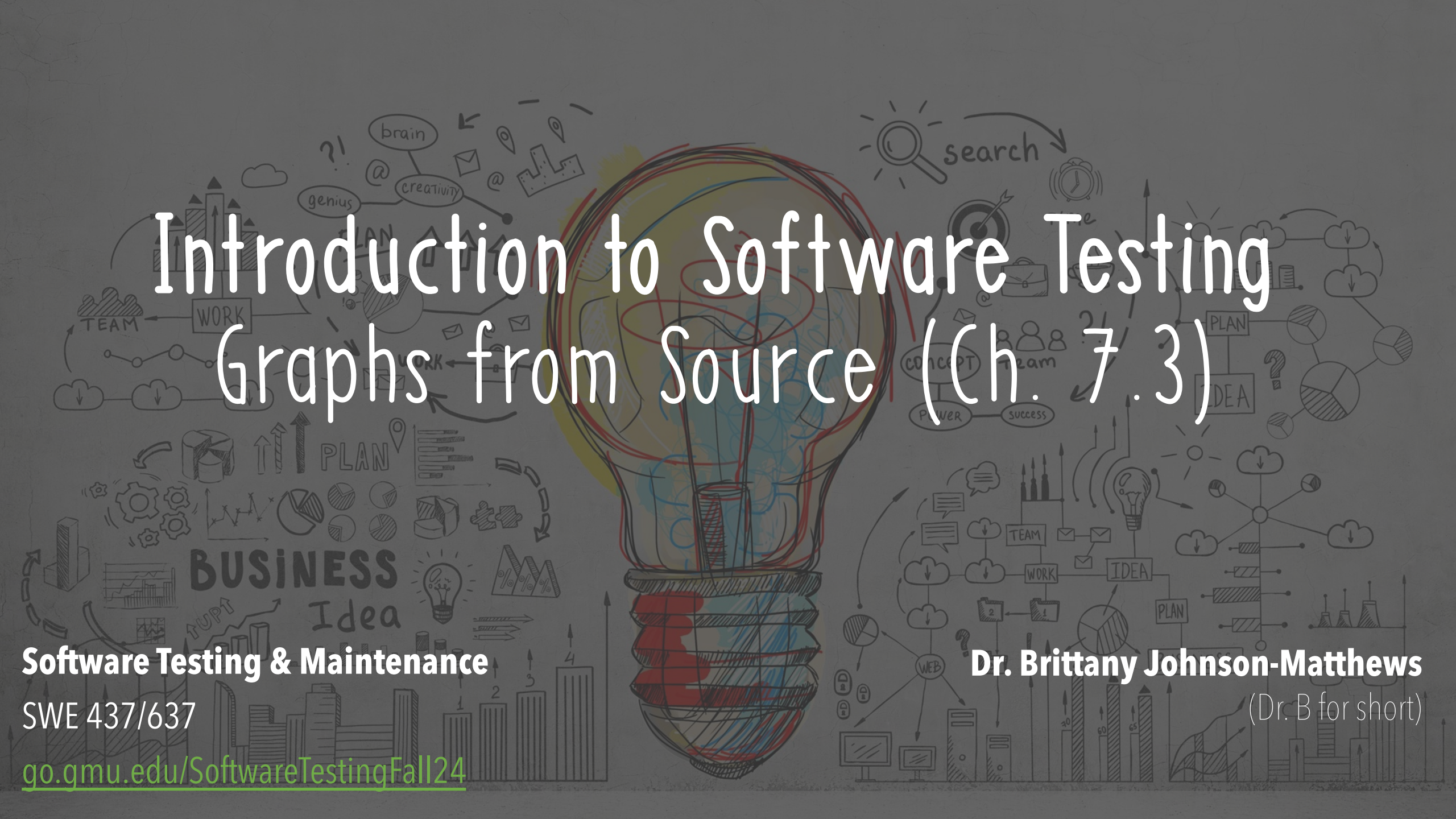


Introduction to Software Testing
Graphs from Source (Ch. 7.3)

Software Testing & Maintenance

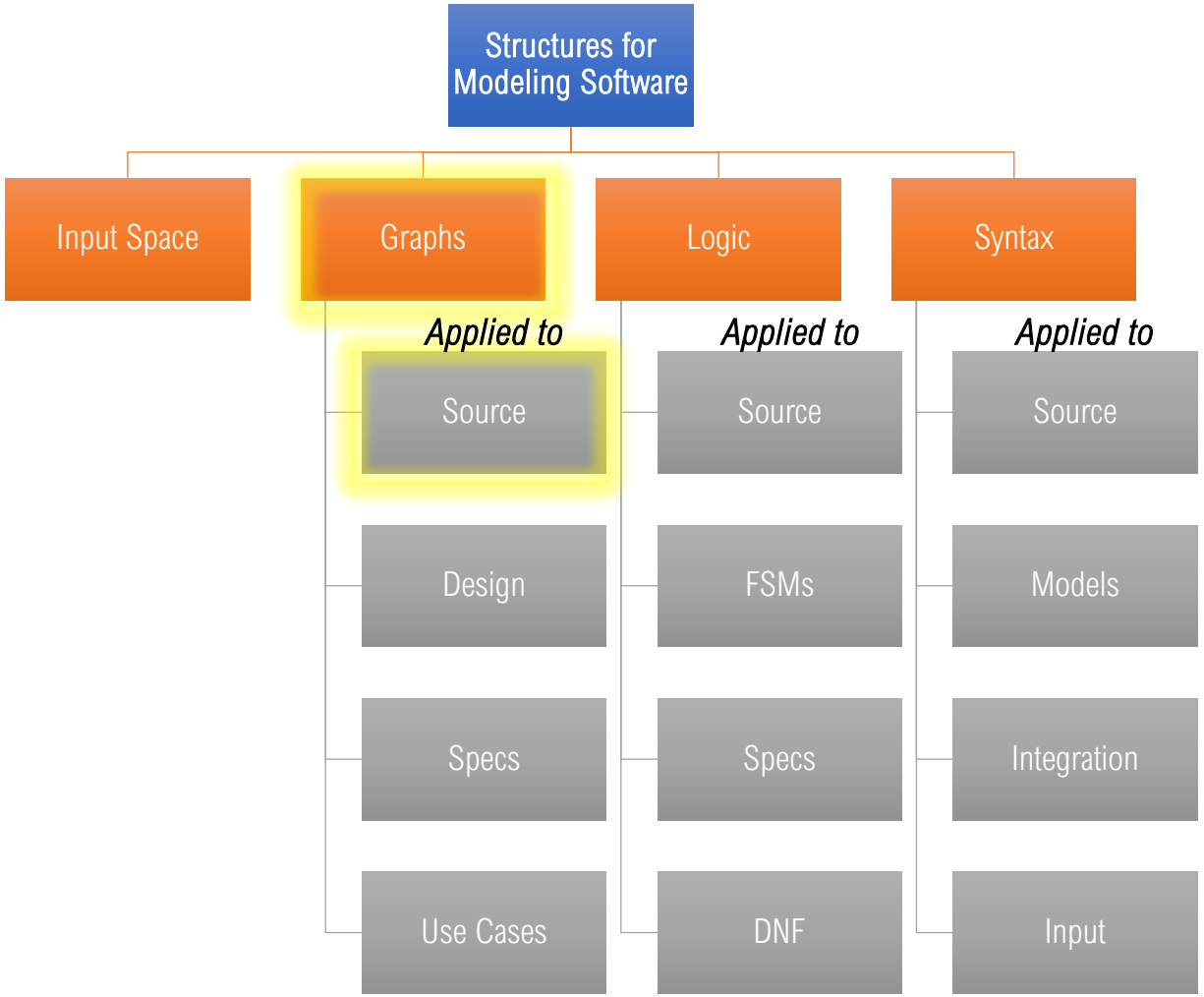
SWE 437/637

go.gmu.edu/SoftwareTestingFall24

Dr. Brittany Johnson-Matthews

(Dr. B for short)

Graph Coverage



Overview

A common application of graph criteria is to program **source**

Graph: Usually the control flow graph (CFG)

Node coverage: Execute every statement

Edge coverage: Execute every branch

Loops: Looping structures such as for loops, while loops, etc.

Control Flow Graphs

A **CFG** models all executions of a method by describing control structures

Nodes: statements or sequences of statements (*basic blocks*)

> A sequence of statements such that if the first statement is executed, all statements will be (no branches)

Edges: Transfers of control

CFGs are sometimes annotated with extra information

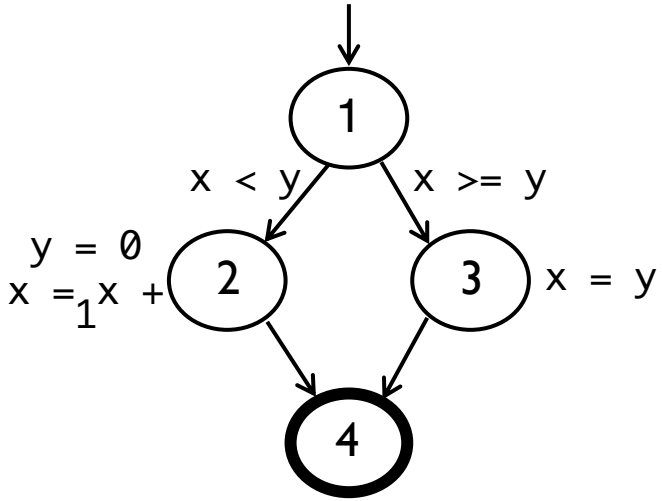
- *branch predicates*
- *defs*
- *uses*

Rules for translating statements into graphs…

CFG: The `if` Statement

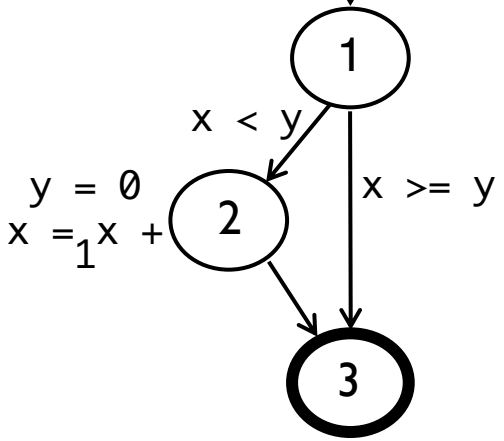
```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```

Draw the graph. Label the edges with the Java statements.




