Coverage

Lecture Objectives:

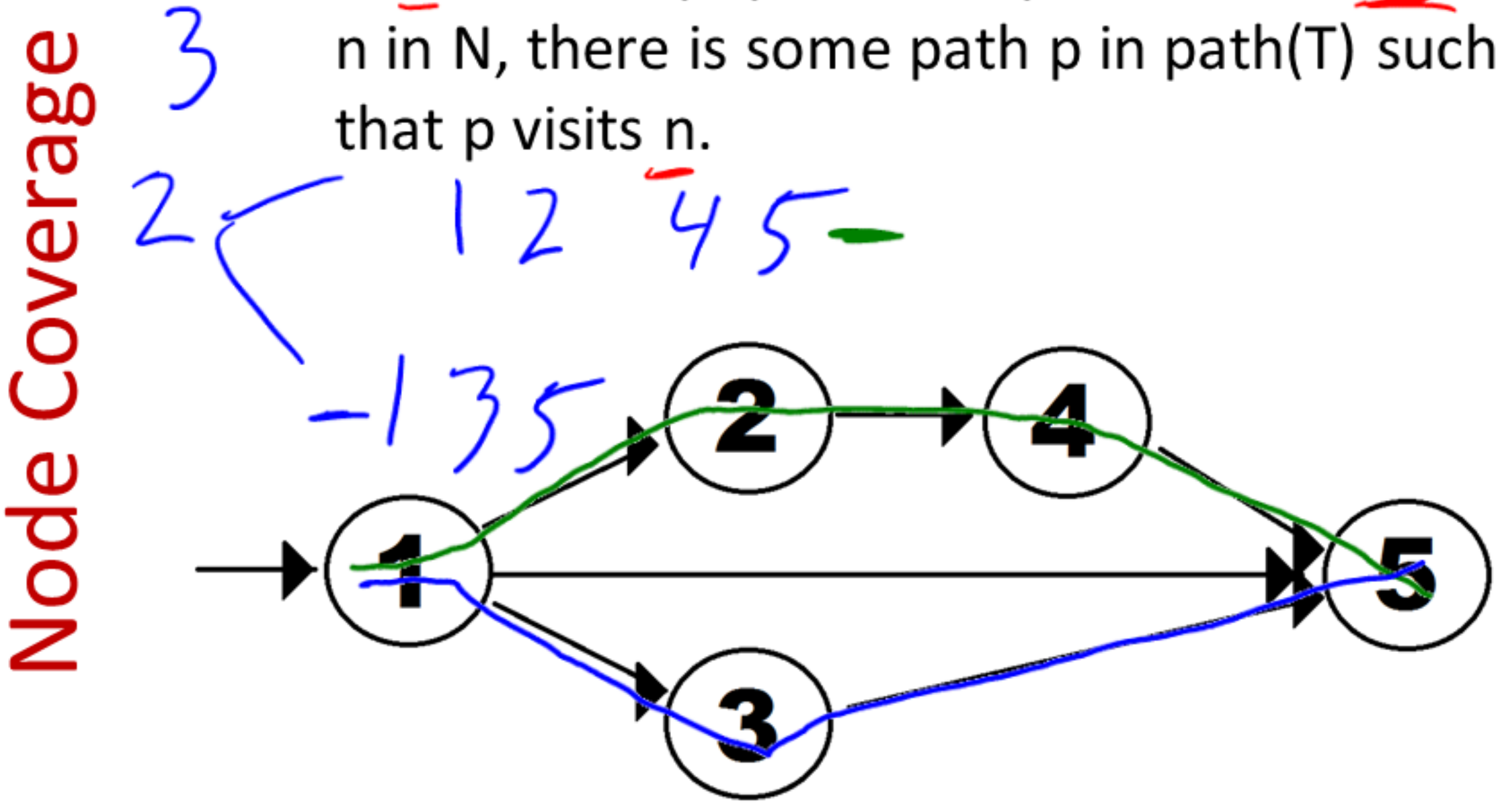
1) Define node coverage
2) Define edge coverage
3) Define edge-pair coverage
4) Define prime path
5) Define prime path coverage
6) Construct a set of test paths which meet the criteria for node coverage
7) Construct a set of test paths which meet the criteria for edge coverage

Testing and Graphs

- We use graphs in testing as follows :
  - Developing a model of the software as a graph
  - Requiring 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
- Data Flow Coverage Criteria
  - Requires a graph to be annotated with references to variables

- 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.

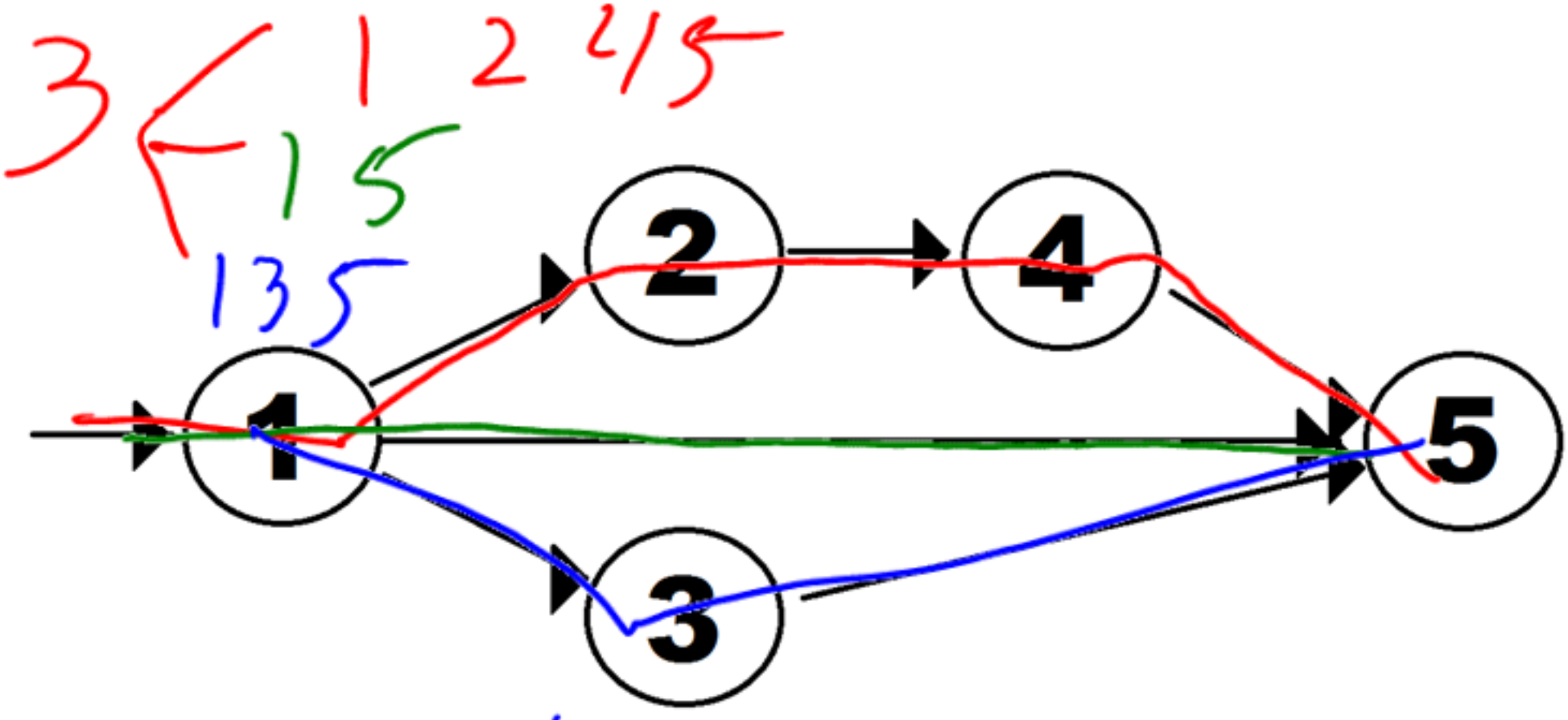
Node Coverage



- Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.

