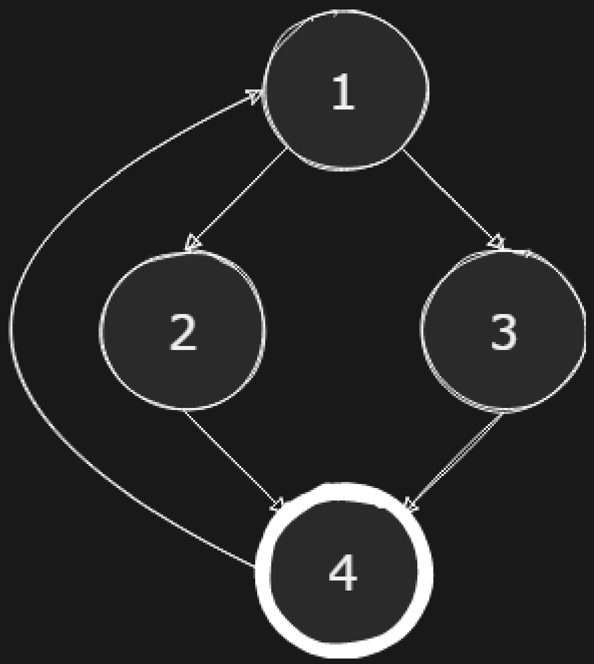
That wasn't even my final form!



- Prime Paths:

PRIME PATHS

That wasn't even my final form!

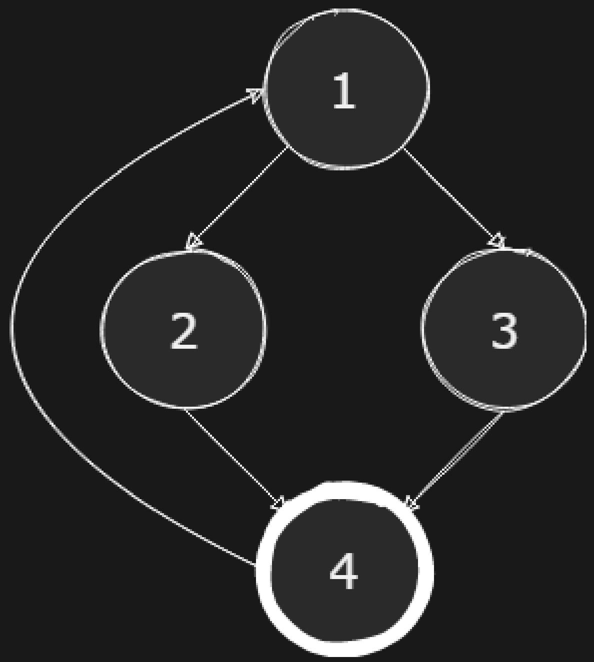


- Prime Paths: ?

PRIME PATHS

That wasn't even my final form!

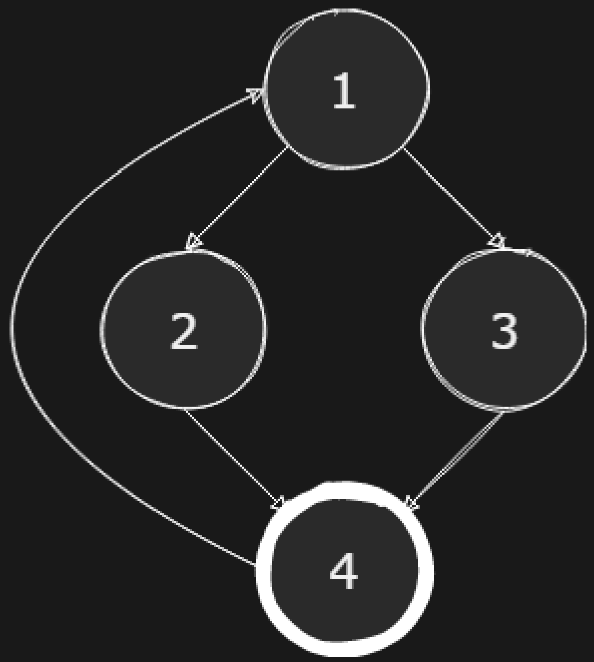
- Prime Paths: ?