```
if (x < y)
{
   y = 0;
   x = x + 1;
}
```

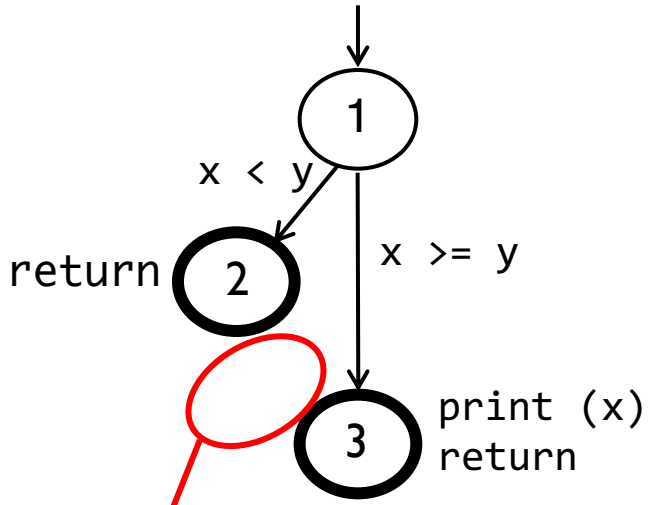
Draw the graph and label the edges.



CFG: The `if-return` Statement

```
if (x < y)
{
   return;
}
print (x);
return;
```

Draw the graph and label the edges.



No edge from node 2 to 3.
The return nodes must be distinct.

Loops

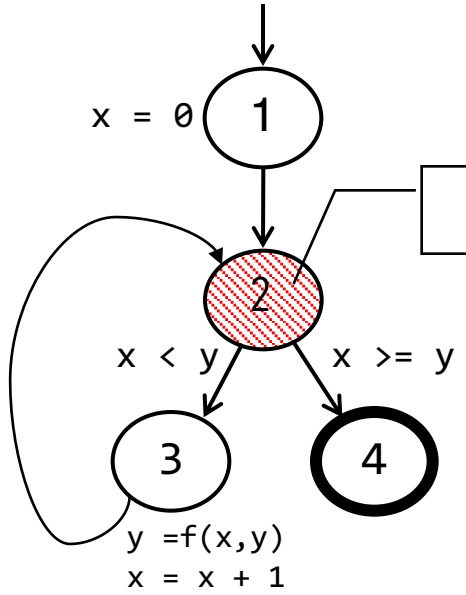
Loops require “extra” nodes to be added

Nodes that **do not** represent statements or basic blocks

CFG: while and for loops

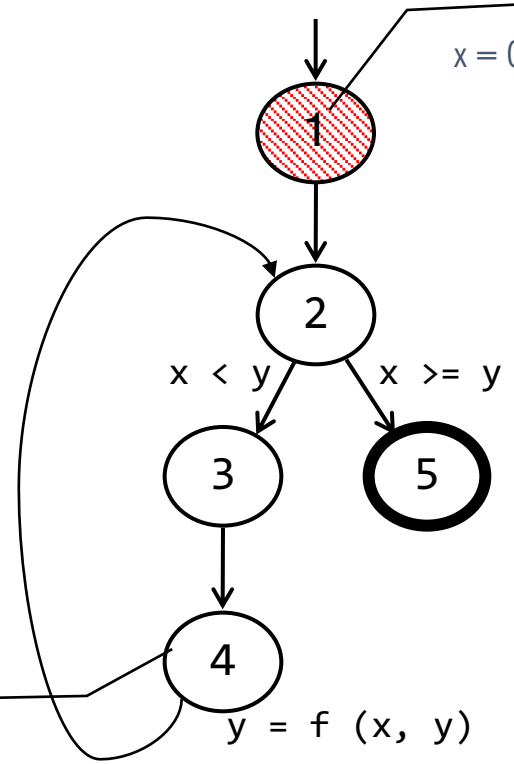
```
x = 0;
while (x < y)
{
    y = f (x, y);
    x = x + 1;
}
return (x);
```

Draw the graph and label the edges.



dummy node

implicitly initializes loop




```
for (x = 0; x < y; x++)
{
    y = f (x, y);
}
return (x);
```

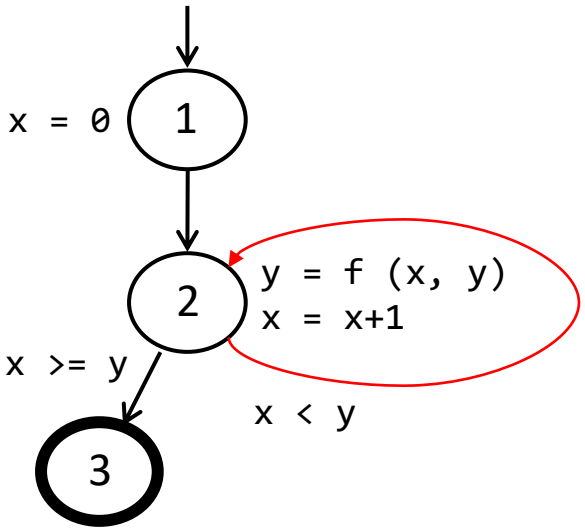
Draw the graph and label the edges.

implicitly increments loop

CFG: do loop, break, and continue

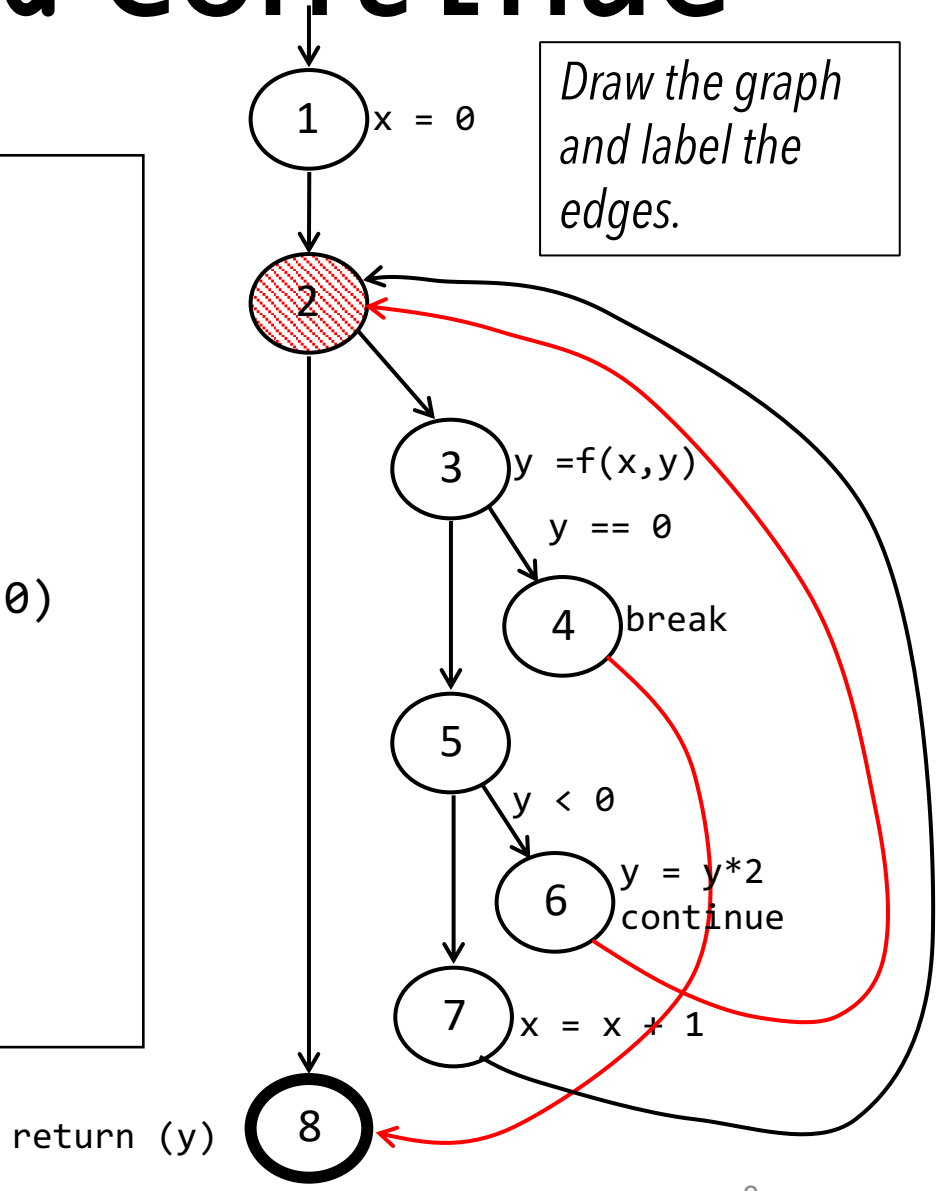
```
x = 0;
do
{
   y = f (x, y);
   x = x + 1;
} while (x < y);
return (y);
```

Draw the graph and label the edges.




```
x = 0;
while (x < y)
{
   y = f (x, y);
   if (y == 0)
   {
      break;
   } else if (y < 0)
   {
      y = y*2;
      continue;
   }
   x = x + 1;
}
return (y);
```

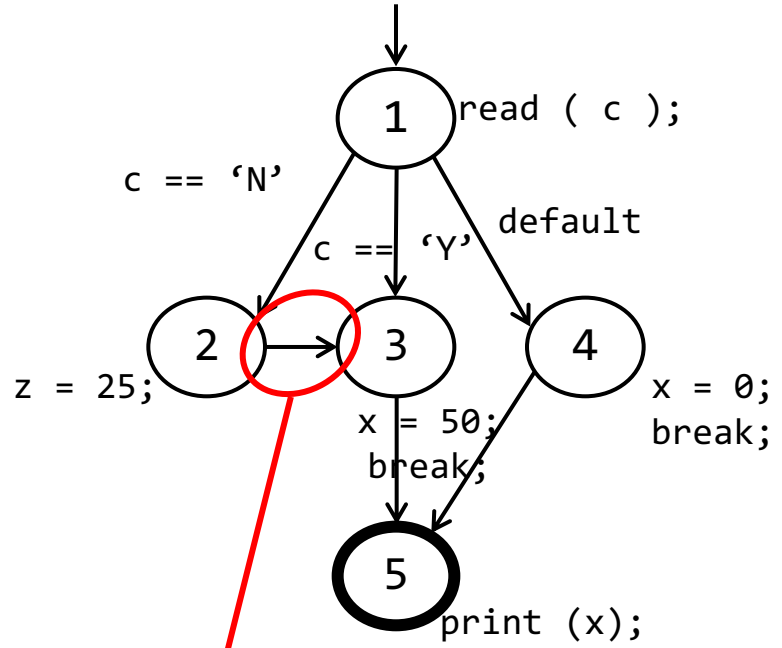
Draw the graph and label the edges.



CFG: The case (switch) Structure

