More On Paths

Supplement to Chapter 4, Graph Theory

Path definition

- A path is a sequence of nodes $\langle\, n_1 , n_2 , \ldots , n_p \,\rangle$
  - **where each adjacent pair of nodes is in the set of edges**

$$\forall_{.} : 1 \ldots_{.} - 1 \bullet (._{.} ,._{.+1}) \in .$$

- Length
  - **Is the number of edges it contains**
    - **#path = #sequence – 1**

- We say a path **visits** the nodes in the sequence

Simple path definition

- A path from node n_j to node n_k is **simple** if
  - **No node appears more than once**
    - **There is no internal loop**
  - **Exception the end points may be the same**
    - **The entire path may form a loop**
  - **Useful property**
    - **Any path is a composition of simple paths**

Test path definition

- **What is a test path?**

Test path definition – 2

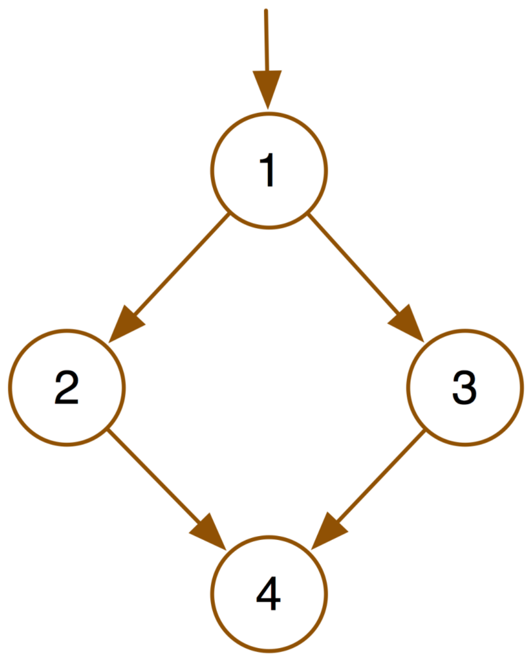
- A path
  - **Possibly of length zero**
  - **Starts at a starting node of a control flow graph**
  - **Terminates at an exit node of a control flow graph**

Prime path definition

- A path from node n_j to node n_k is **prime** if
  - **It is a simple path**
  - **It is not a proper sub-path of any other simple path**
    - **It is a maximal length simple path**
- Usefulness
  - **Reduces the number of test cases for path coverage**
- Problem
  - **A prime path may be infeasible but contain feasible simple paths**
    - **In such cases, the prime path is factored into the simple paths in order that they may be covered by testing**

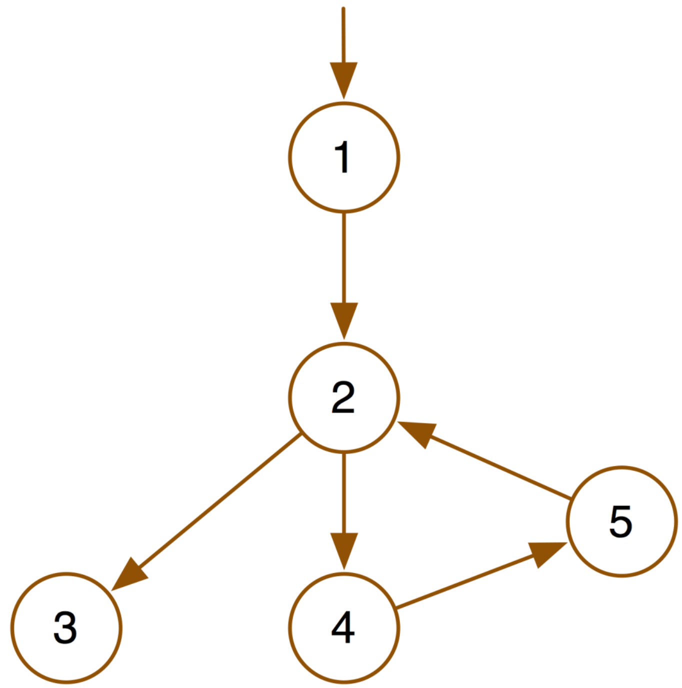
Prime path – Example 1

- Prime paths
  - $\langle n_1, n_2, n_4 \rangle$ $\langle n_1, n_3, n_4 \rangle$



