Intro to Software Testing

chapter 7.3.1

Graph Coverage from Source Code

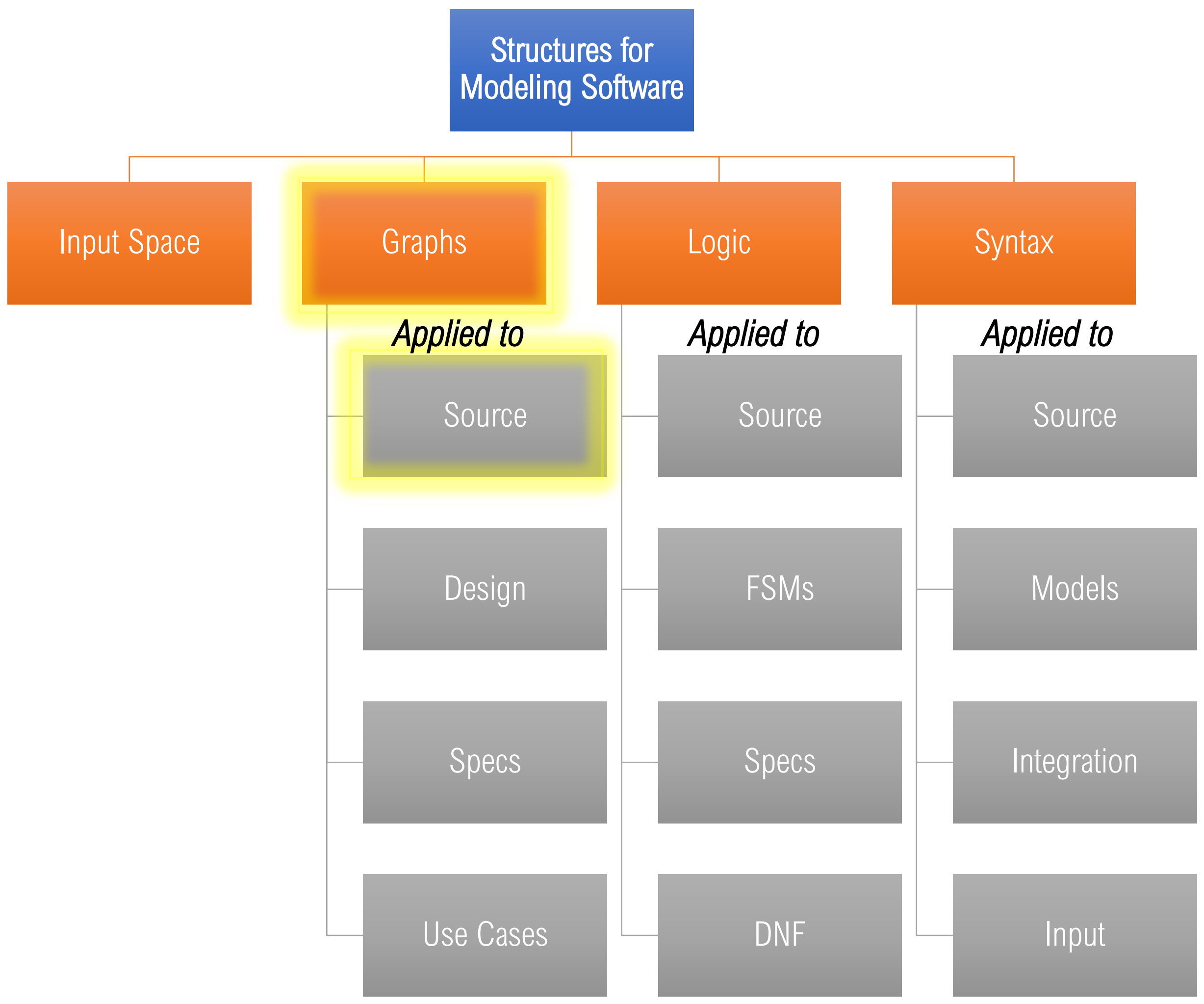
Dr. Brittany Johnson-Matthews

(Dr. B for short)

https://go.gmu.edu/SWE637

Provided by Bob Kurtz

Graph Coverage



Overview

Graph criteria are often applied to program source code

The graph is generally the control flow graph (CFG)

Node coverage requires execution of every statement

Edge coverage requires execution of every branch

Data flow coverage requires augmenting the CFG, where *defs* are variable assignments and *uses* are variable references

Control Flow Graphs

A CFG models execution of a method by describing control flow structures

A *node* contains a statement or sequence of statements such that if the first statement in the sequence is executed, all statements in the sequence are executed (a “basic block”)

An *edge* is a transfer of control (decision)