```
read ( c) ;
switch ( c )
{
   case ‘N’:
      z = 25;
   case ‘Y’:
      x = 50;
      break;
   default:
      x = 0;
      break;
}
print (x);
```

Draw the graph and label the edges.

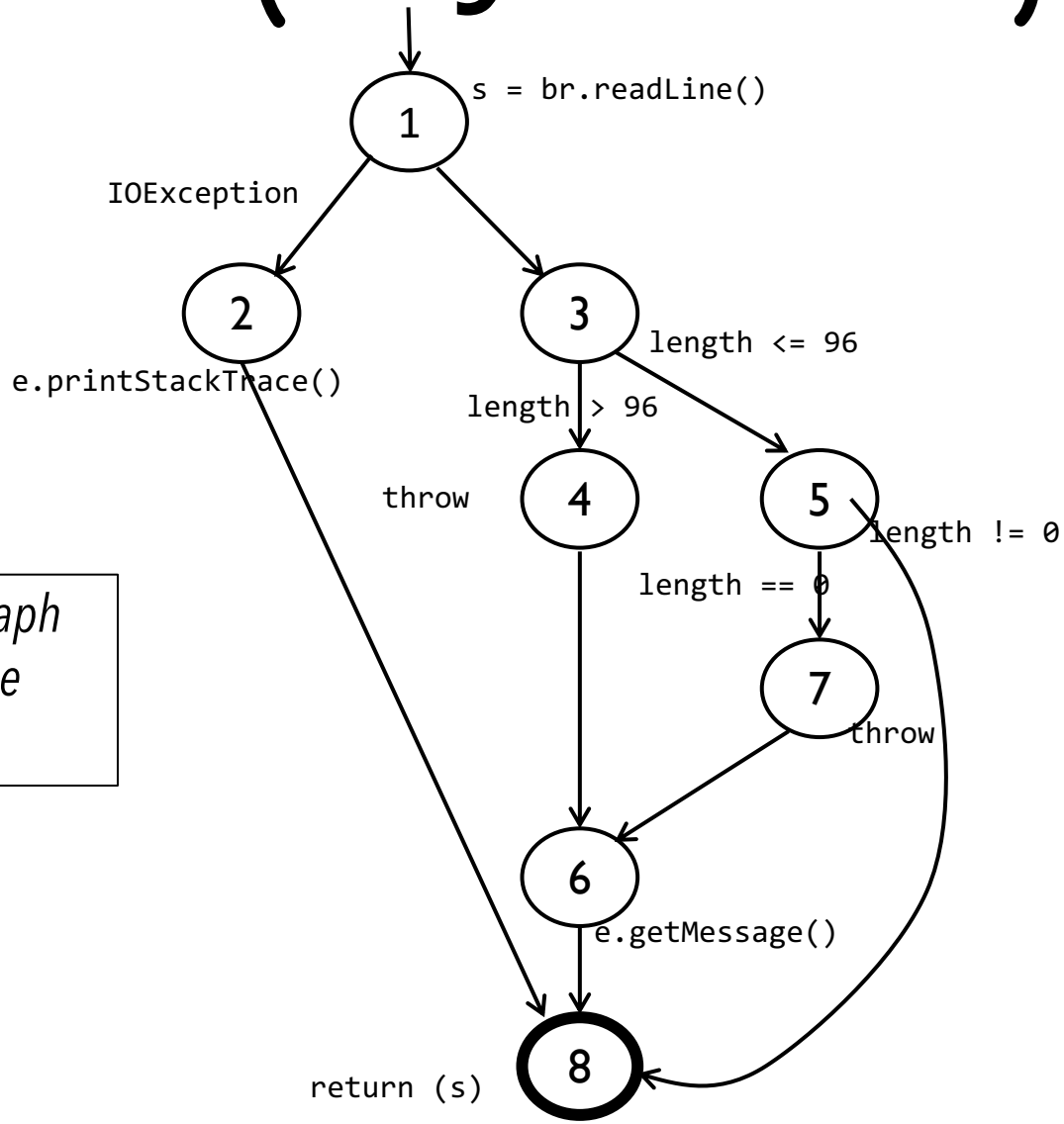


Cases without breaks fall through to the next case

CFG: Exceptions (try/catch)

```
try
{
   s = br.readLine();
   if (s.length() > 96)
      throw new Exception
         (“too long”);
   if (s.length() == 0)
      throw new Exception
         (“too short”);
} (catch IOException e) {
   e.printStackTrace();
} (catch Exception e) {
   e.getMessage();
}
return (s);
```

Draw the graph and label the edges.

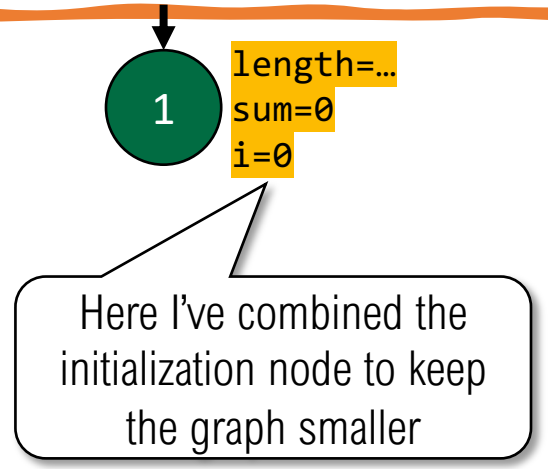


CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```

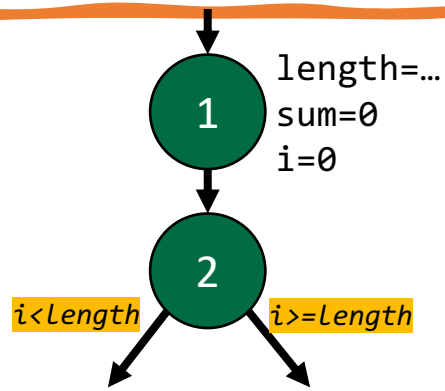
CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```



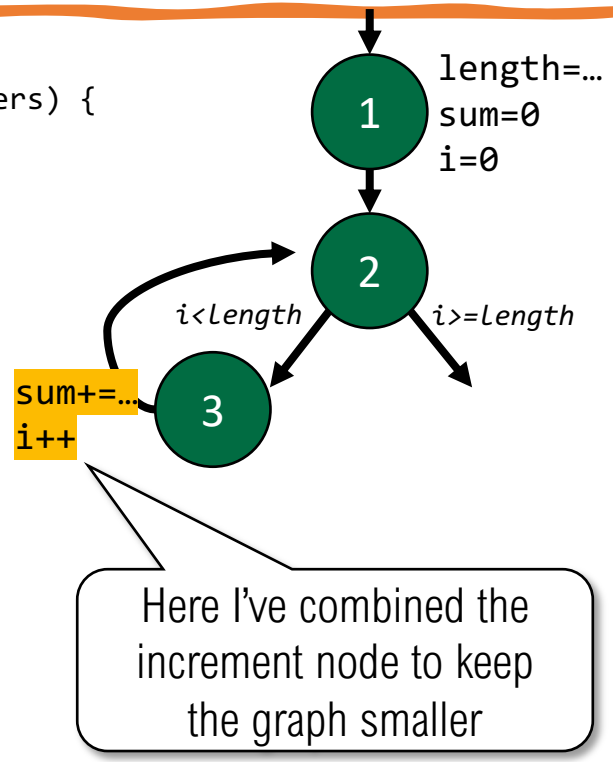
CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```



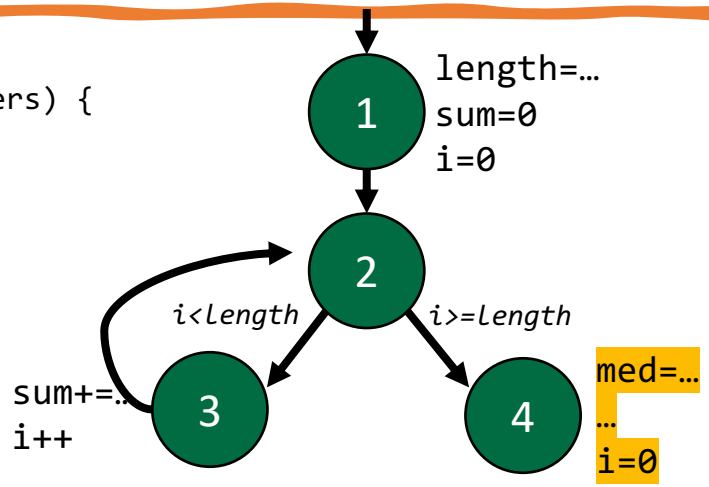
CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```



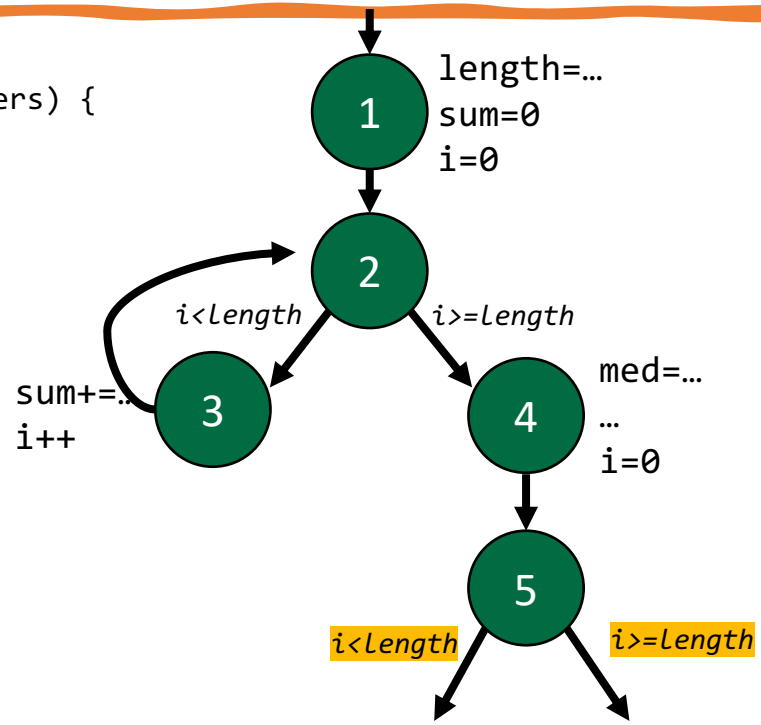
CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```



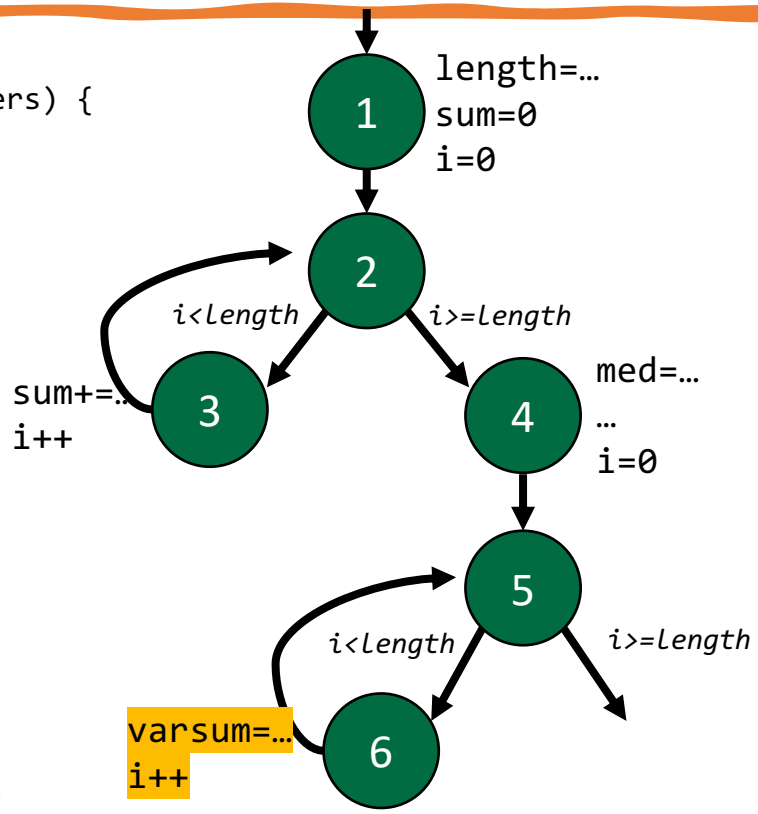
CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```



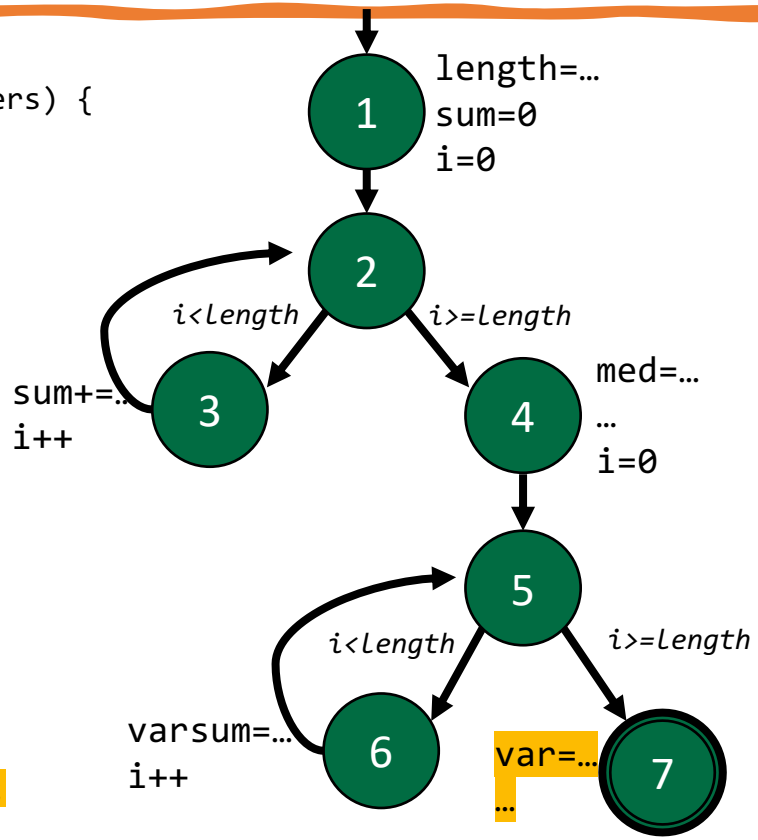
CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```

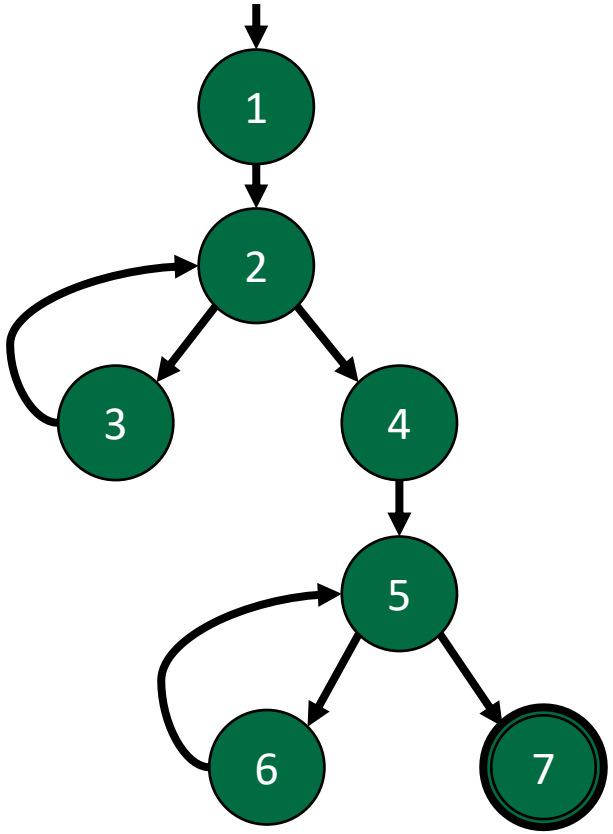


CFG Example: computeStats