Prime path – Example 2

- Prime paths
  - $\langle n_1, n_2, n_3 \rangle \langle n_1, n_2, n_4, n_5 \rangle \langle n_2, n_4, n_5, n_2 \rangle$
  - $\langle n_4, n_5, n_2, n_4 \rangle \langle n_5, n_2, n_4, n_5 \rangle \langle n_4, n_5, n_2, n_3 \rangle$



Round trip path definition

- **What is a round trip path?**

Round trip path definition – 2

- A path P is a round trip path if
  - **P is prime**
  - **#P > 0**
  - **head P = last P**

Tour definition

- **What is a tour?**

Tour definition – 2

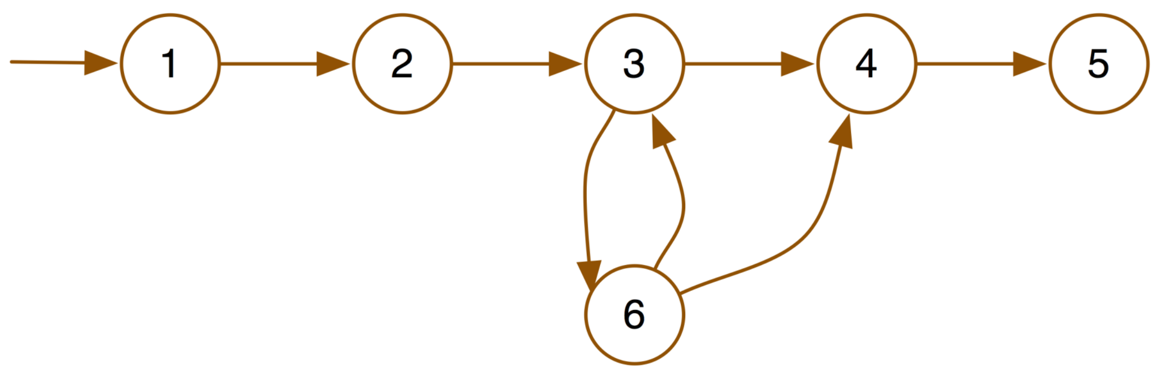
- A test path is said to **tour** a graph path if
  - **graph-path $\subseteq$ test-path**
    - **$\subseteq$ in this context means sub-path – not subset**
    - **The test-path must visit the graph-path nodes in exactly the specified sequence with no intervening nodes**

Tour with side trips definition

- **What is a tour with side trips?**

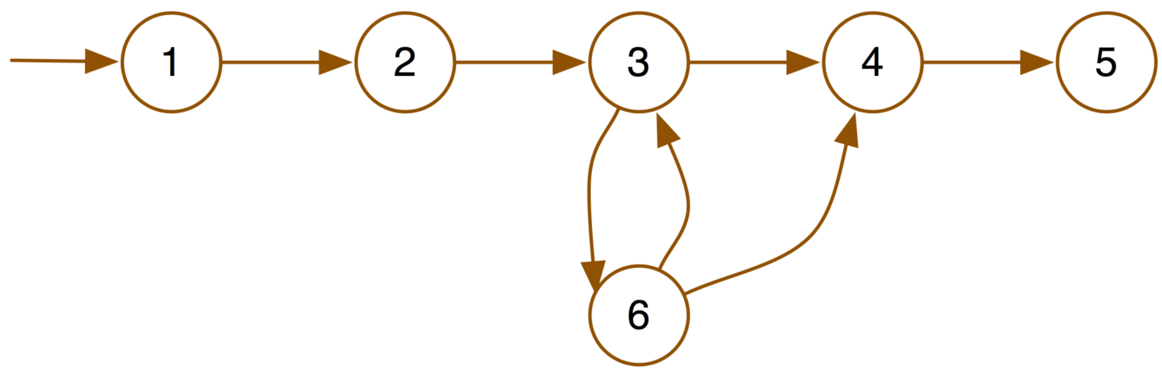
Tour with side trips definition – 2

- A tour as specified is restrictive in that many test paths would be infeasible
  - **This occurs when loops are in the path**
  - **The path $\langle n_2, n_3, n_4 \rangle$ would be impossible to tour if the condition in n_3 is such that n_6 must be visited at least once**