PRIME PATHS

That wasn't even my final form!

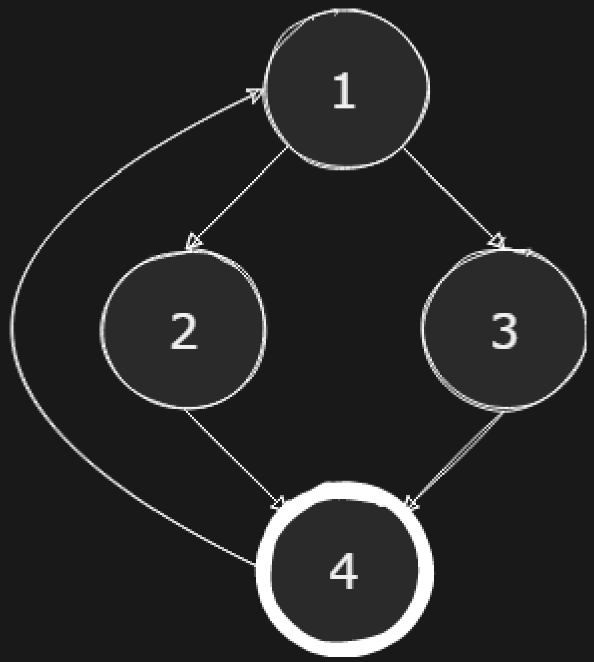
- Prime Paths:



PRIME PATHS

That wasn't even my final form!

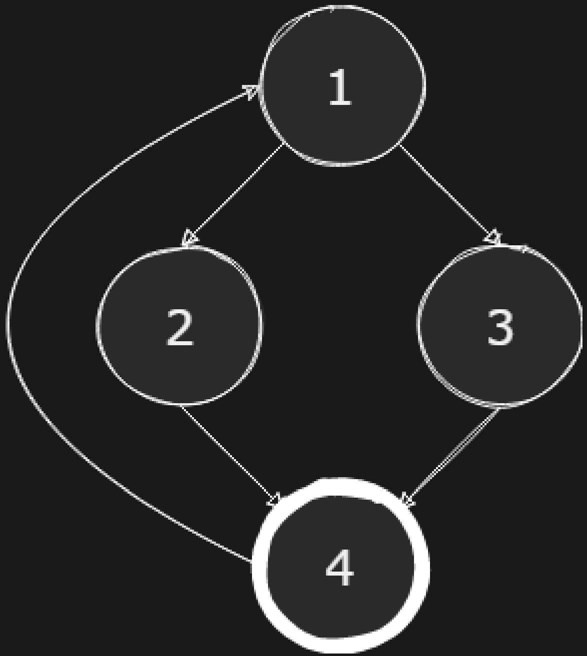
- Prime Paths: There are eight of them now.



PRIME PATHS

That wasn't even my final form!

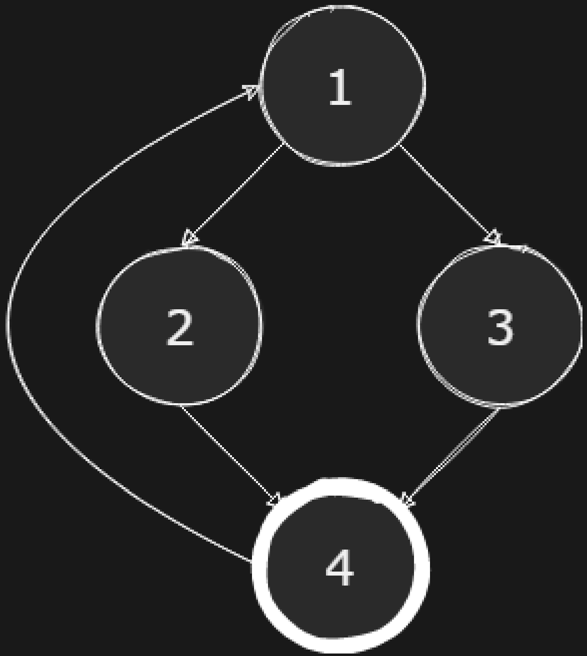
- Prime Paths:



PRIME PATHS

That wasn't even my final form!