```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  sum = 0;
  for (int i=0; i<length; i++) {
    sum += numbers[i];
  }
  med = numbers[length/2];
  mean = sum / (double) length;

  varsum = 0;
  for (int i=0; i<length; i++) {
    varsum = varsum  + ((numbers[i] - mean)
      * (numbers[i] - mean));
  }
  var = varsum / (length - 1.0);
  sd  = Math.sqrt(var);

  System.out.println("length:   " + length);
  System.out.println("mean:     " + mean);
  System.out.println("median:   " + med);
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);

}
```



TRs and Test Paths (Edge Coverage)

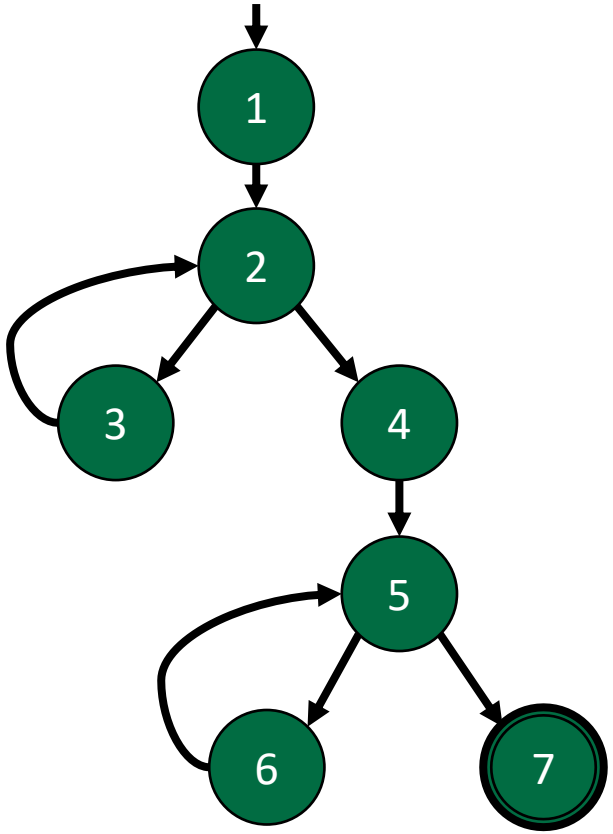


Edge Coverage TRs

[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

TRs and Test Paths (Edge Coverage)



Edge Coverage TRs

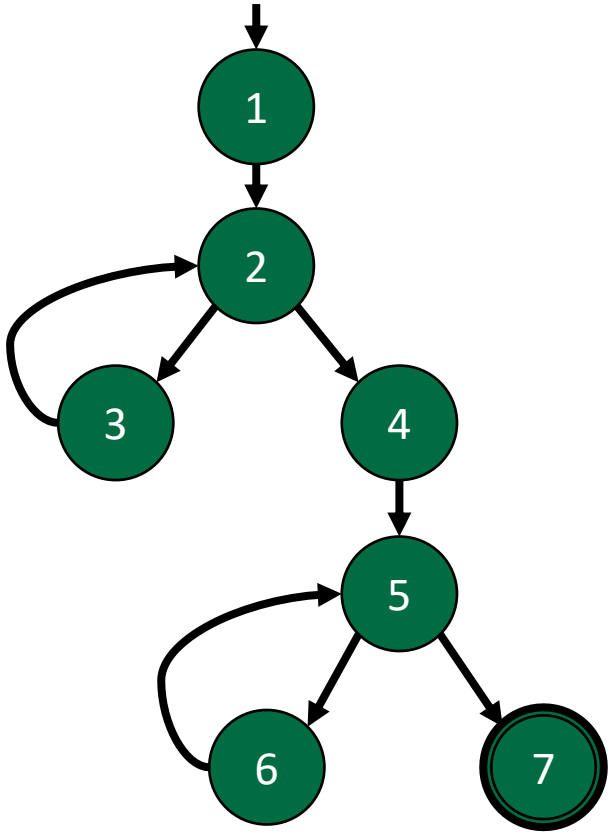
[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

[1,2

Start at the initial node

TRs and Test Paths (Edge Coverage)



Edge Coverage TRs

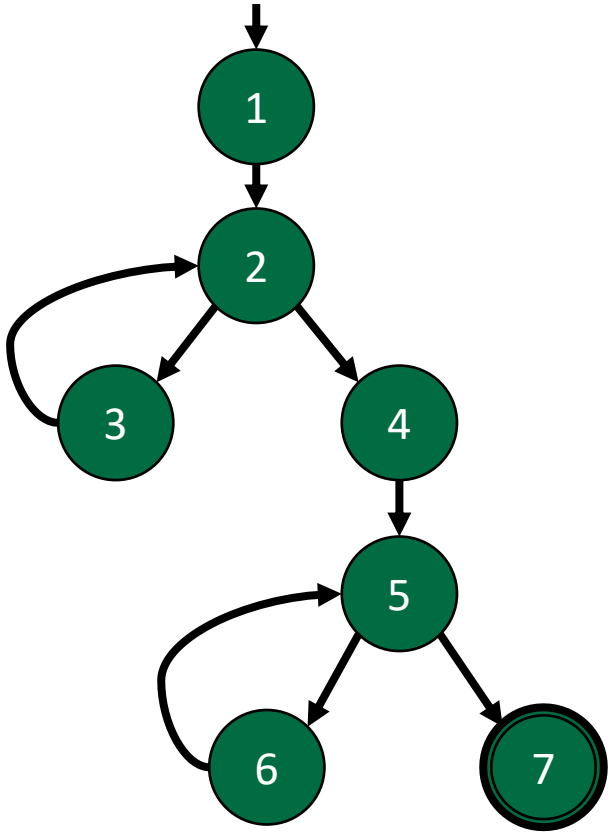
[1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3

Pick an edge that increases coverage (tip: take the loop first to maximize the coverage from this test path)

TRs and Test Paths (Edge Coverage)



Edge Coverage TRs

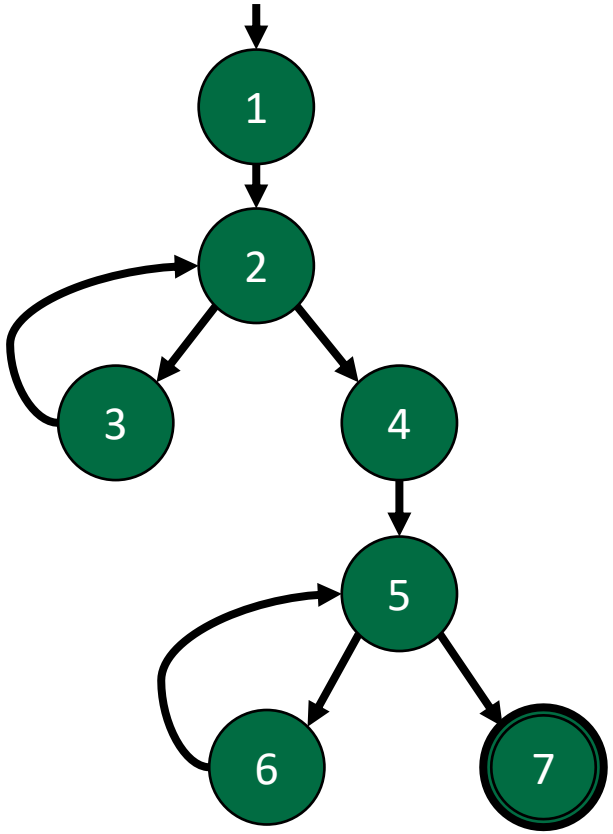
[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3,2

Continue to pick edges that increase coverage

TRs and Test Paths (Edge Coverage)