Edge Coverage

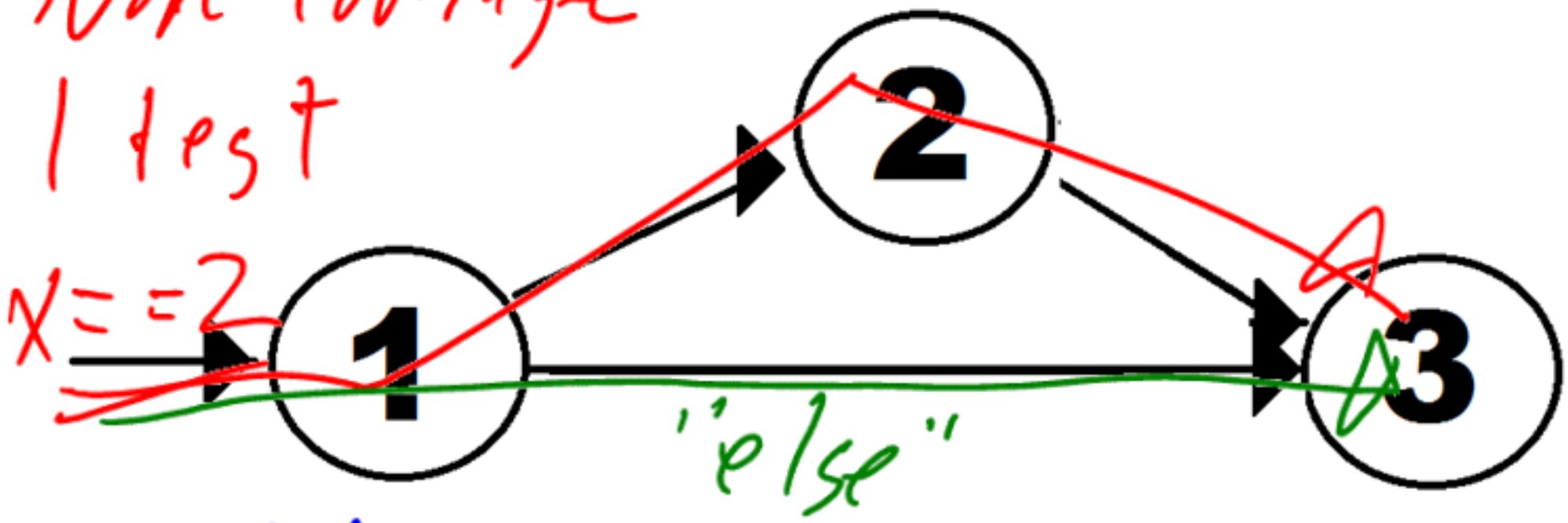
Slightly stronger, better form of testing.



Better but harder

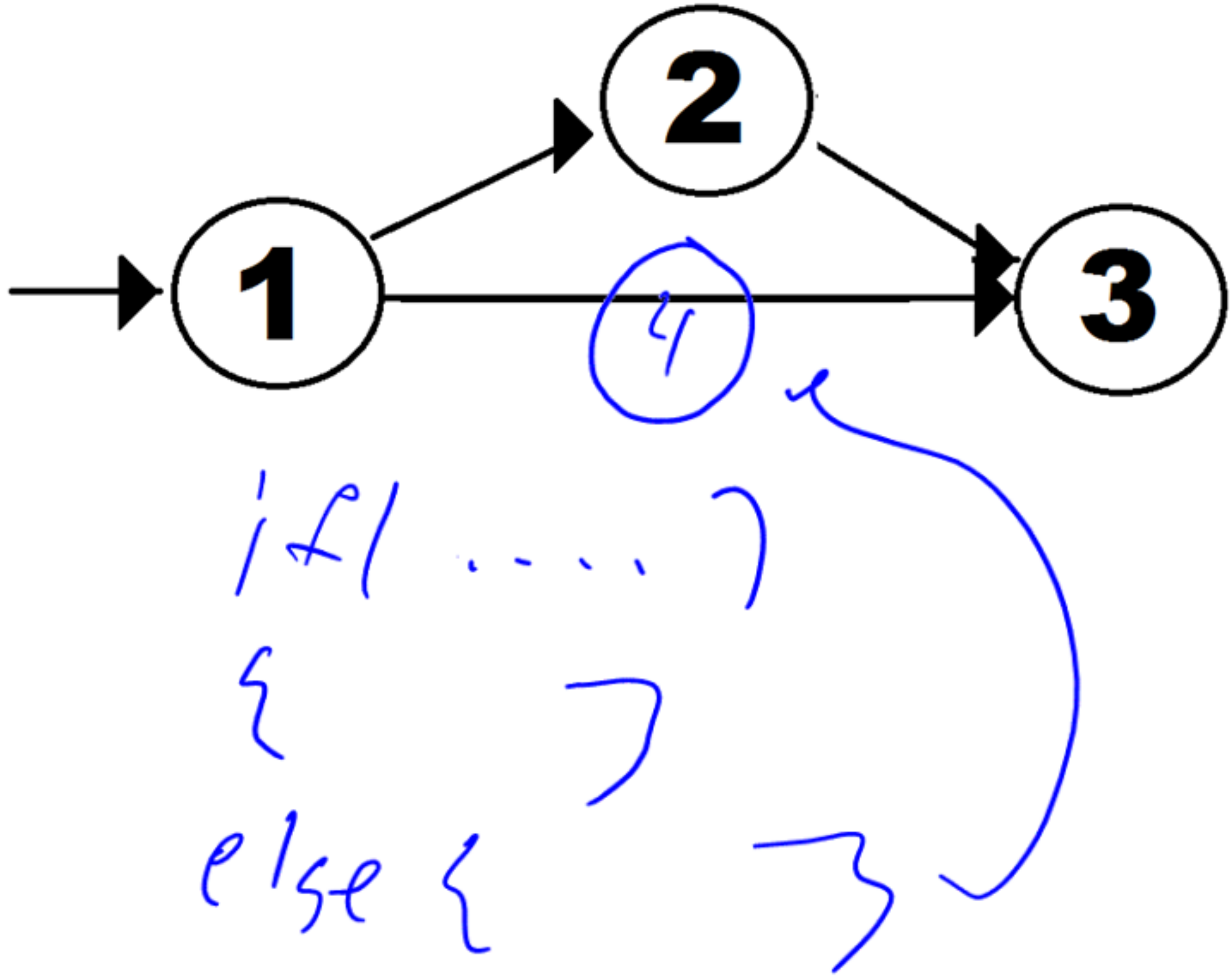
Why is edge coverage better than node coverage?

Node coverage
1 test




```
if(x==2)
{ System.out.println("2");
}
System.out.println("3");
```

Why is edge coverage better than node coverage?

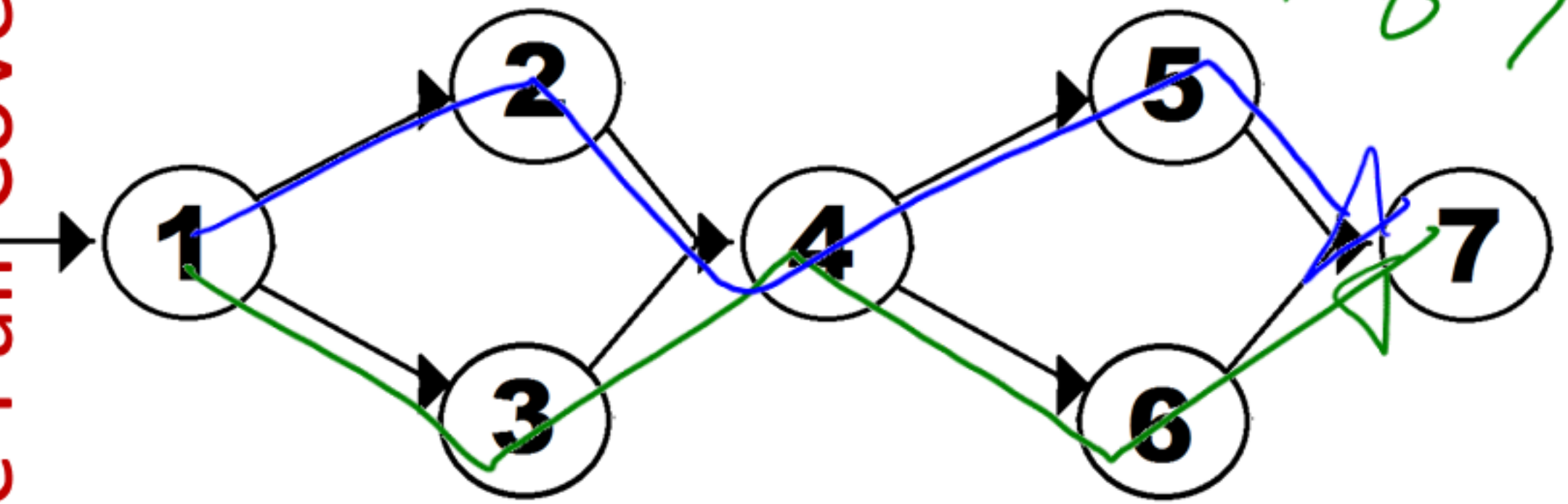


- TR contains each reachable path of length up to 2, inclusive, in G.