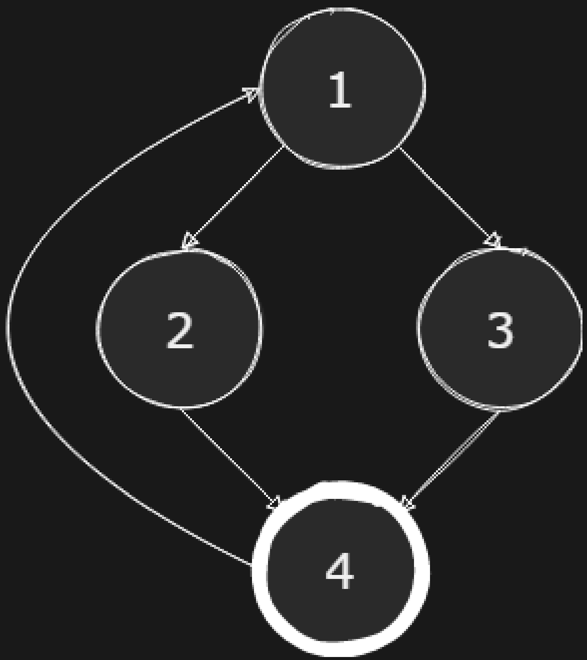
- Prime Paths: [1 2 4 1]



PRIME PATHS

That wasn't even my final form!

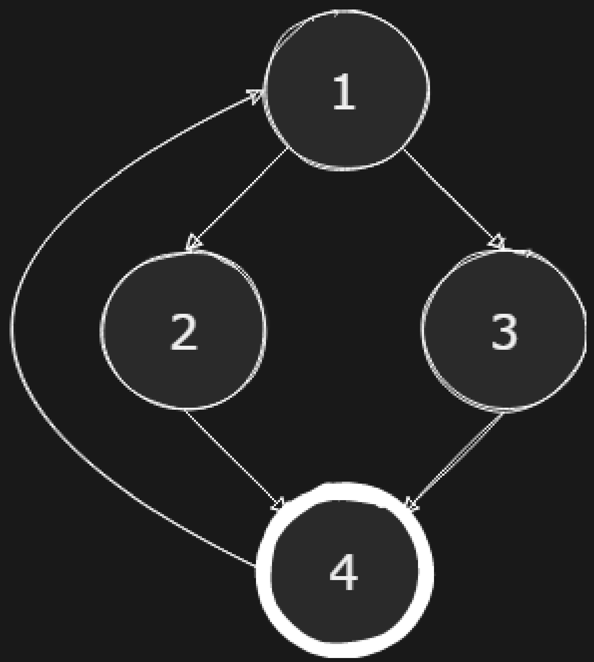
- Prime Paths: [1 2 4 1] [1 3 4 1]



PRIME PATHS

That wasn't even my final form!

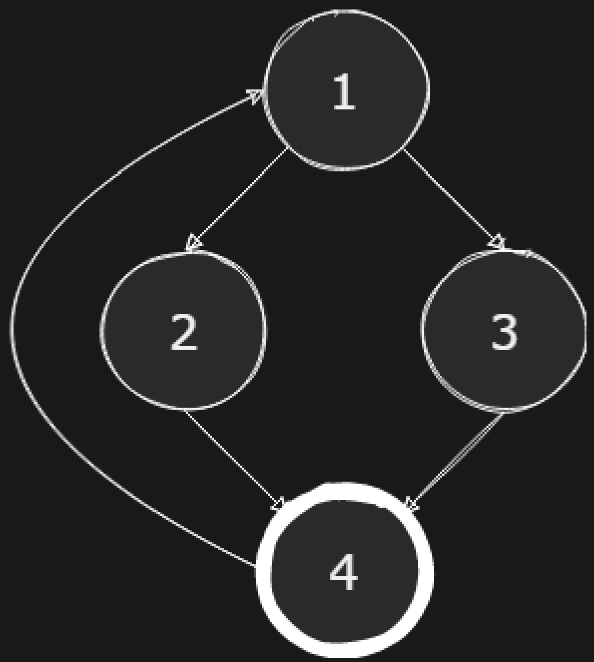
- Prime Paths: [1 2 4 1] [1 3 4 1] [2 4 1 2]



PRIME PATHS

That wasn't even my final form!