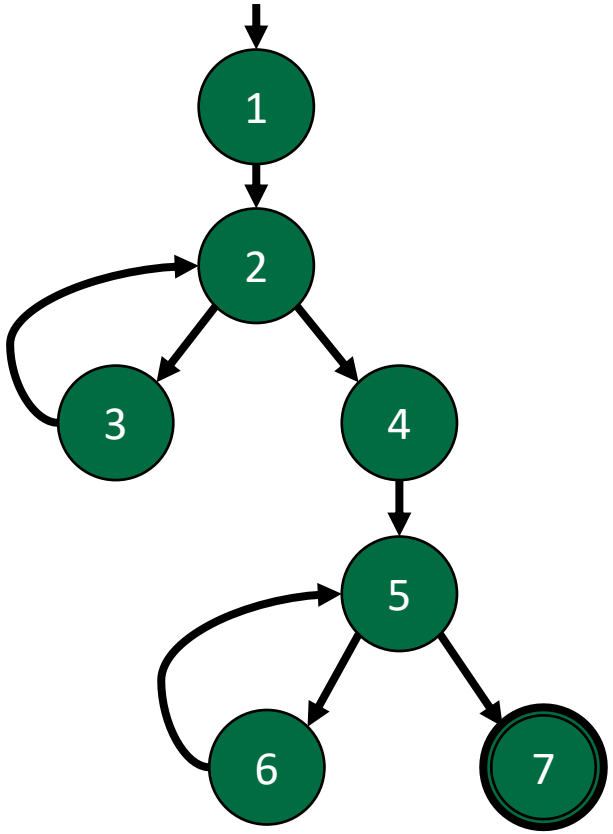
Edge Coverage TRs

[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3,2,4

TRs and Test Paths (Edge Coverage)



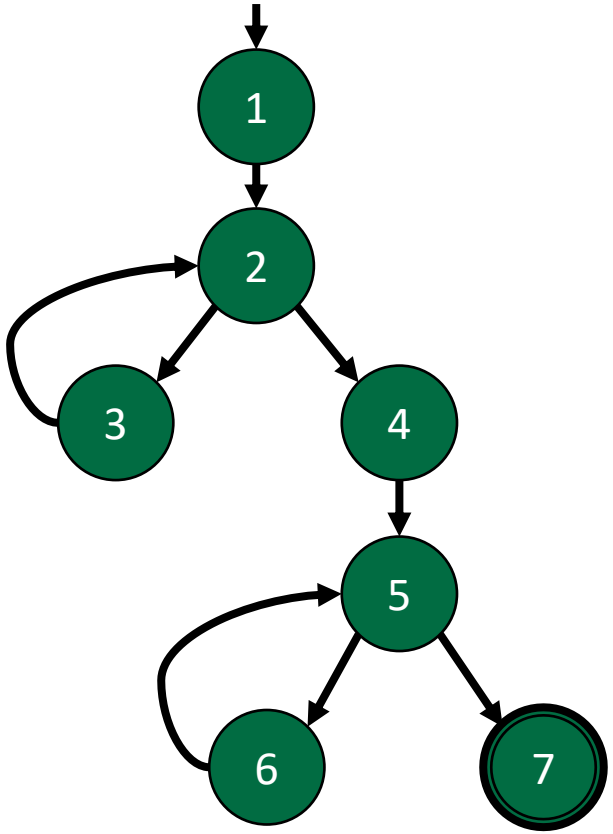
Edge Coverage TRs

[1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3,2,4,5

TRs and Test Paths (Edge Coverage)



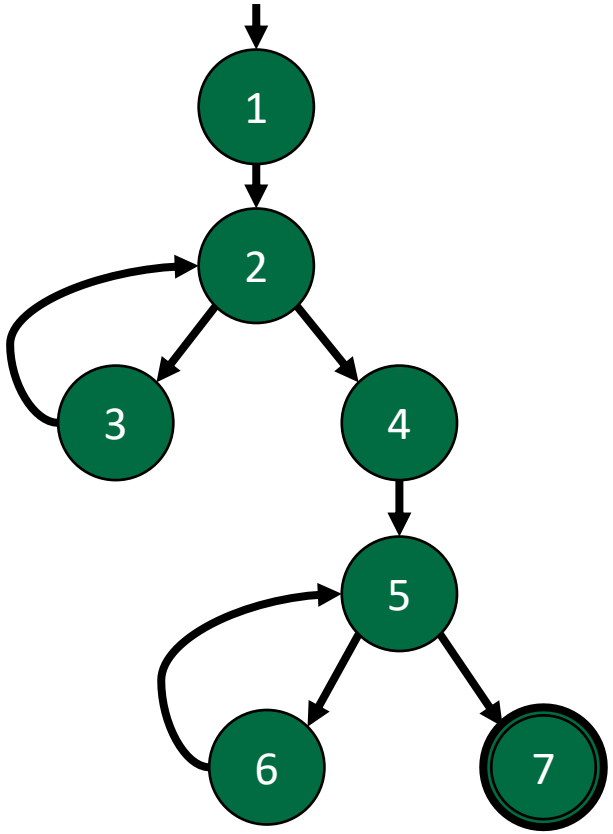
Edge Coverage TRs

[1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]

Test paths

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

TRs and Test Paths (Edge Coverage)



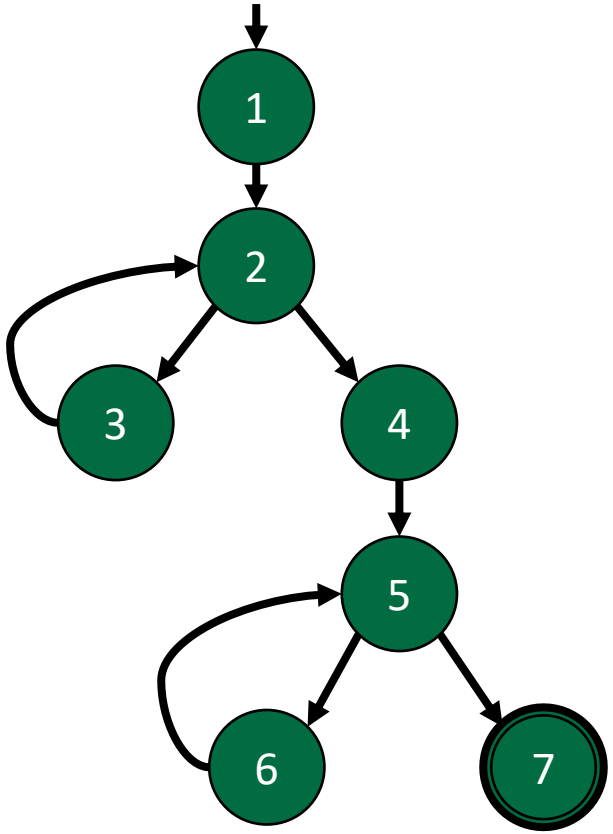
Edge Coverage TRs

[1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3,2,4,5,6,5,7]

TRs and Test Paths (Edge Coverage)



Edge Coverage TRs

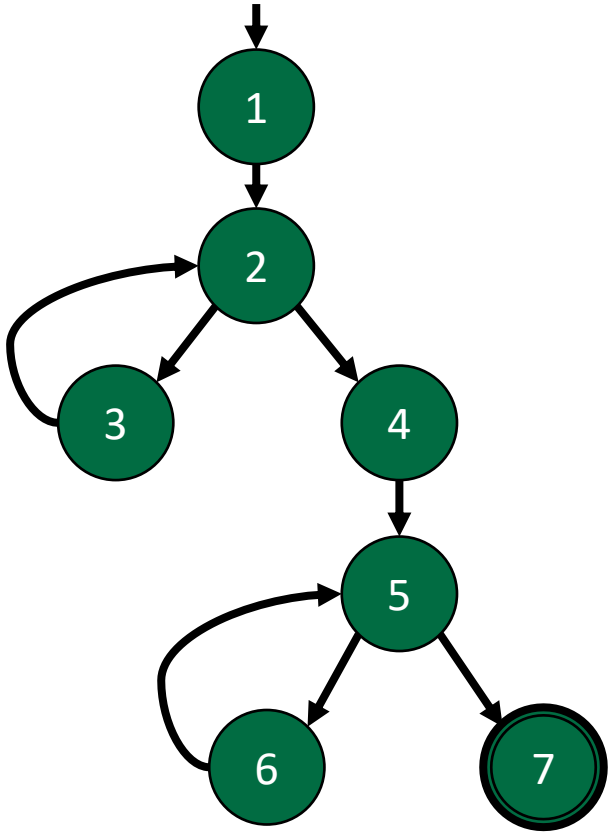
[1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3,2,4,5,6,5,7]

Edge coverage is satisfied with 1 test path

TRs and Test Paths (Edge-Pair Coverage)

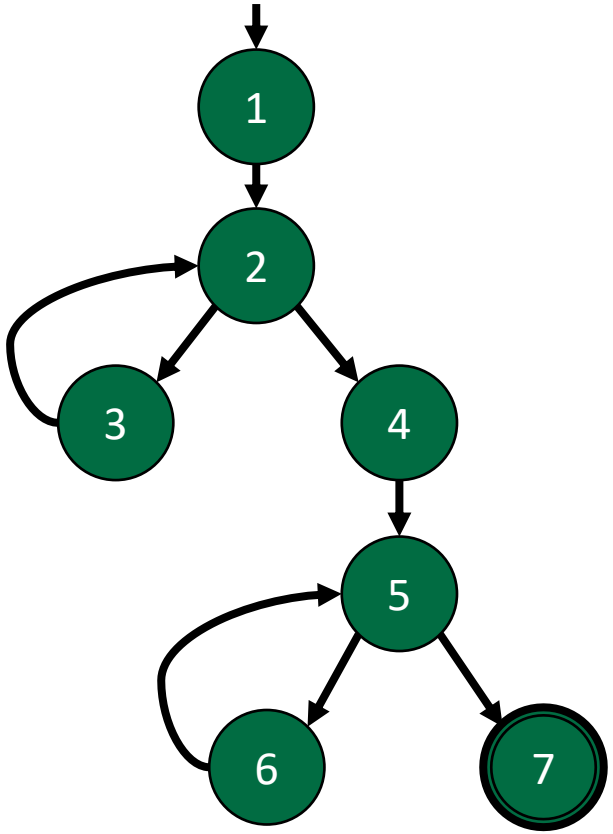


Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5],
[3,2,3], [3,2,4],
[4,5,6], [4,5,7], [5,6,5], [6,5,6],
[6,5,7]

Test paths

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

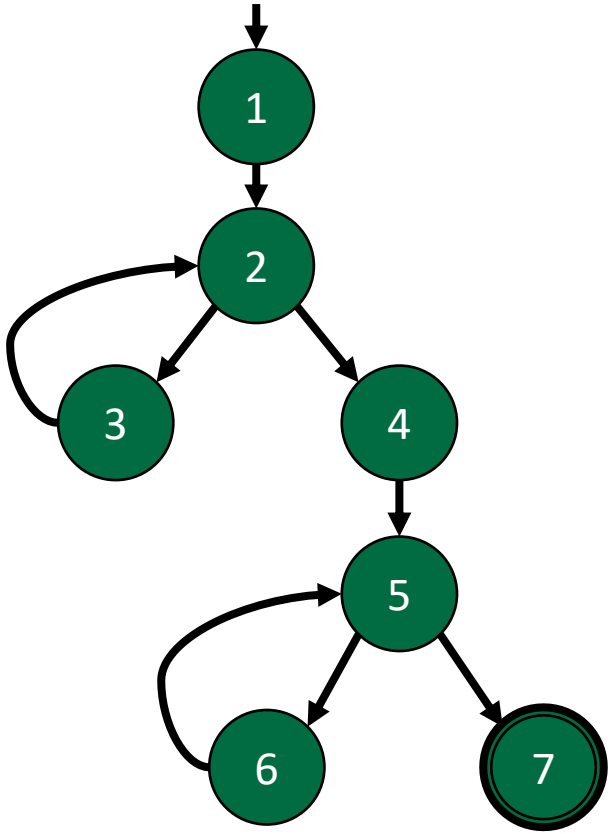
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3

Start at the initial node and pick a starting edge-pair

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

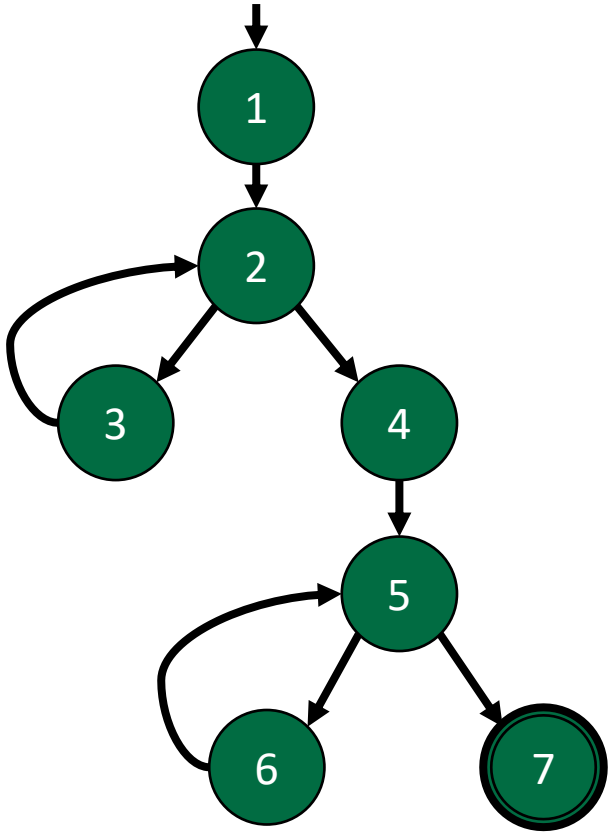