Edge-Pair Coverage

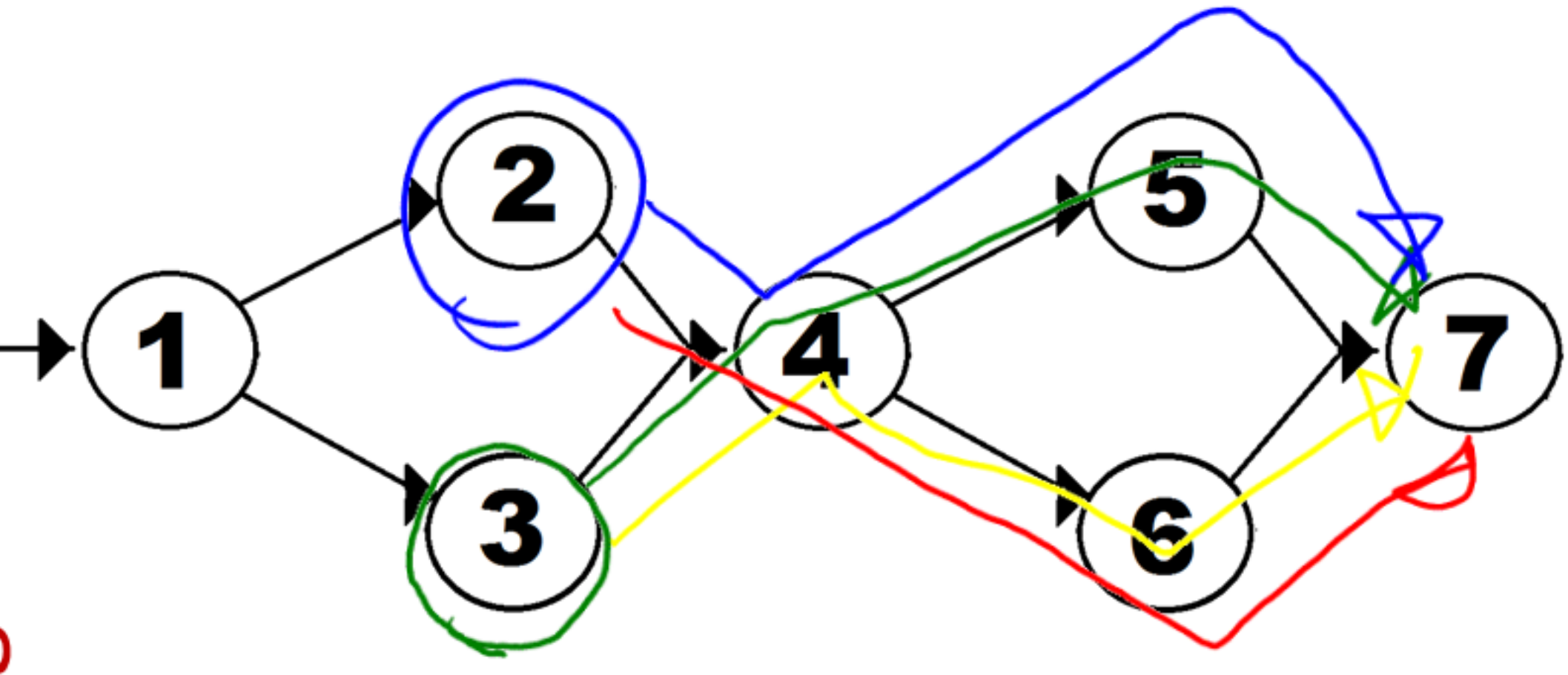
Note Coverage: 2=7 1 2 4 5 7

1 3 4 6 7



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

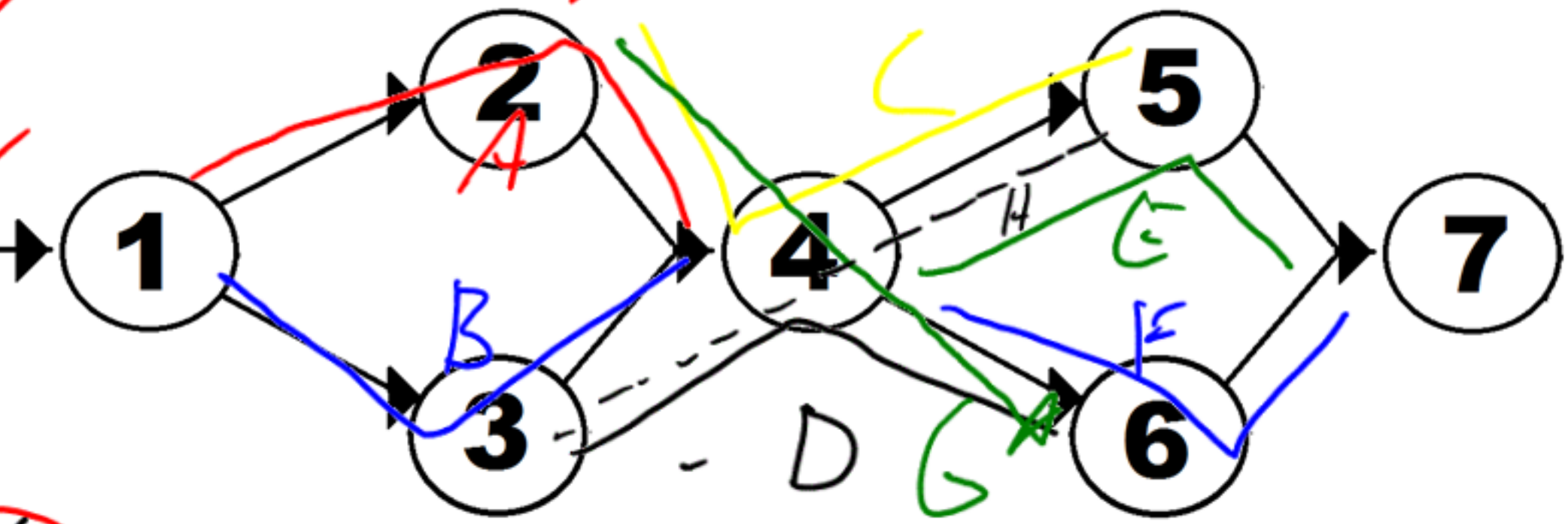
Edge-Pair Coverage



Edge coverage: 2 test cases as well

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

All Lengths of 2 must be tested.