Tour definition augmented

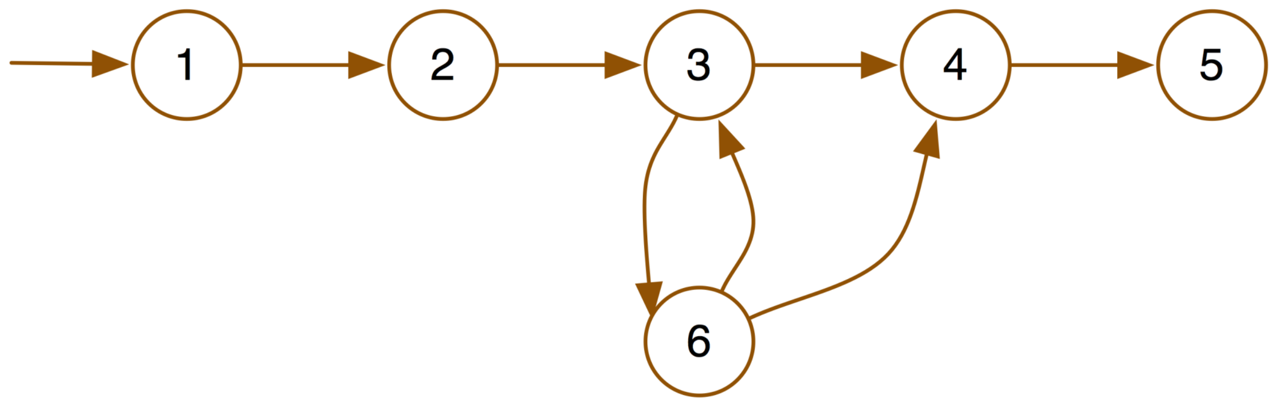
- We relax the definition of a tour to include side trips
  - **Leave the sub-path**
  - **But come back to the same node before continuing the sub-path – e.g. $\langle n_2 , n_3 , n_6 , n_3 , n_6 , n_3 , n_4 \rangle$**



- Test path tours the graph-path with side trips iff every edge of the graph-path is followed in the same order

Tour with detours definition

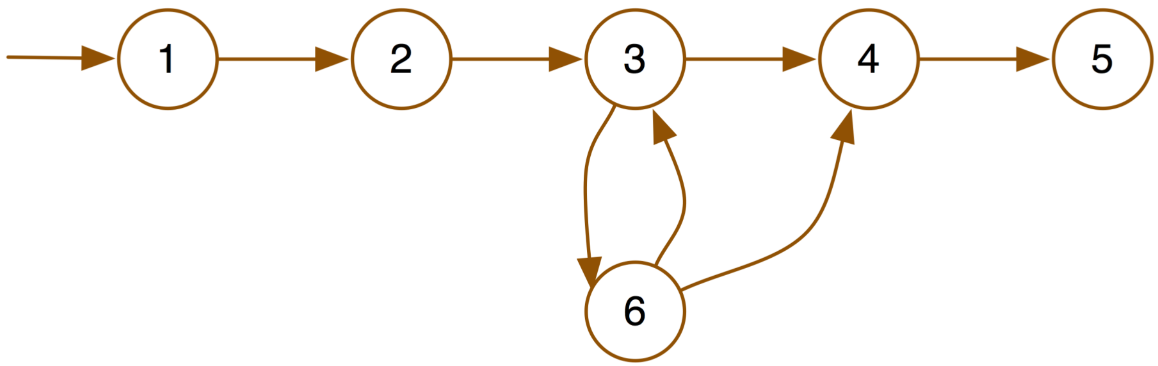
- **What is a tour with detours?**



Tour with detours definition – 2

- We relax the definition of a tour to include detours
  - **Leave the sub-path**
  - **But come back to the node that follows the node where the sub-path was left**

    – e.g. $\langle n_2, n_3, n_6, n_3, n_6, n_4 \rangle$



- Test path tours the graph-path with detours iff every node of the graph-path is followed in the same order

Best effort touring

- TR_{tour} is a set of test requirements such that
  - **Paths in a graph that must be covered**
    - **Can be directly toured**
- $TR_{sidetrips}$ is a set of test requirements
  - **Paths in a graph that must be covered**
    - **Can be directly toured**
    - **Or toured with sidetrips**