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2

Select an edge that increases edge-pair coverage

TRs and Test Paths (Edge-Pair Coverage)



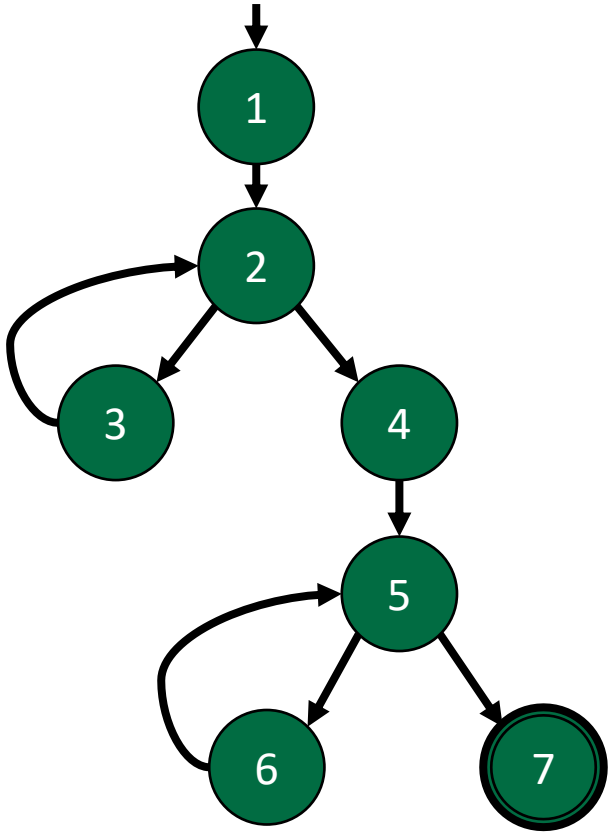
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

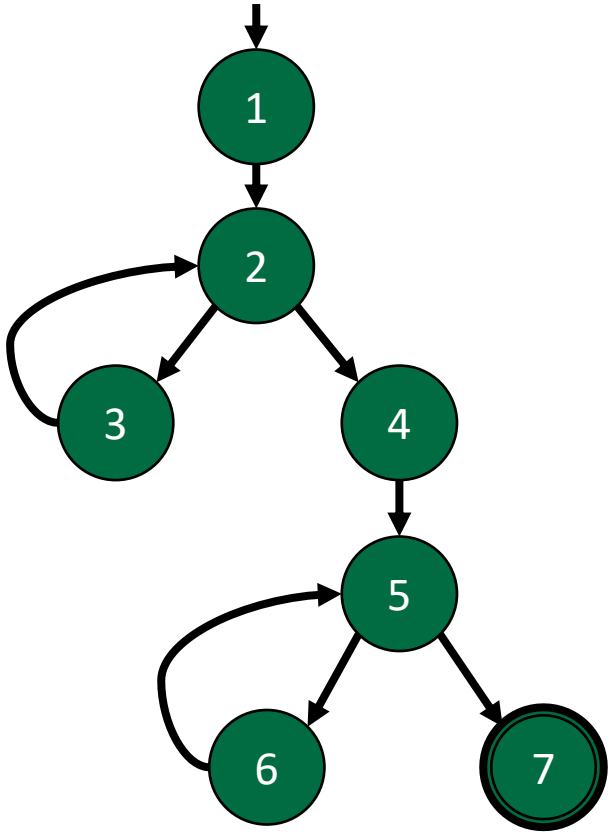
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2

It's not always possible to increase coverage with every selected edge

TRs and Test Paths (Edge-Pair Coverage)



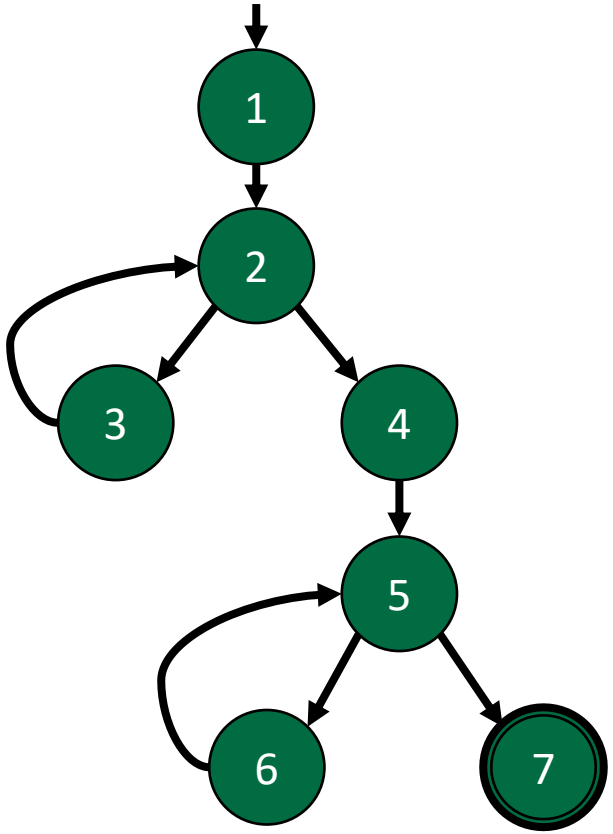
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

TRs and Test Paths (Edge-Pair Coverage)



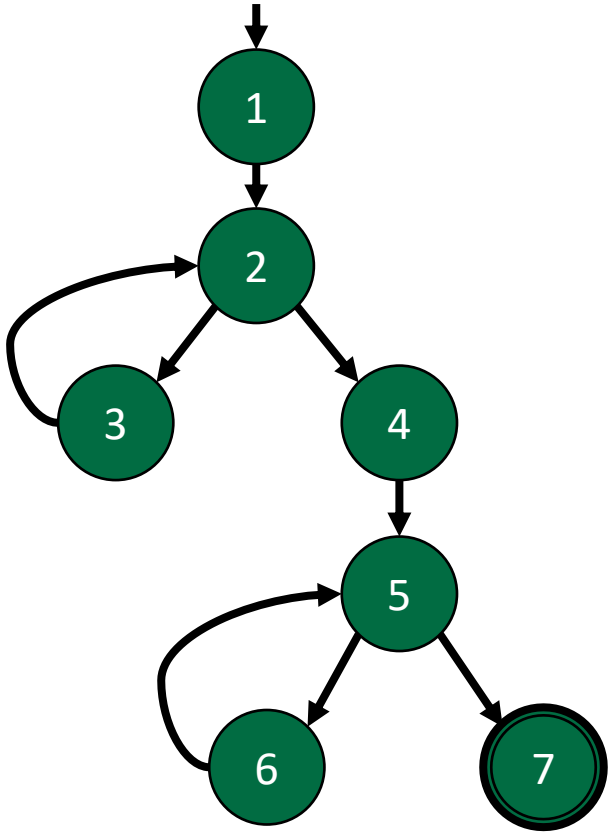
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5

TRs and Test Paths (Edge-Pair Coverage)



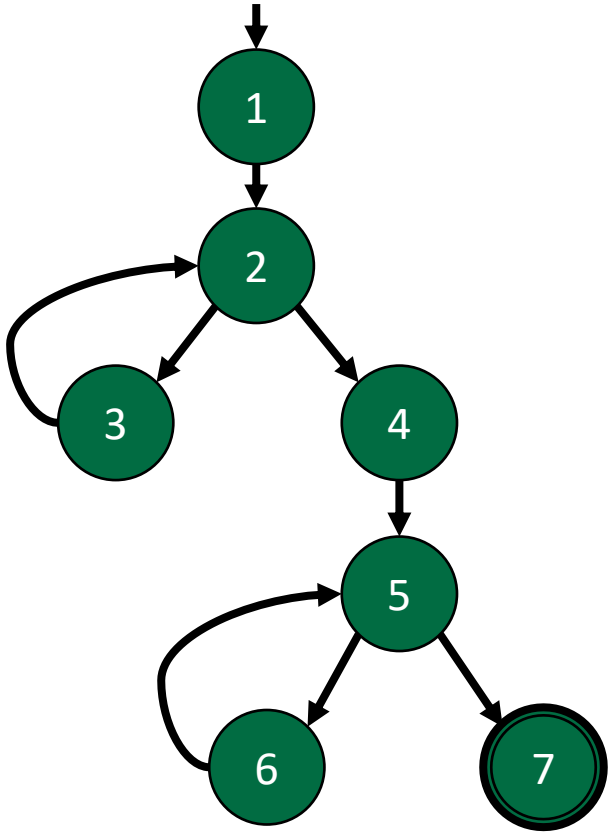
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

TRs and Test Paths (Edge-Pair Coverage)



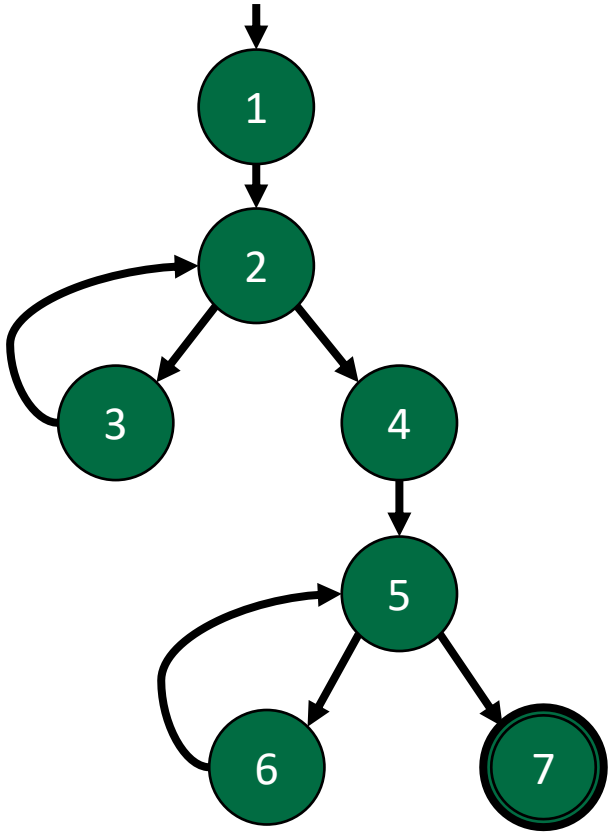
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

TRs and Test Paths (Edge-Pair Coverage)



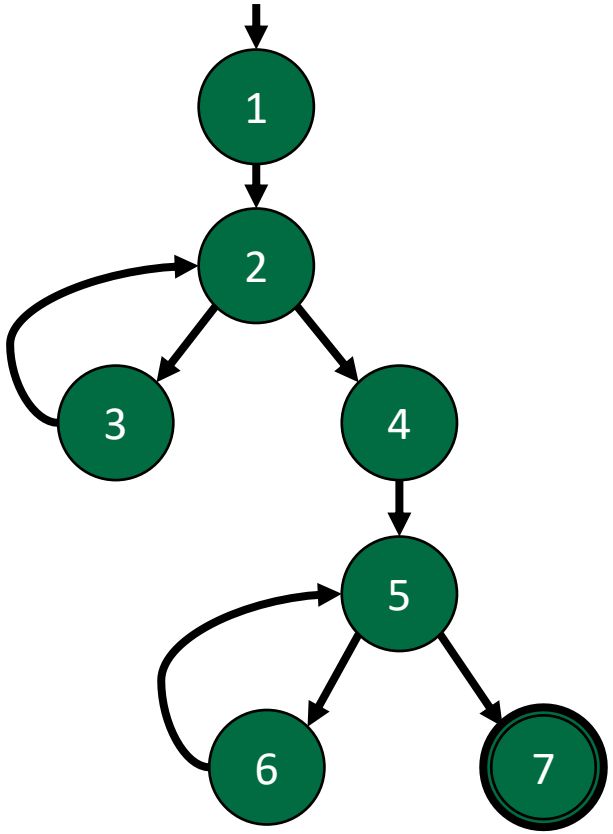
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

TRs and Test Paths (Edge-Pair Coverage)



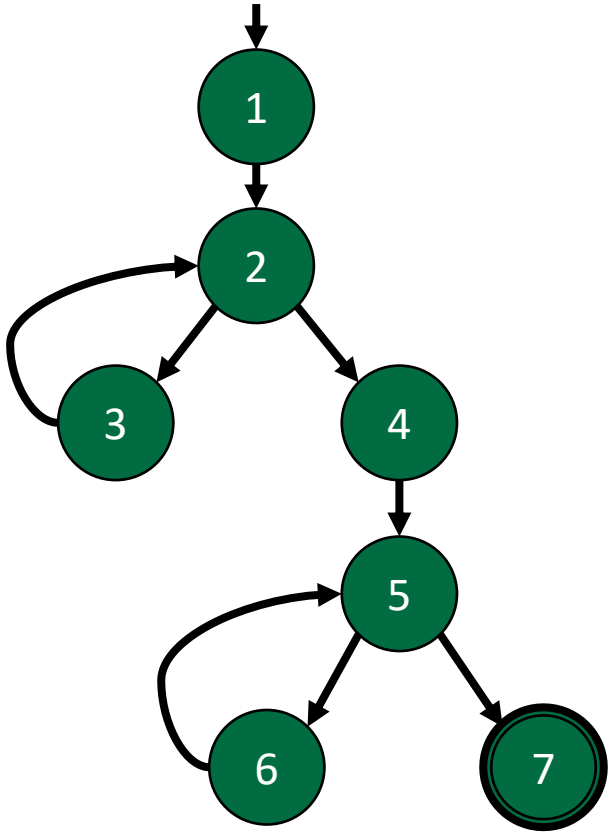
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

TRs and Test Paths (Edge-Pair Coverage)



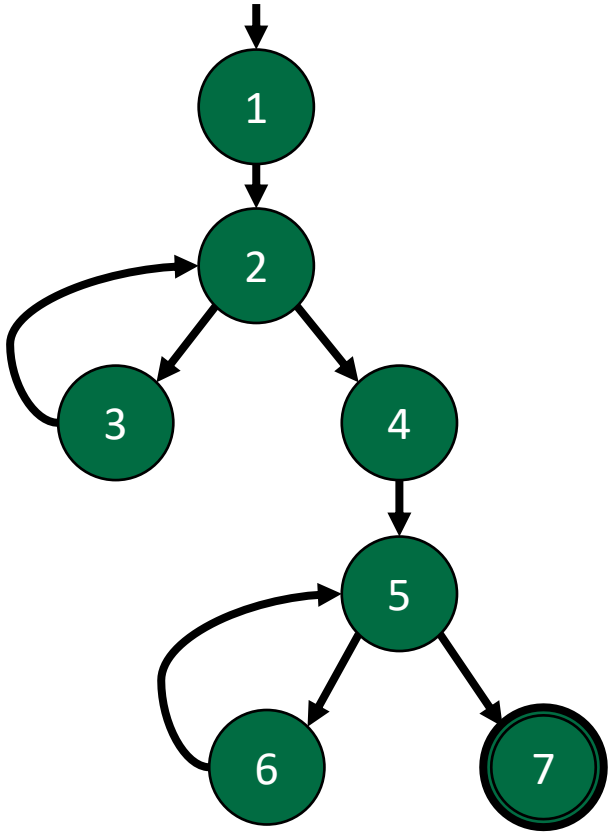
Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

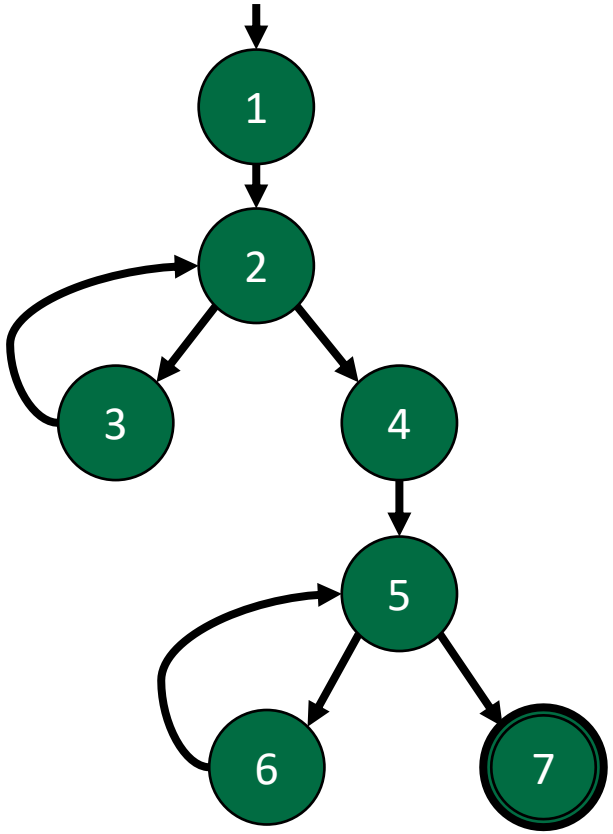