Edge-Pair Coverage

A ✓
B ✓
C ✓
D ✓
E ✓
F ✓
G ✓
H ✓

✓ 1 2 4 5 7 | G 1 2 4 6 7
1 3 4 6 7 | H 1 3 4 5 7

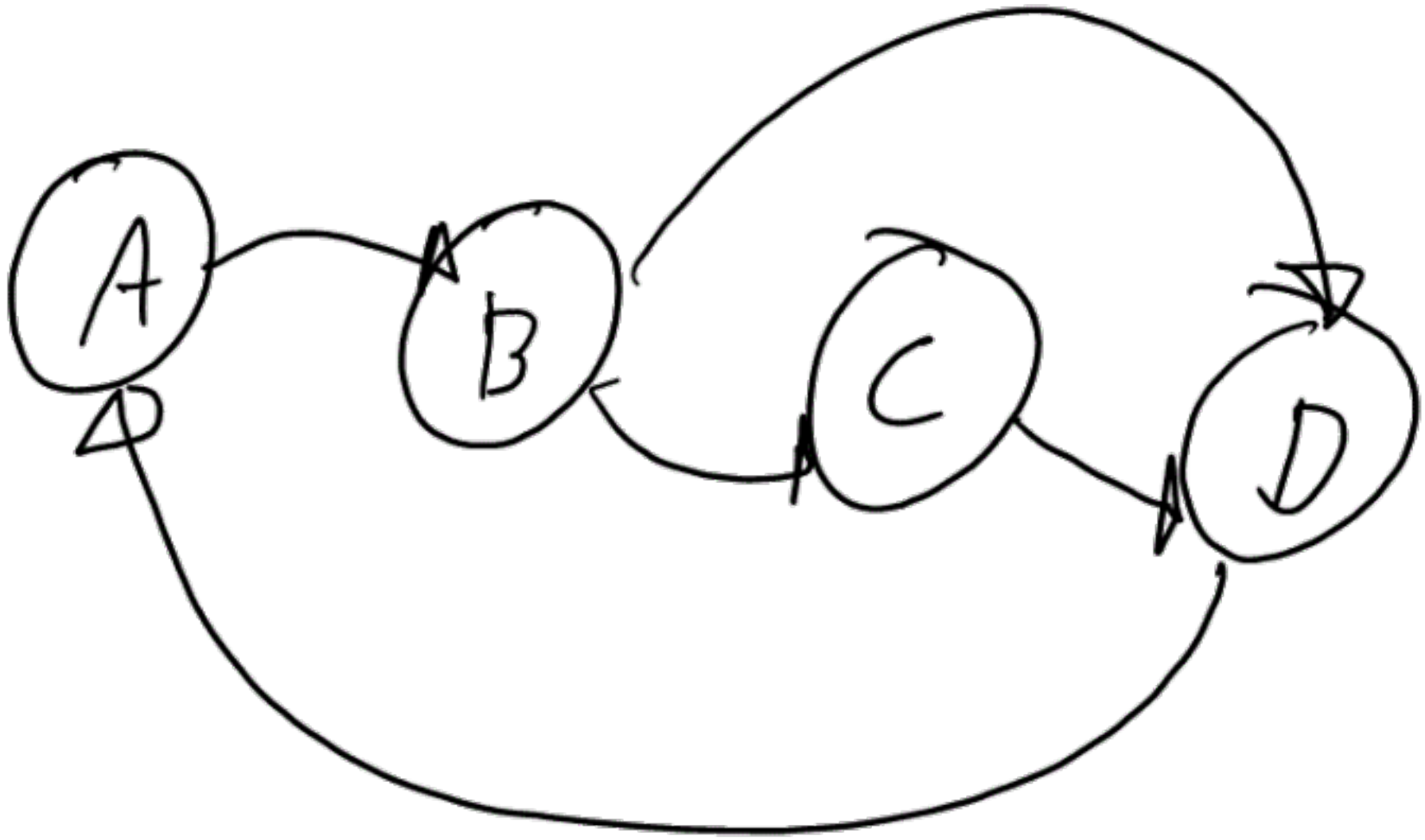
Edge Pair: Superior do node

Superior to edge.

But: More test cases.