Best effort touring – 2

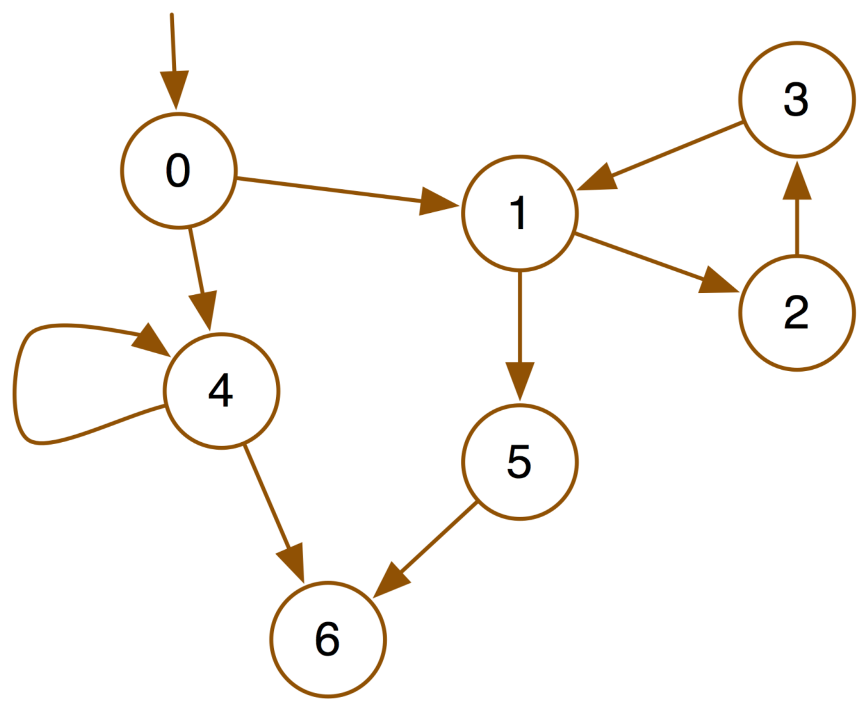
- A test set T is best effort touring if
  - **For every path p in TR_{tour}**
    - **Some path in T tours p directly**
  - **For every path p in $TR_{sidetrips}$**
    - **Some path in T tours p either directly or with a side trips**
- Each test requirement is met in the strictest possible way

Best effort touring – 3

- Trips with detours are rarely considered
  - **They are less practical than sidetrips in dealing with infeasible paths**

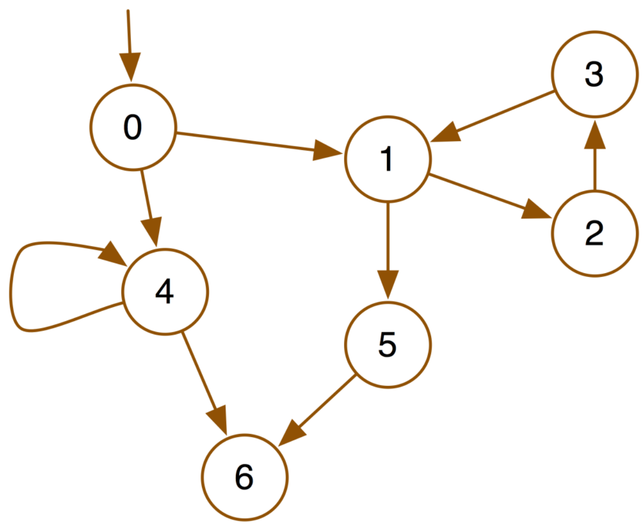
Finding prime paths

- Consider the following graph, what are its prime paths?



Finding prime paths – length 0 paths

- Start with a list of the nodes
  - **The ! Indicates that the path cannot be extended**

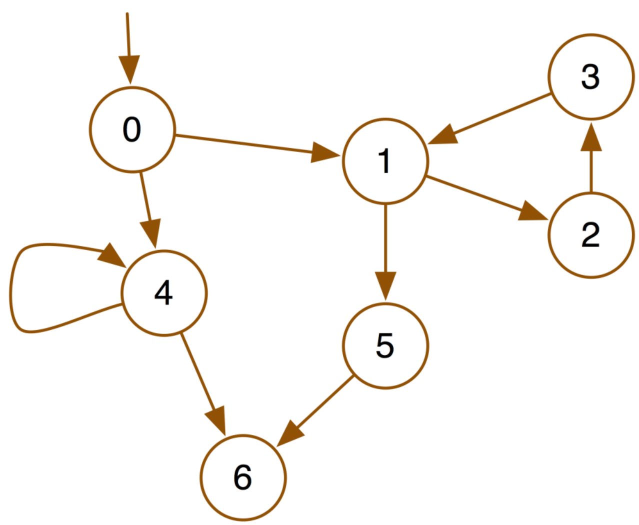


1 **[0]**
2 **[1]**
3 **[2]**
4 **[3]**
5 **[4]**
6 **[5]**
7 **[6] !**

Finding prime paths – length 1 paths

- Extend length 0 paths by one edge
  - **Path 7 cannot be extended**
  - **The * indicates a loop – cannot be extended**