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

We need another test path to achieve edge-pair coverage

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

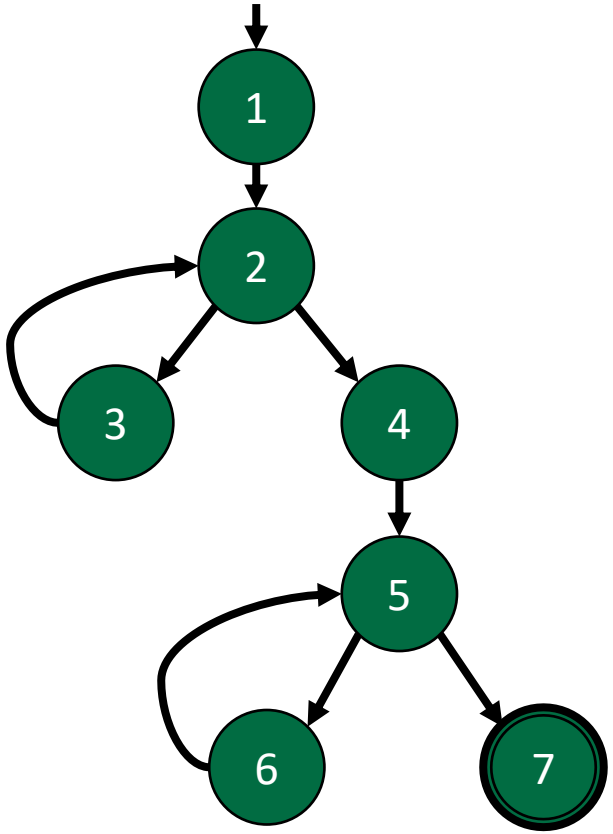
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

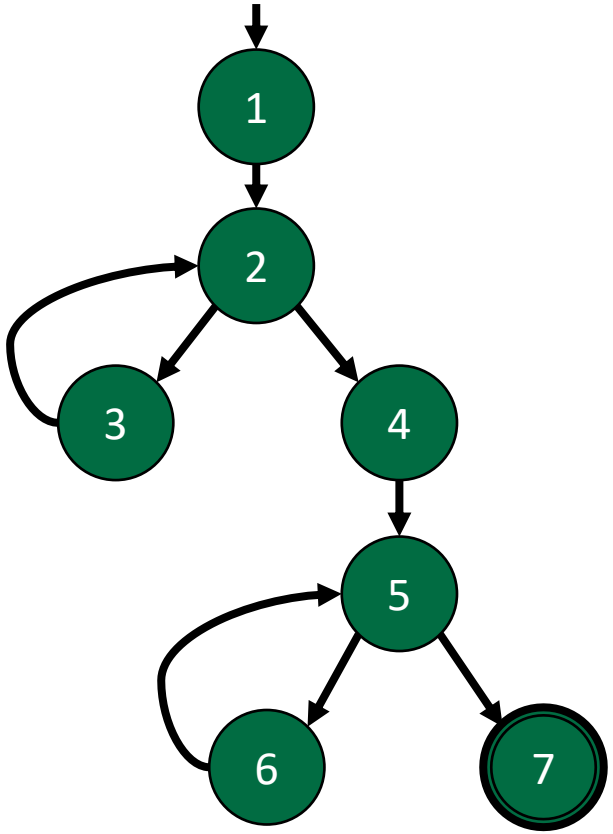
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

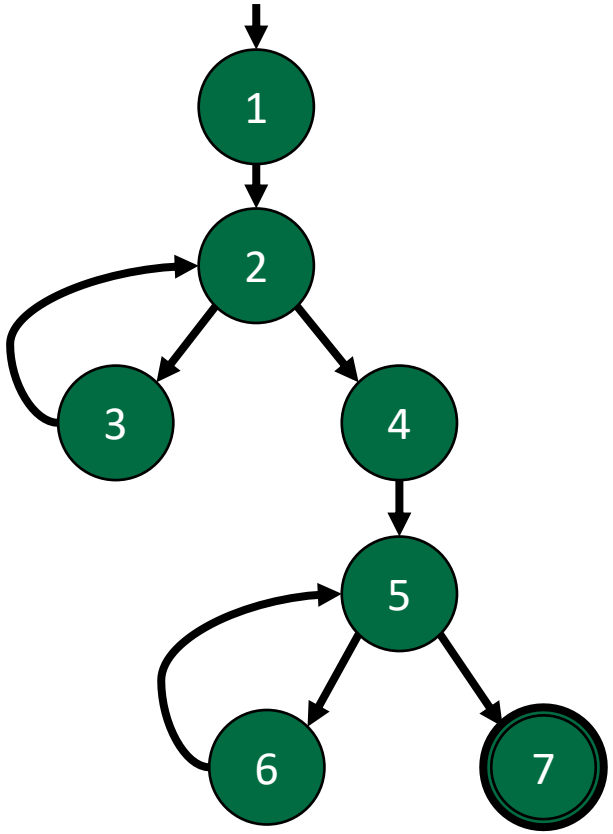
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

TRs and Test Paths (Edge-Pair Coverage)



Edge-Pair TRs

[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

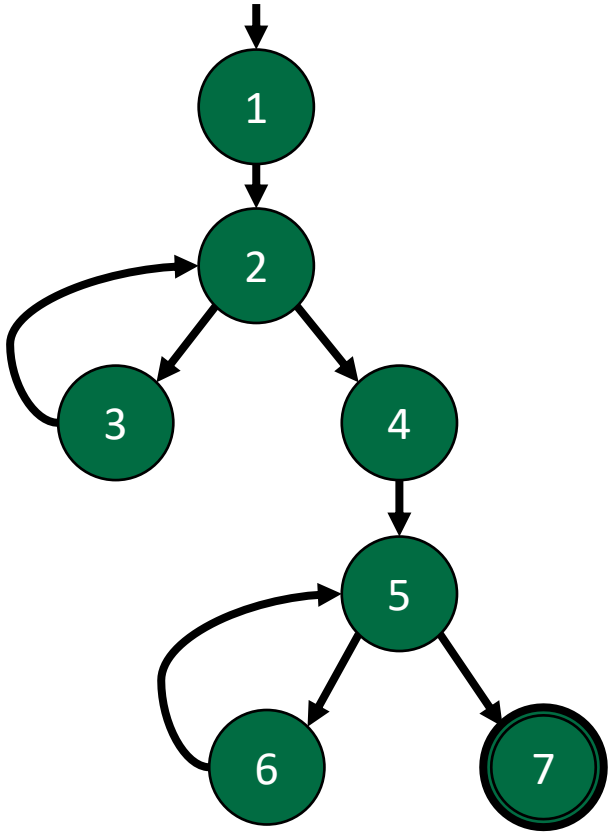
Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

Edge-pair coverage is satisfied with 2 test paths

TRs and Test Paths (Prime Path Coverage)

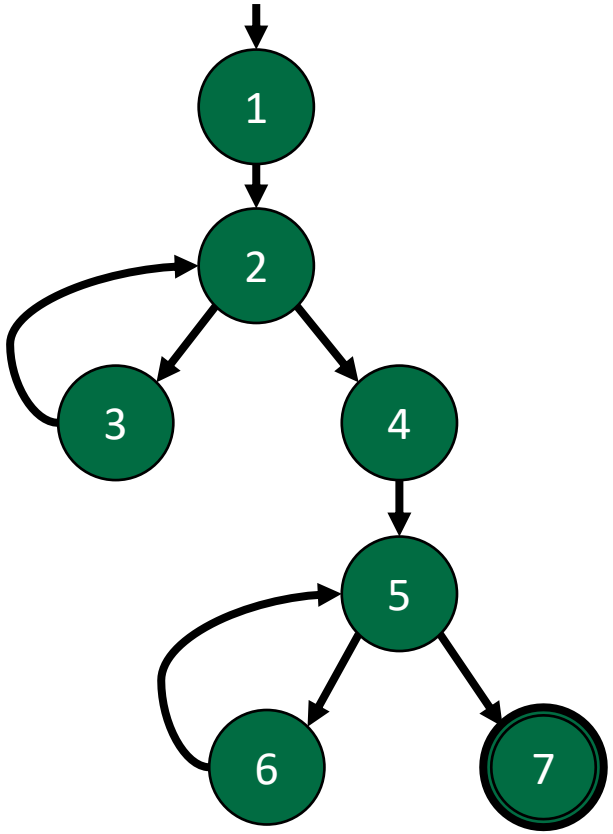


Prime Path TRs

[1,2,3], [1,2,4,5,6],
[1,2,4,5,7], [2,3,2], [3,2,3],
[3,2,4,5,6], [3,2,4,5,7], [5,6,5],
[6,5,6], [6,5,7]

Test paths

TRs and Test Paths (Prime Path Coverage)



Prime Path TRs

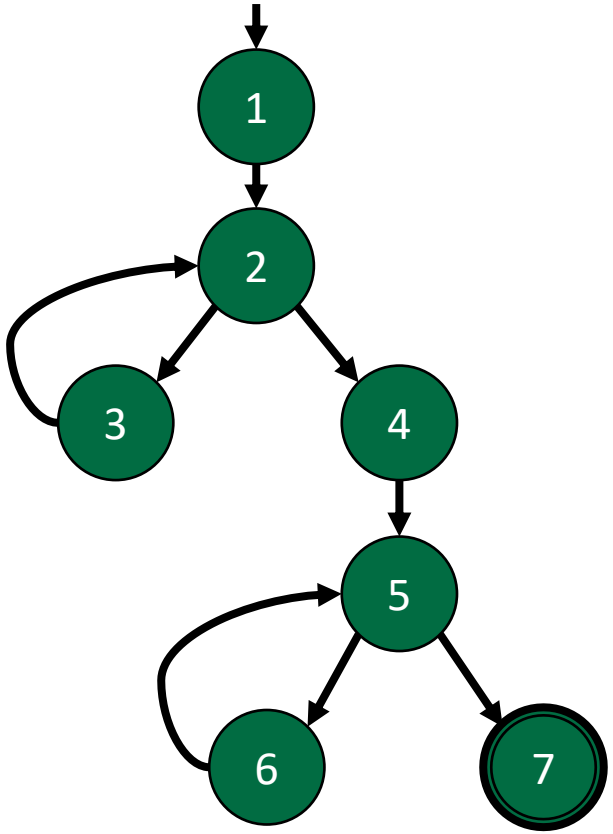
[1,2,3], [1,2,4,5,6], [1,2,4,5,7], [2,3,2], [3,2,3], [3,2,4,5,6], [3,2,4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

Tip: take a "greedy algorithm" approach and try to maximize the coverage of each test path

TRs and Test Paths (Prime Path Coverage)



Prime Path TRs

[1,2,3], [1,2,4,5,6], [1,2,4,5,7], [2,3,2], [3,2,3], [3,2,4,5,6], [3,2,4,5,7], [5,6,5], [6,5,6], [6,5,7]