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

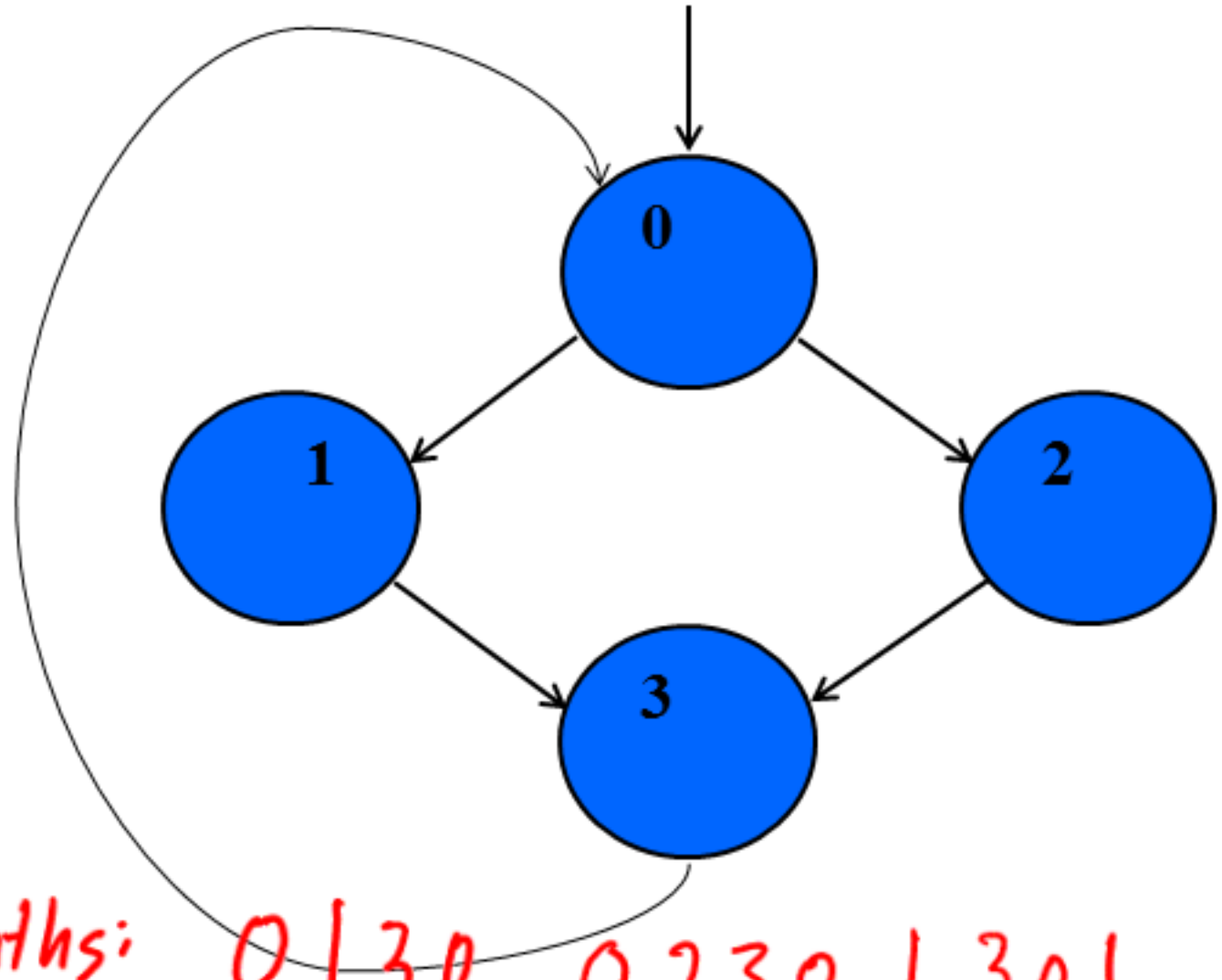
Simple and prime paths



A B C D

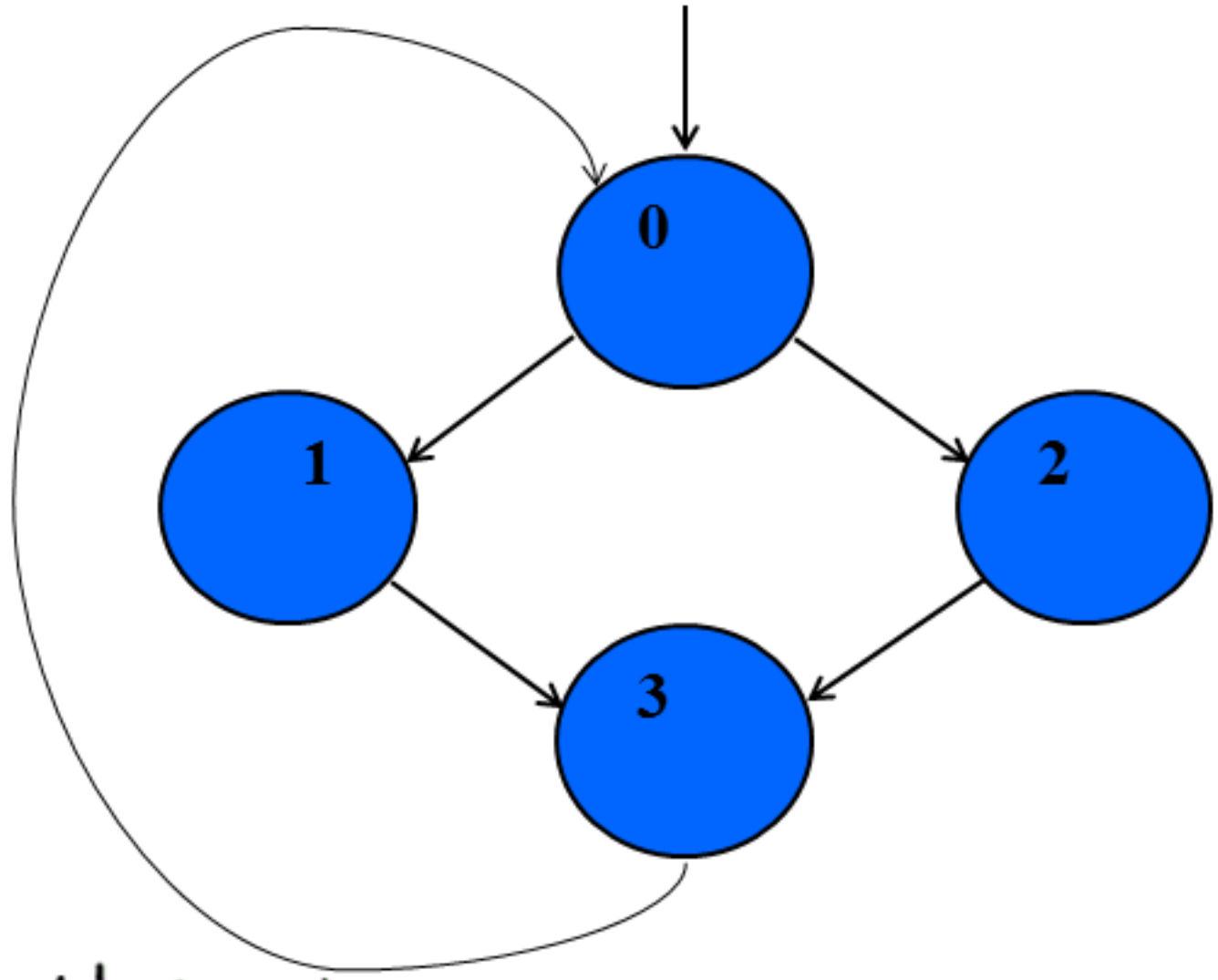
A B C D A

Simple and Prime Paths



Simple Paths: 0130, 0230, 1301,
2302, 3013, 3023, 1302,
301 302 01 02 13 23
301 0 1 2 3

Simple and Prime Paths



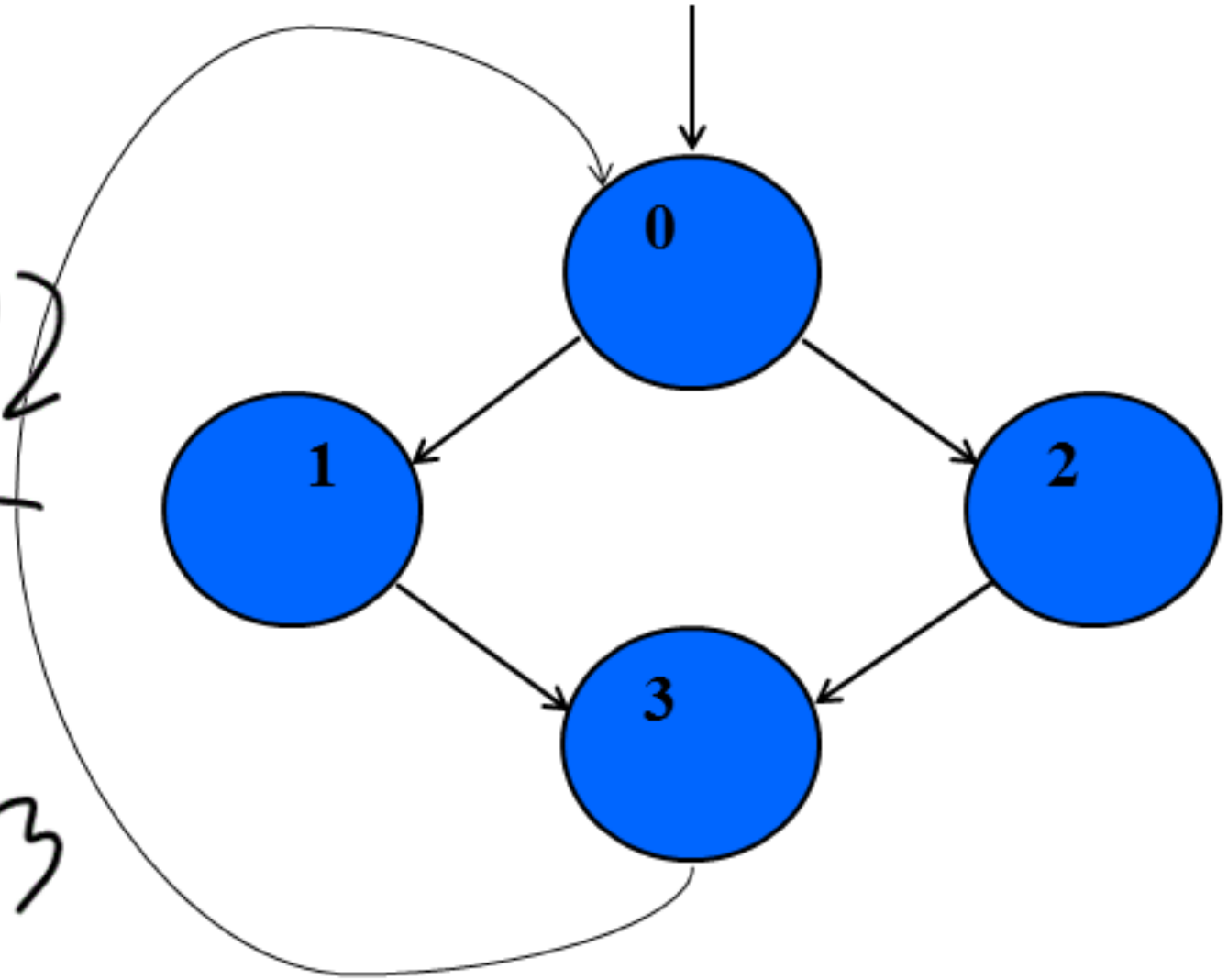
Prime paths: Prime path all have 4 nodes in them

0130, 0230, 1301, 2302, 3013, 3023, 1302, 2301

Simple and Prime Paths

Node:
0 1 3 0 2

Edge
0 1 3 0 2 3
:



Prime Path Coverage / Complete Path Coverage

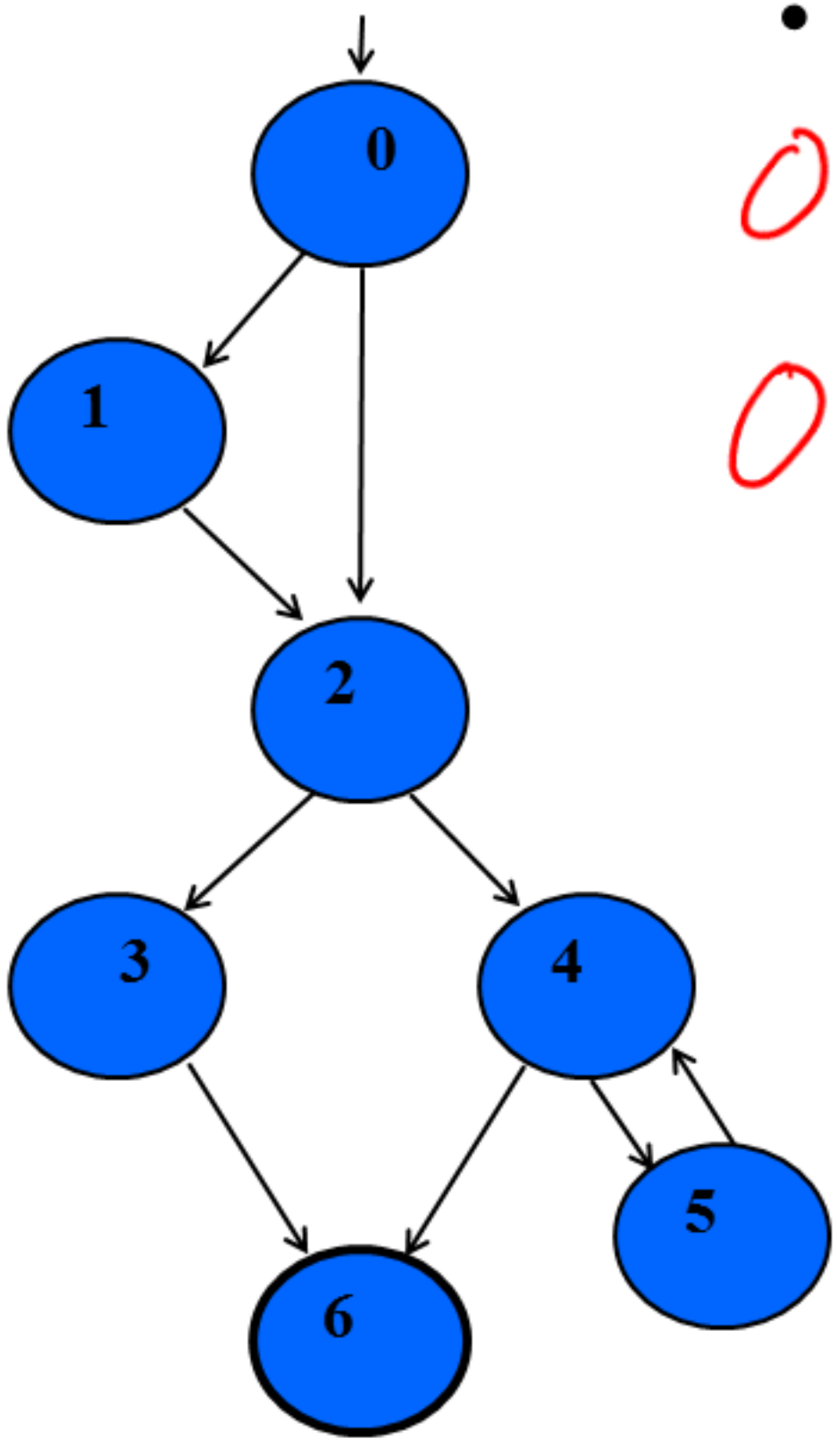
- Prime Path coverage
  - TR contains each prime path in G.

Every prime path is a test case. => # of test cases. "Lots"

- Complete Path Coverage
  - TR contains all paths in G.