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

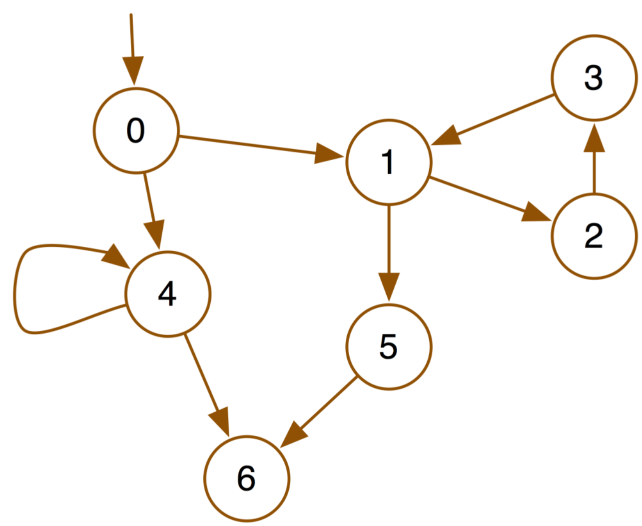


8	**[0, 1]**
9	**[0, 4]**
10	**[1, 2]**
11	**[1, 5]**
12	**[2, 3]**
13	**[3, 1]**
14	**[4, 4] ***
15	**[4, 6] !**
16	**[5, 6] !**

Finding prime paths – length 2 paths

- Extend length 1 paths by one edge
  - **Paths 14, 15 and 16 cannot be extended**

8 [0, 1]
9 [0, 4]
10 [1, 2]
11 [1, 5]
12 [2, 3]
13 [3, 1]
14 [4, 4] *
15 [4, 6] !
16 [5, 6] !

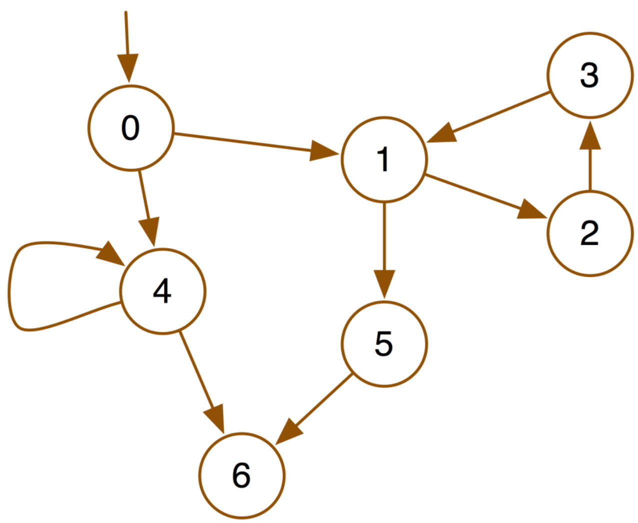


17 [0, 1, 2]
18 [0, 1, 5]
19 [0, 4, 6] !
20 [1, 2, 3]
21 [1, 5, 6] !
22 [2, 3, 1]
23 [3, 1, 2]
24 [3, 1, 5]

Finding prime paths – length 3 paths

- Extend length 2 paths by one edge
  - **Paths 19 and 21 cannot be extended**

17 [0, 1, 2]
18 [0, 1, 5]
19 [0, 4, 6] !
20 [1, 2, 3]
21 [1, 5, 6] !
22 [2, 3, 1]
23 [3, 1, 2]
24 [3, 1, 5]

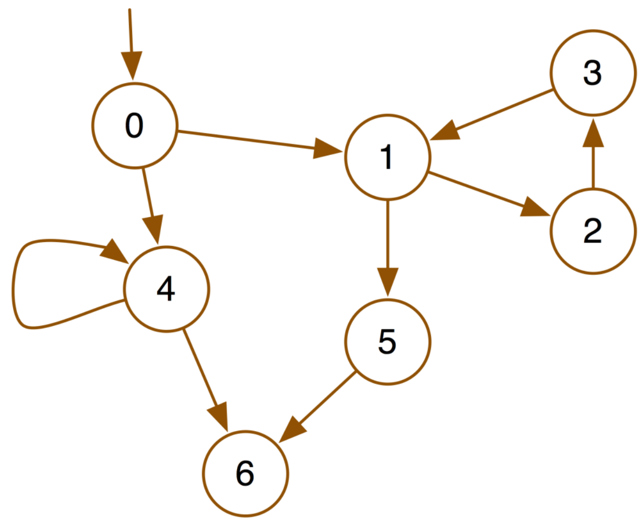


25 [0, 1, 2, 3] !
26 [0, 1, 5, 6] !
27 [1, 2, 3, 1] *
28 [2, 3, 1, 2] *
29 [2, 3, 1, 5]
30 [3, 1, 2, 3] *
31 [3, 1, 5, 6] !