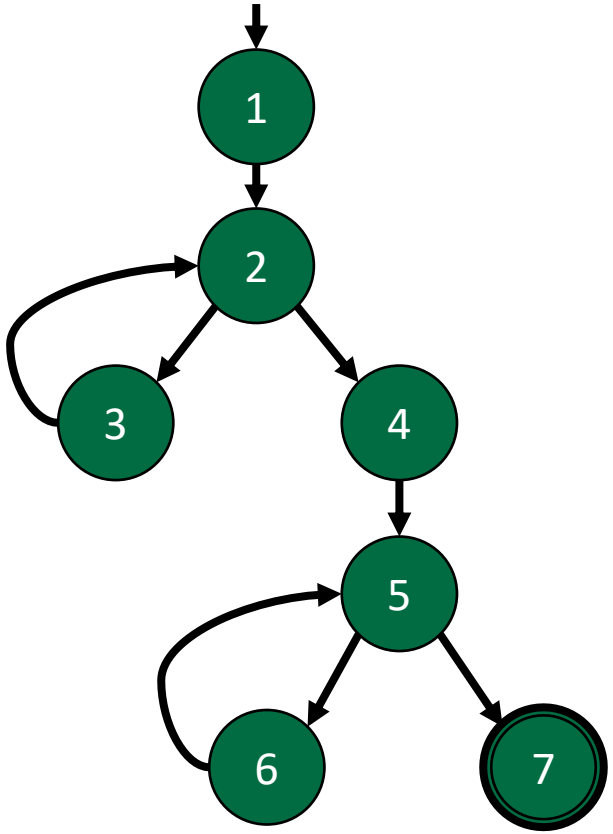
Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

Add additional test paths to capture the remaining TRs

TRs and Test Paths (Prime Path Coverage)



Prime Path TRs

[1,2,3], [1,2,4,5,6],
[1,2,4,5,7], [2,3,2], [3,2,3],
[3,2,4,5,6], [3,2,4,5,7], [5,6,5],
[6,5,6], [6,5,7]

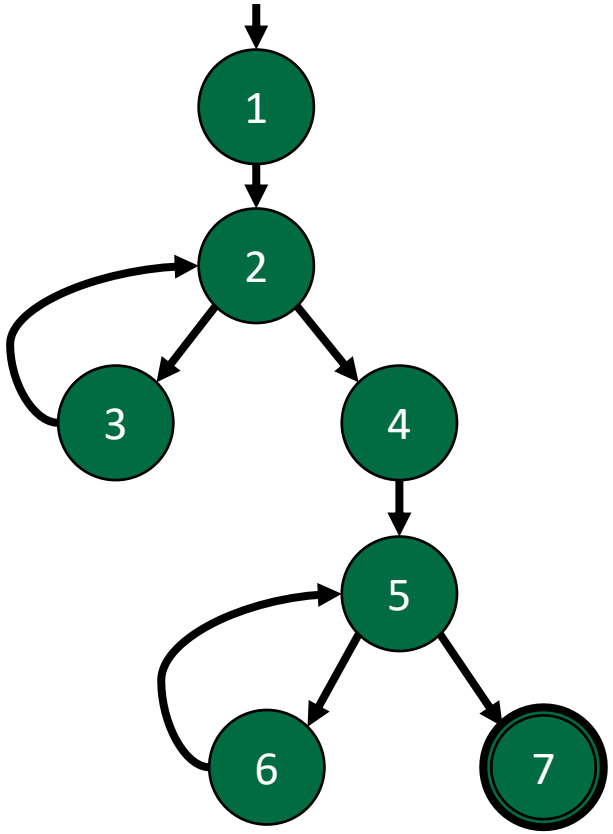
Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

[1,2,4,5,6,5,7]

TRs and Test Paths (Prime Path Coverage)



Prime Path TRs

[1,2,3], [1,2,4,5,6], [1,2,4,5,7], [2,3,2], [3,2,3], [3,2,4,5,6], [3,2,4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

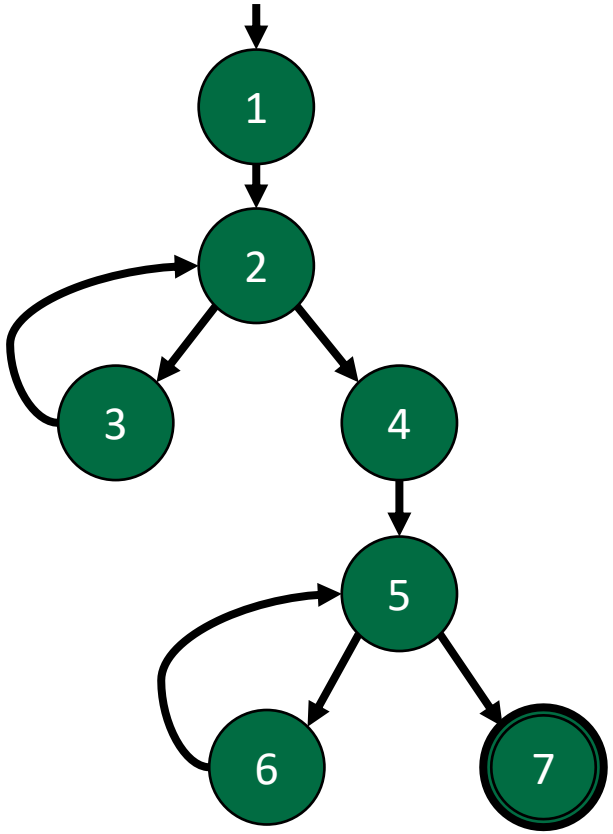
[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

[1,2,4,5,6,5,7]

[1,2,3,2,4,5,7]

TRs and Test Paths (Prime Path Coverage)



Prime Path TRs

[1,2,3], [1,2,4,5,6],
[1,2,4,5,7], [2,3,2], [3,2,3],
[3,2,4,5,6], [3,2,4,5,7], [5,6,5],
[6,5,6], [6,5,7]

Test paths

[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

[1,2,4,5,6,5,7]

[1,2,3,2,4,5,7]

Summary

Applying the graph test criteria to **control flow graph** is relatively straightforward

- Most of the developmental **research** work was done with CFGs

A few **subtle decisions** must be made to translate control structures into the graph

Some tools will assign each statement to a **unique node**

- These slides and the book use **basic blocks**
- Coverage is the same, although the **bookkeeping** will differ