CFGs may be annotated with extra information

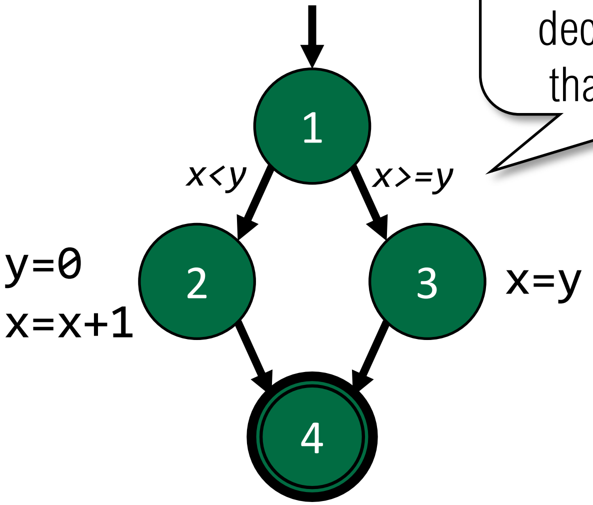
- Variable defs
- Variable uses
- Source code

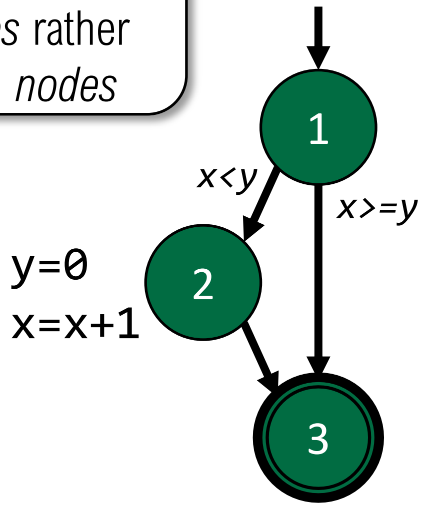
CFG Example: if

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

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

Note that the text chooses to annotate decision *edges* rather than decision *nodes*



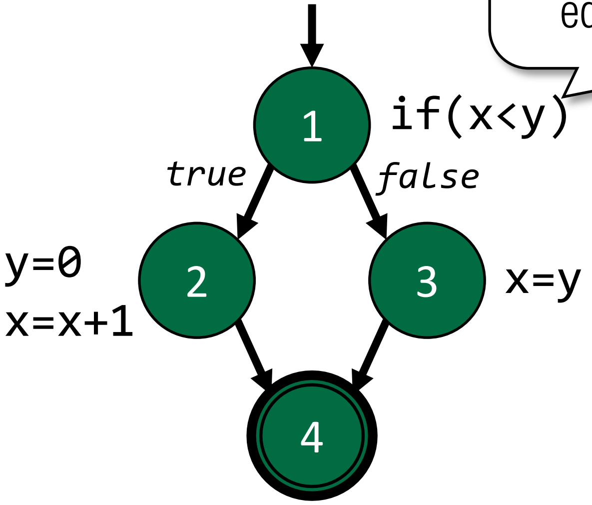


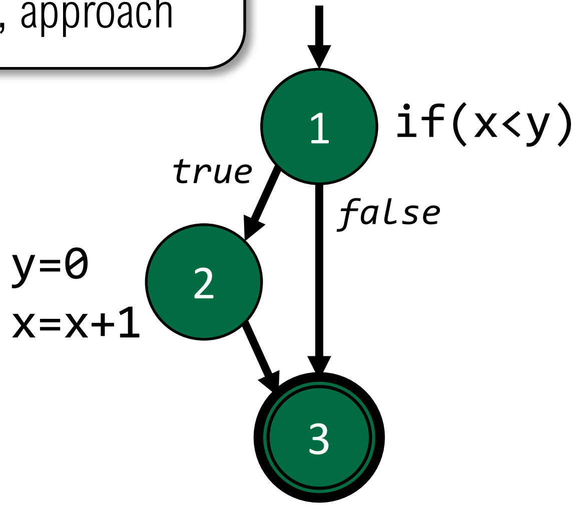
CFG Example: if

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

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

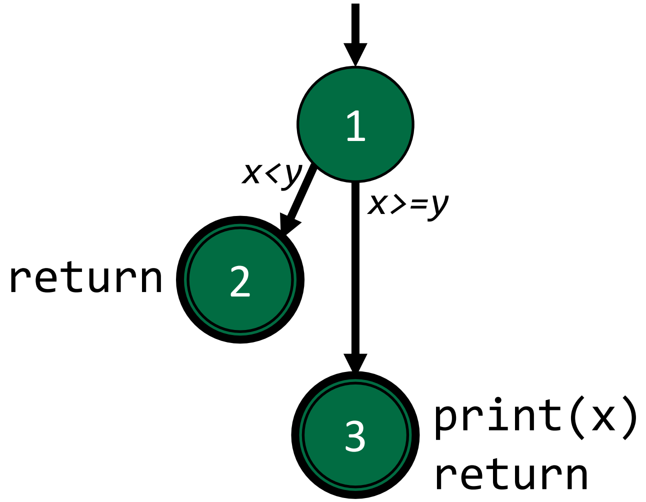
Annotating decision nodes is an alternative, and equally valid, approach