Finding prime paths – length 4 paths

- Extend length 3 paths by one edge
  - **Only path 29 be extended and no further extensions are possible**

25 **[0, 1, 2, 3] !**
26 **[0, 1, 5, 6] !**
27 **[1, 2, 3, 1] ***
28 **[2, 3, 1, 2] ***
29 **[2, 3, 1, 5]**
30 **[3, 1, 2, 3] ***
31 **[3, 1, 5, 6] !**



32 **[2, 3, 1, 5, 6] !**

Finding prime paths – Collect paths

- No more paths can be extended.
- Collect all the paths that terminate with ! or *
- Eliminate any path that is a subset of another path in the list

14 [4, 4] *
19 [0, 4, 6] !
25 [0, 1, 2, 3] !
26 [0, 1, 5, 6] !
27 [1, 2, 3, 1] *
28 [2, 3, 1, 2] *
30 [3, 1, 2, 3] *
32 [2, 3, 1, 5, 6] !

These are the 8 prime paths In the example graph

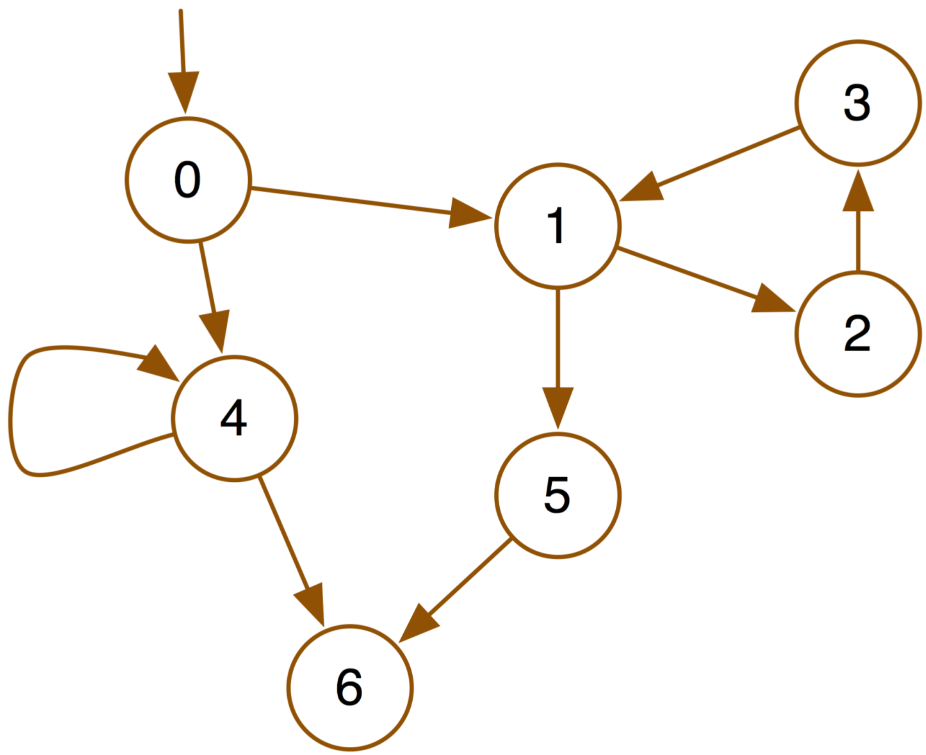
Coverage criteria

- A coverage criterion is a rule or collection of rules that impose test requirements on a test set.