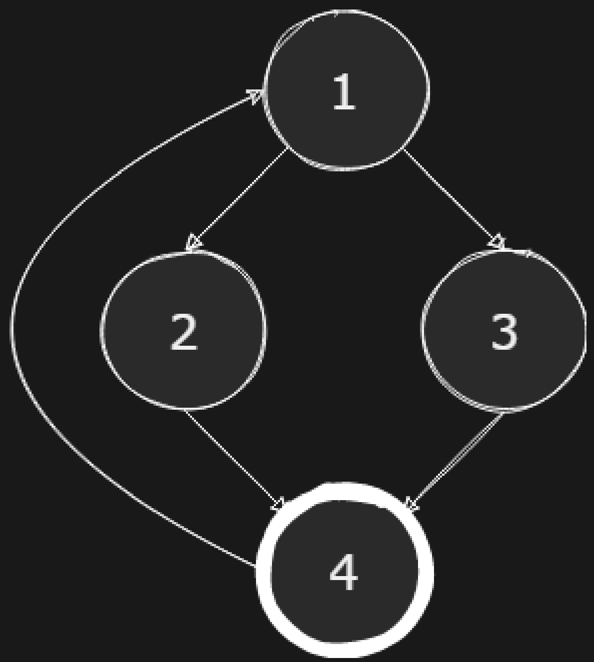
- Prime Paths: [1 2 4 1] [1 3 4 1] [2 4 1 2] [2 4 1 3]



PRIME PATHS

That wasn't even my final form!

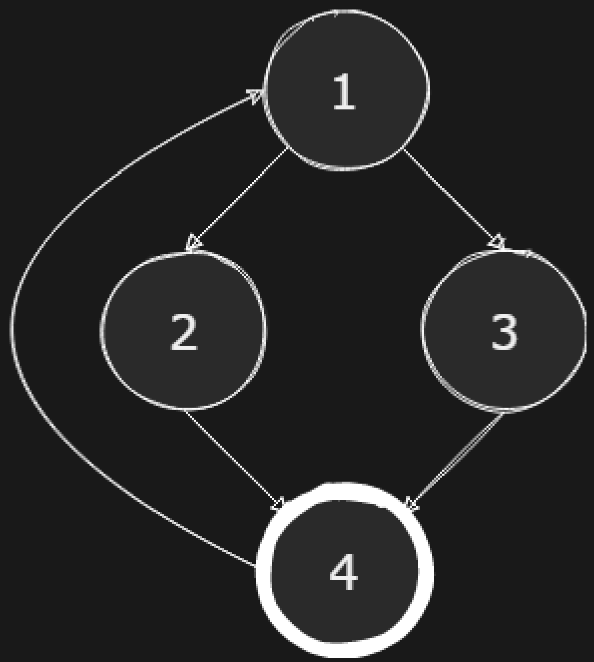
- Prime Paths: [1 2 4 1] [1 3 4 1] [2 4 1 2] [2 4 1 3] [3 4 1 3]



PRIME PATHS

That wasn't even my final form!

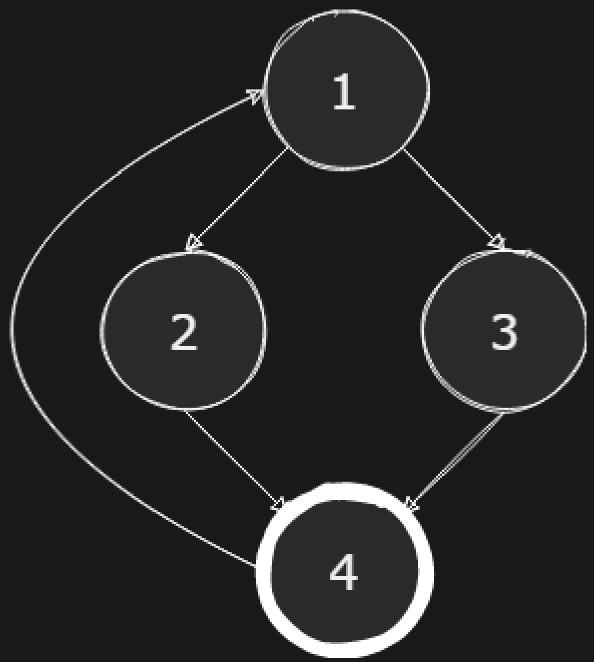
- Prime Paths: [1 2 4 1] [1 3 4 1] [2 4 1 2] [2 4 1 3] [3 4 1 3] [3 4 1 2]



PRIME PATHS

That wasn't even my final form!