All paths are tested. May be infinit # of paths

Exercise: Construct test cases

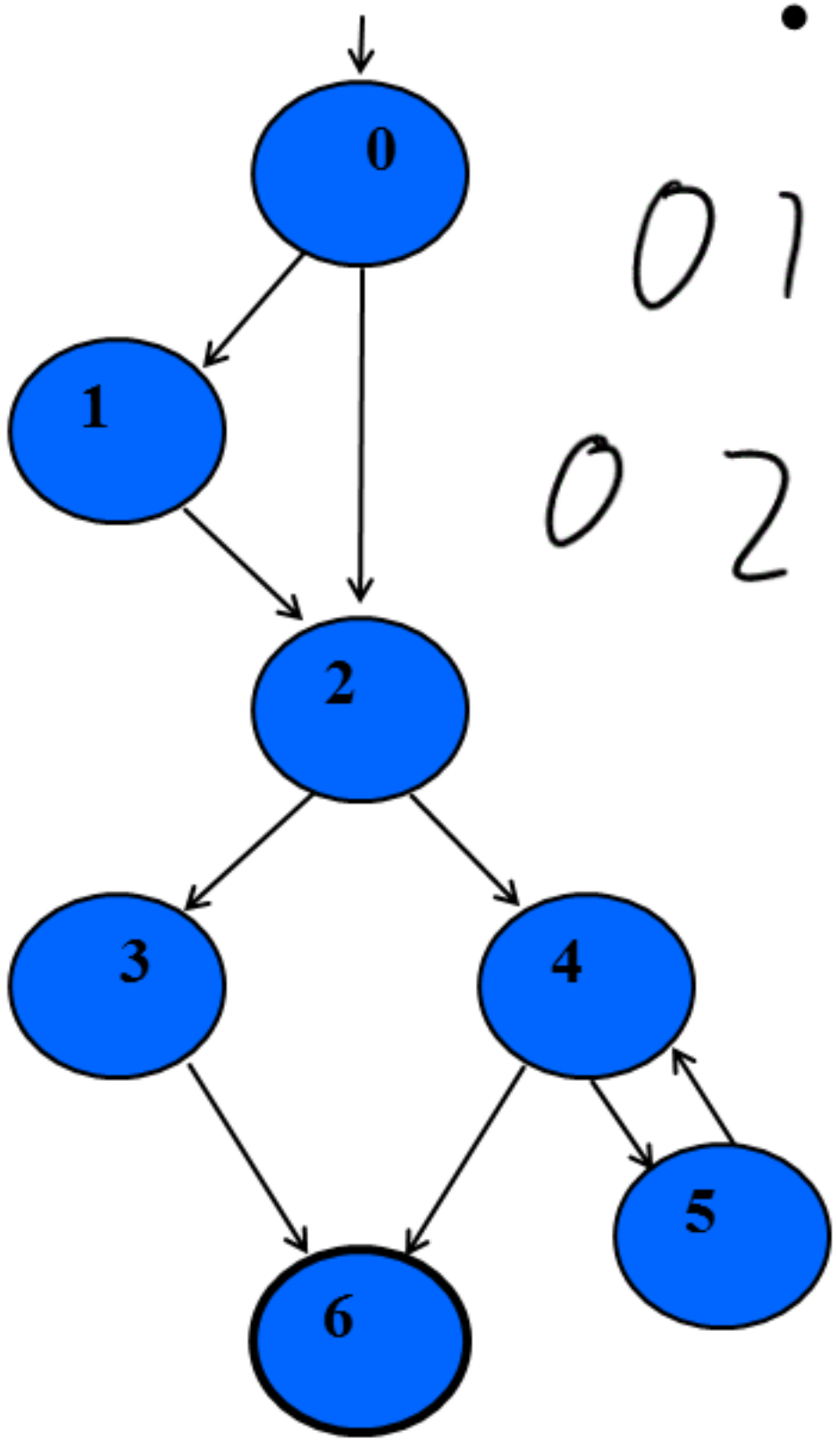


- Node Coverage

0 1 2 4 5 4 6

0 (1) 2 3 6

Exercise: Construct test cases

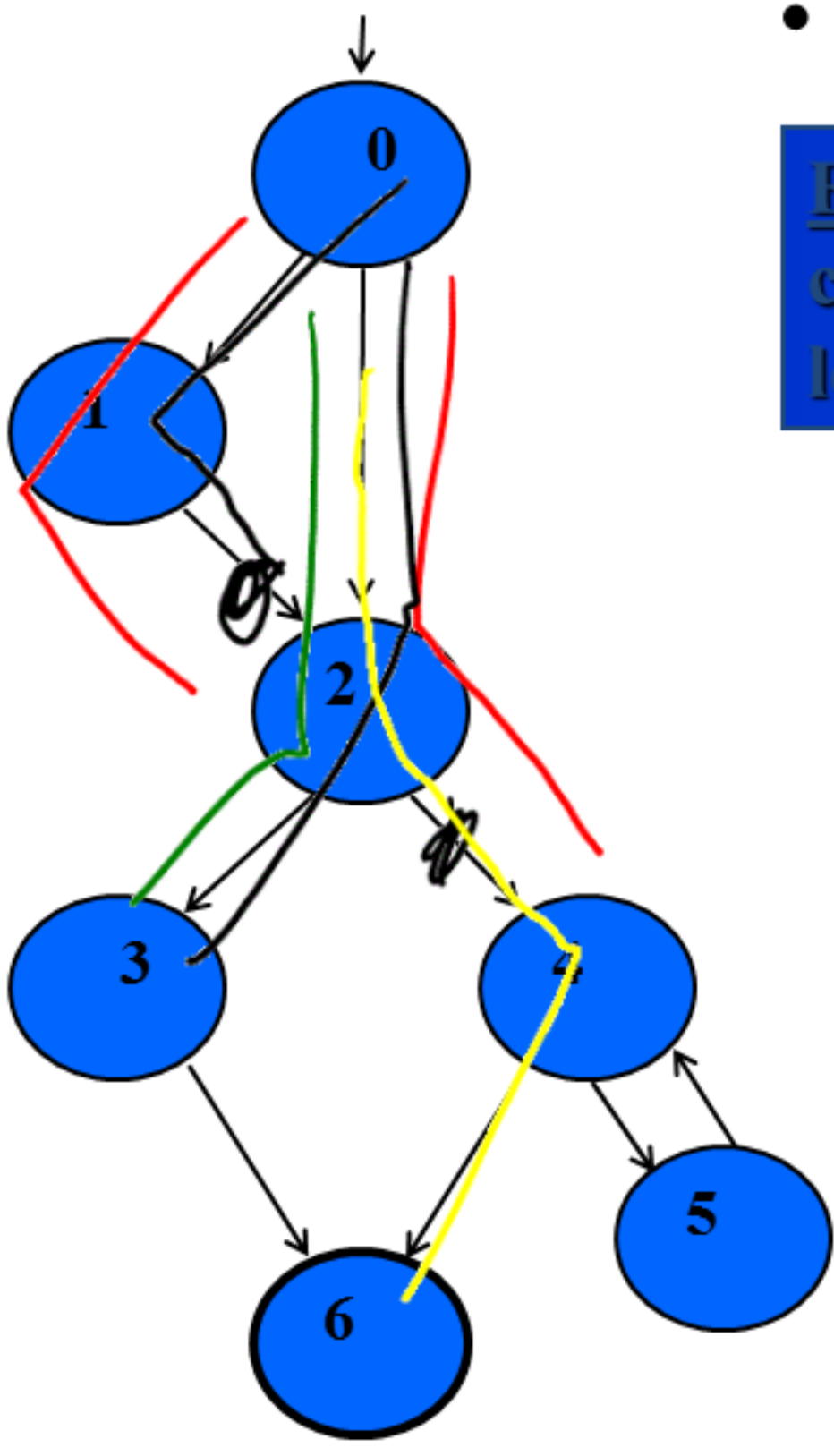


- Edge Coverage

0 1 2 4 5 4 6

0 2 3 6

Exercise: Construct test cases



- Edge Pair Coverage

Edge-Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

0 1 2 3 6

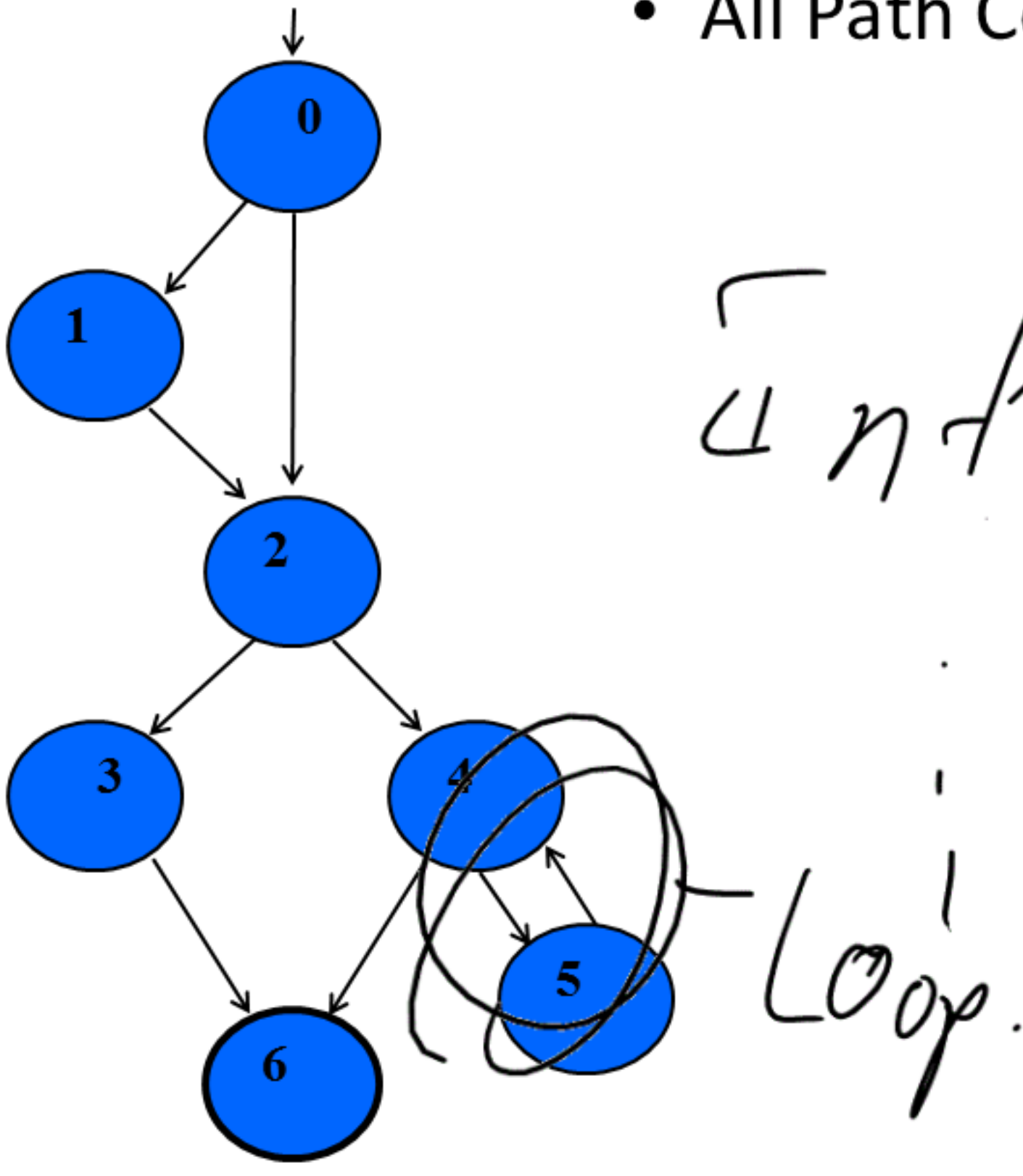
0 2 3 6

0 1 2 4 5 4 6

0 2 4 6

MSOE

Exercise: Construct test cases

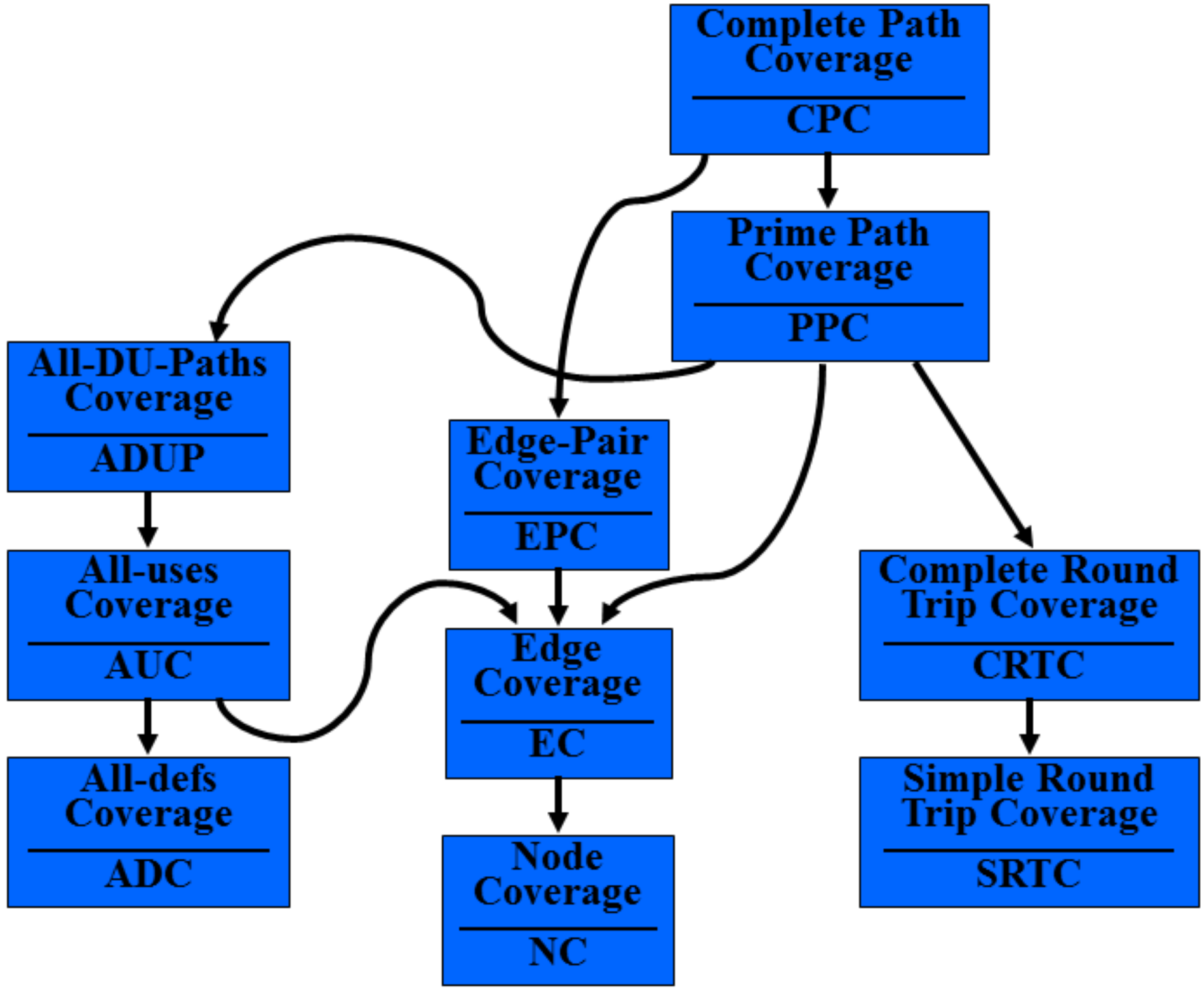


- All Path Coverage

Infinite

Loop.

Relationship between coverage criteria



How does this relate to the real world?

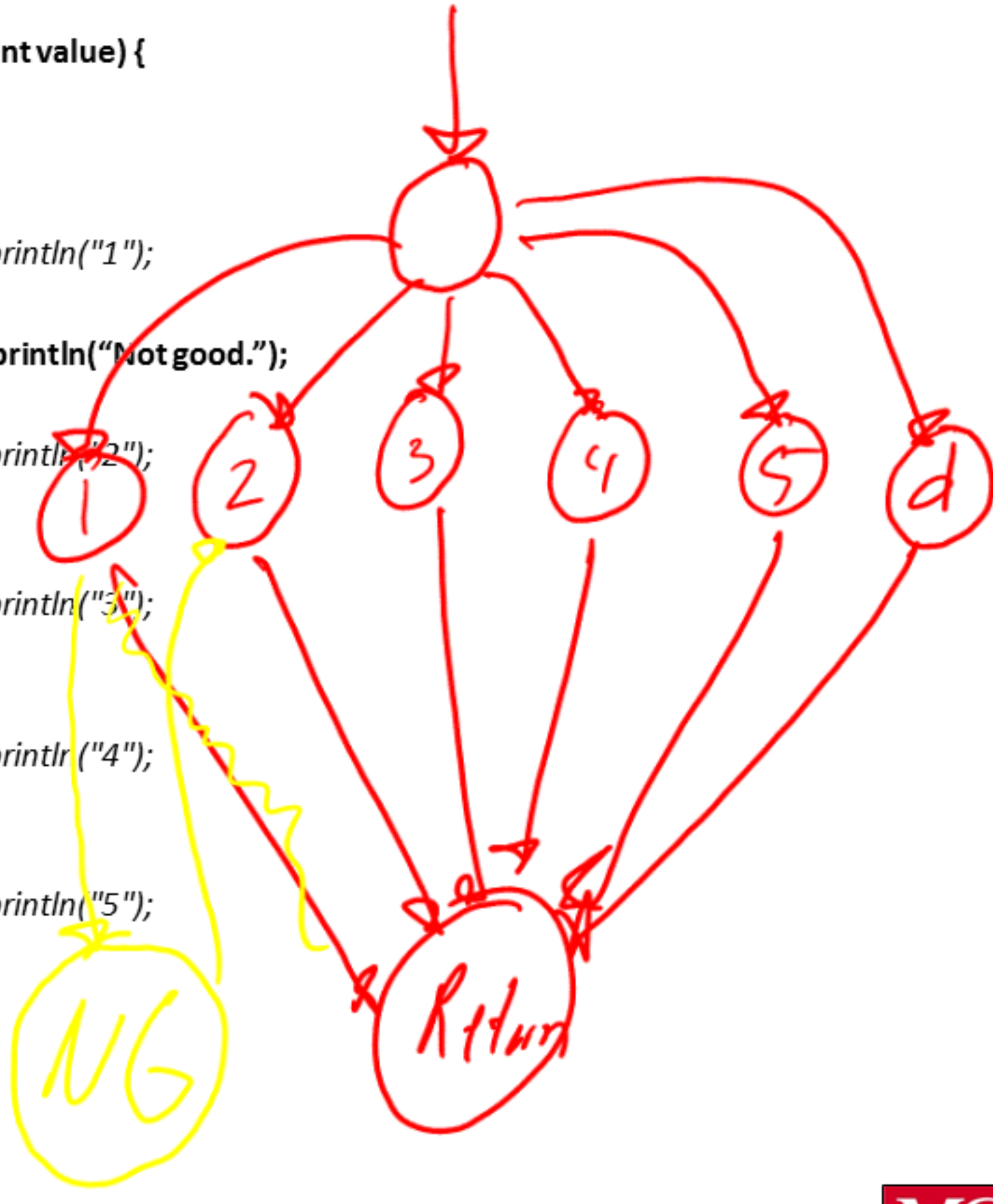
- What does this mean from last week's lab?
- Which coverage types does EMMA provide?

⇒ "essentially" node coverage.

⇒ Branch-it indicates which branches were taken.

Going Back to code

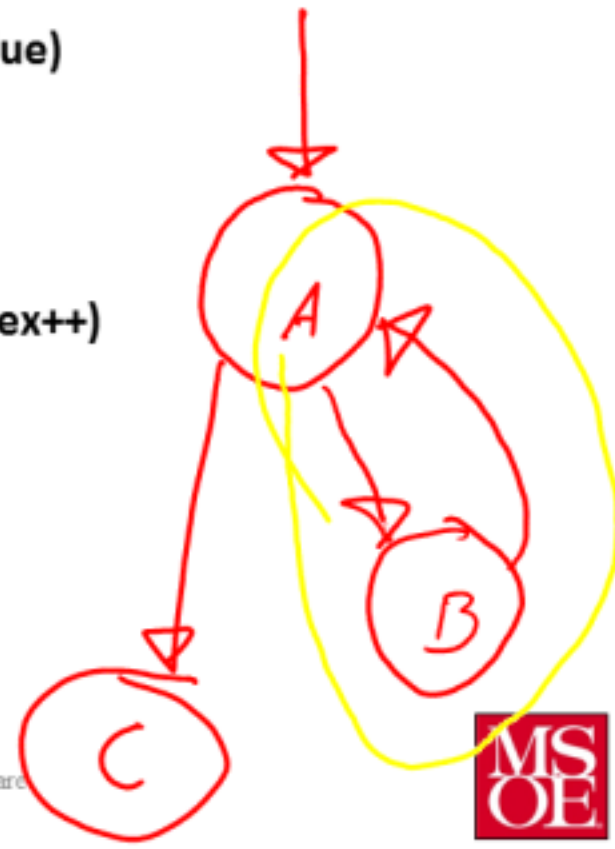
```
public static void printit(int value) {
    value = value % 5;
    switch (value) {
            case 1:
            System.out.println("1");
            break;
            System.out.println("Not good.");
            case 2:
            System.out.println("2");
            break;
            case 3:
            System.out.println("3");
            break;
            case 4:
            System.out.println("4");
            break;
            case 5:
            System.out.println("5");
            break;
            default:
            break;
    }
    return;
}
```



```
/**
 * The method will sum the numbers between 1 and the actual
number.
 * @param value This is the number that is to be counted up to
 * @return
 */
public static int sumNumbers(int value)
{
int retValue = 0;

for (int index = -1; index < value; index++)
{
  retValue += index;
}
return retValue;
}
```

Sum of numbers between 1 and final.



dealing with loops

init

by Dec.

worst test cases

MSOE

3 test Cases

1. Boarder on
Lower

2. Normal

3. Very Large.

4. Negative.

0 times.

1 time.

n times n > 1