  - **A recipe for generating test requirements in a systematic way**

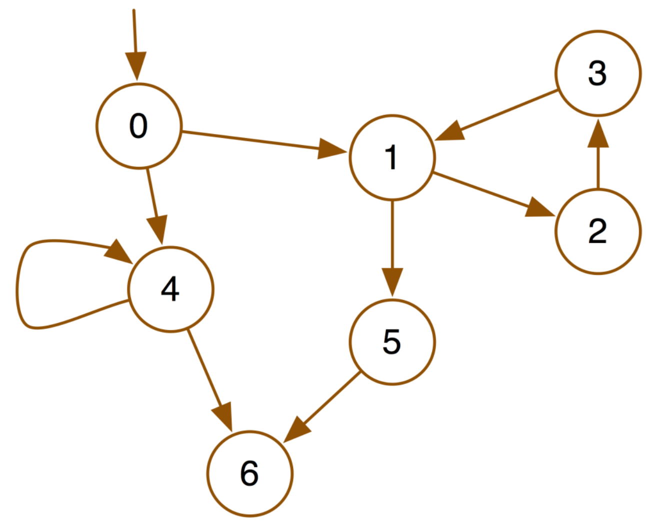
Coverage criteria – 2

- Consider the following graph, what test coverage criteria can we have?



Coverage criteria – 3

- Coverage can be the following
  - **All nodes**
  - **All edges**
  - **All edge pairs**
    - **More edges not useful**
  - **All simple paths**
  - **All prime paths**
  - **All simple round trips**
  - **All complete round trips**
  - **All paths**
  - **All specified paths**

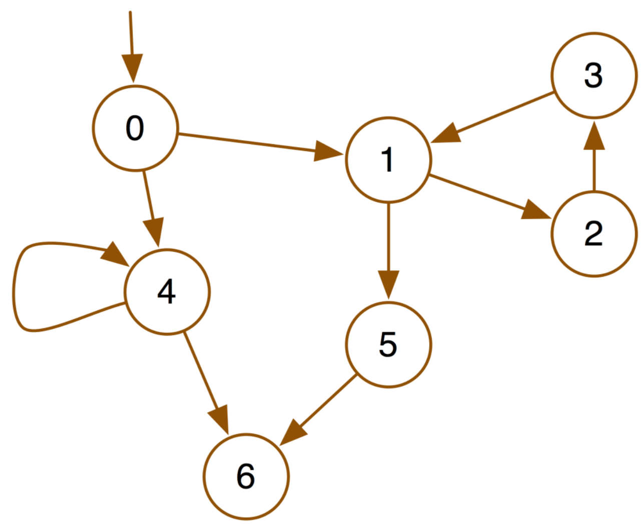


Test requirement

- Is a specific element of a software artifact that a test case must satisfy or cover.
- Usually come in sets
  - **Use the abbreviation *TR* to denote a set of test requirements.**
- Can be described with respect to a variety of software artifacts, including
  - **Program text**
  - **Design components**
  - **Specification modeling elements**
  - **Even descriptions of the input space.**

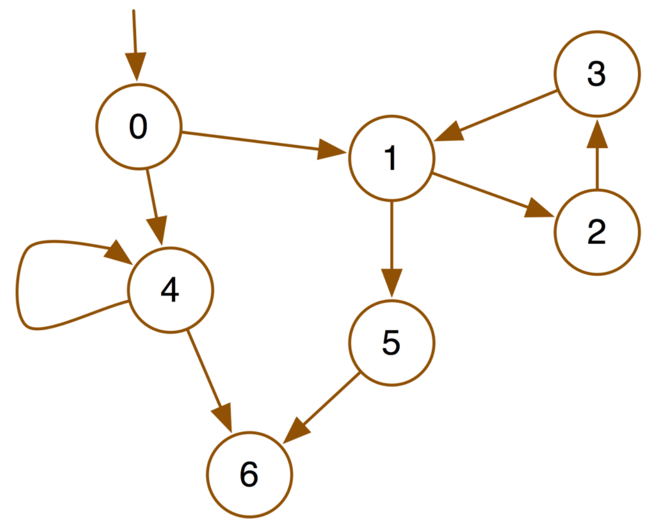
Test requirements

- Given the pictured graph and the coverage criterion "All nodes"
- The test requirements is a listing of all the nodes in the graph
  - **{ 0, 1, 2, 3, 4, 5, 6 }**



Test set

- A test set satisfies test requirements by visiting every artifact in the test requirements
- Given the test requirements to visit all nodes in the following set for the pictured graph
  - **{ 0, 1, 2, 3, 4, 5, 6 }**
- The following test set satisfies the test requirements
  - **{ [0, 4, 4, 4, 6]
, [0, 1, 2, 3, 1, 5, 6] }**