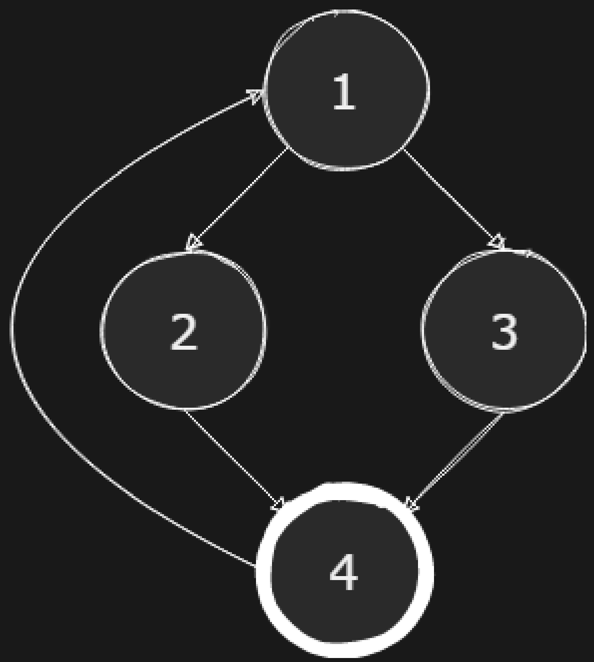
- Prime Paths: [1 2 4 1] [1 3 4 1] [2 4 1 2] [2 4 1 3] [3 4 1 3] [3 4 1 2] [4 1 2 4]



PRIME PATHS

That wasn't even my final form!

- Prime Paths: [1 2 4 1] [1 3 4 1] [2 4 1 2] [2 4 1 3] [3 4 1 3] [3 4 1 2] [4 1 2 4] [4 1 3 4]



PRIME PATH COVERAGE

TR contains each prime path in G

PRIME PATH COVERAGE

TR contains each prime path in G

- This actually subsumes node, edge, and edge-pair coverage.

ROUND TRIPS

A round trip is a prime path that starts and ends on the same node. We can define a few forms of coverage from these.

SIMPLE ROUND TRIP COVERAGE

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

TR contains all round-trip paths for each reachable node in G

INFEASIBLE TEST REQUIREMENTS

Remember side trips and detours?

INFEASIBLE TEST REQUIREMENTS

- Realistically, test criteria will end up having infeasible test requirements.

INFEASIBLE TEST REQUIREMENTS

- Realistically, test criteria will end up having infeasible test requirements.
- Some statements may simply be unreachable

INFEASIBLE TEST REQUIREMENTS

- Realistically, test criteria will end up having infeasible test requirements.
- Some statements may simply be unreachable
- Some subpaths may rely on logical contradictions

INFEASIBLE TEST REQUIREMENTS

- Realistically, test criteria will end up having infeasible test requirements.
- Some statements may simply be unreachable
- Some subpaths may rely on logical contradictions
- Worse still, it's undecidable whether all test requirements are feasible

INFEASIBLE TEST REQUIREMENTS

- Realistically, test criteria will end up having infeasible test requirements.
- Some statements may simply be unreachable
- Some subpaths may rely on logical contradictions
- Worse still, it's undecidable whether all test requirements are feasible
- Allowing side trips (or even detours) may help this issue.

INFEASIBLE TEST REQUIREMENTS

Wherever possible, satisfy as many requirements directly. Failing that, allow side trips to satisfy test requirements that are otherwise infeasible.

UP NEXT...

That concludes structural coverage. Next week:

- Examples of Prime Paths!
- Data Flow Criteria
- Finite State Machines