Path behaviours

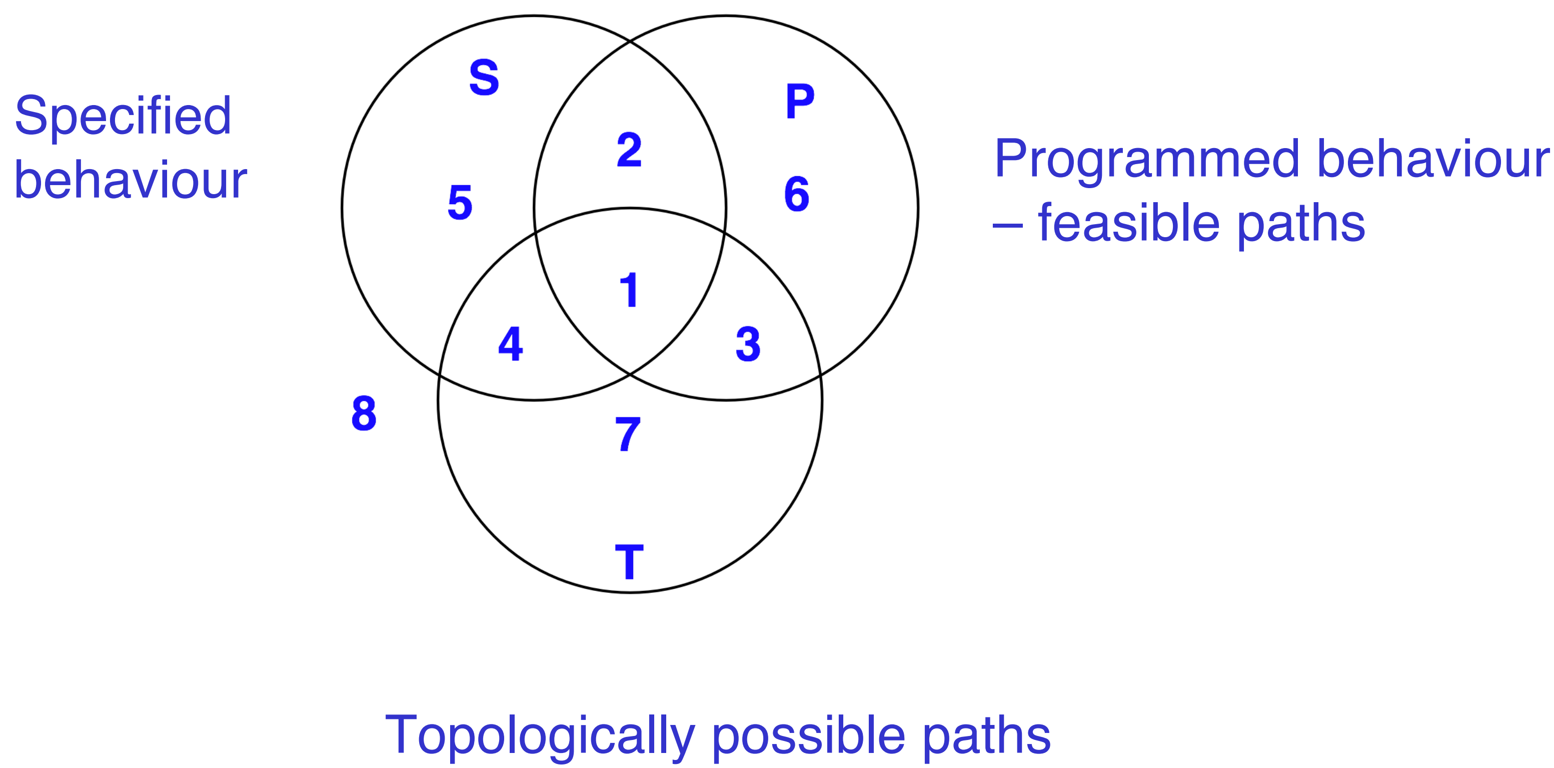
- **When looking at paths we distinguish three types of path behaviours , what are they?**

Path behaviours– 2

- Distinctions are made with the following types of paths
  - **Feasible**
  - **Specified**
  - **Topologically possible**

Path behaviours – 3

- Re-examine the Venn diagram in the context of path testing



Guidelines

- Functional testing
  - **Too far from the program text**

- Path testing
  - **Too close to the program text**
  - **Obscures feasible and infeasible paths**
    - **Use dataflow testing to move out a bit**

Guidelines – 2

- Path testing
  - **does not give good help in finding test cases**
  - **does give good measures of quality of testing through coverage analysis**
  - **Provides set of metrics that cross-check functional testing**
  - **Use to resolve gap and redundancy questions**
    - **Missing DD-paths – have gaps**
    - **Repeated DD-paths – have redundancy**