CFG Example: `if-return`

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



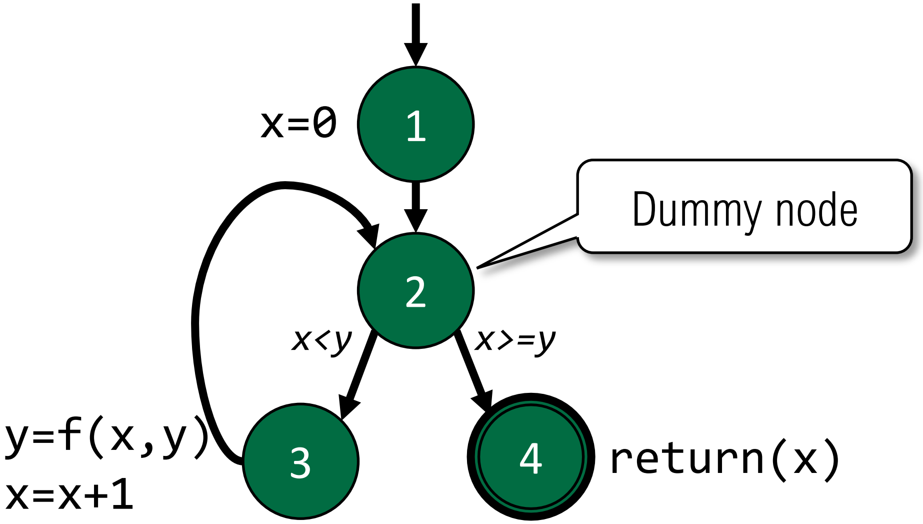
Note that there is no edge from node 2 to node 3

The return statements map to two distinct terminal nodes

CFG Example: `while loop`

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

Loops may require *dummy nodes* to correctly model the control flow

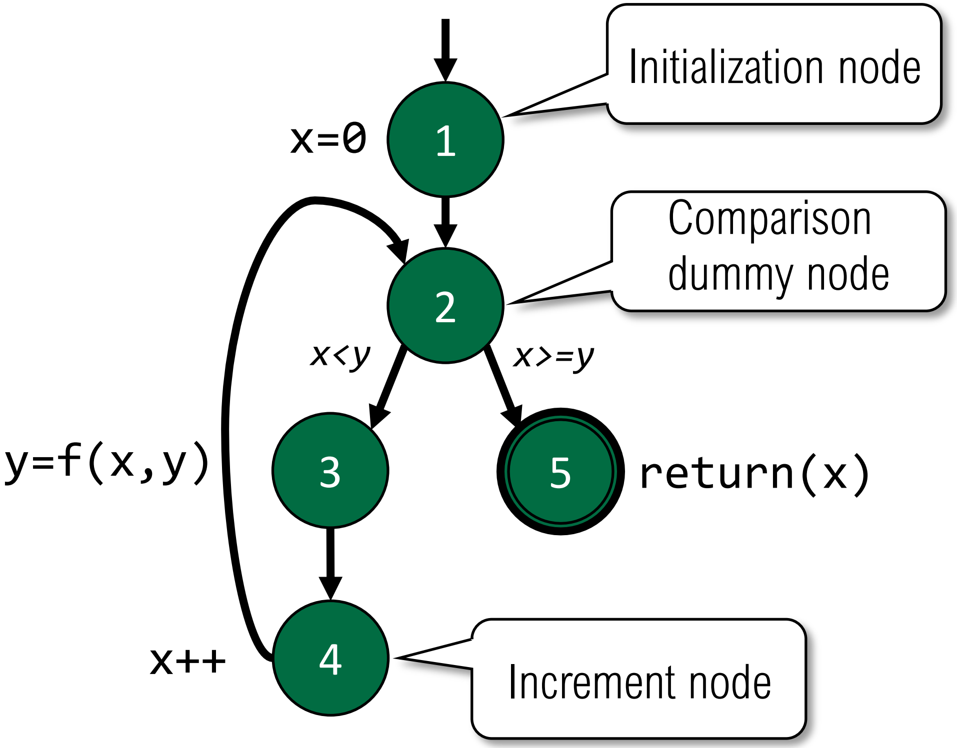


Dummy nodes do not represent statements or basic blocks

Alternate option: annotate node (2) with "while(x<y)" and mark branches "True" and "False"

CFG Example: for loop

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

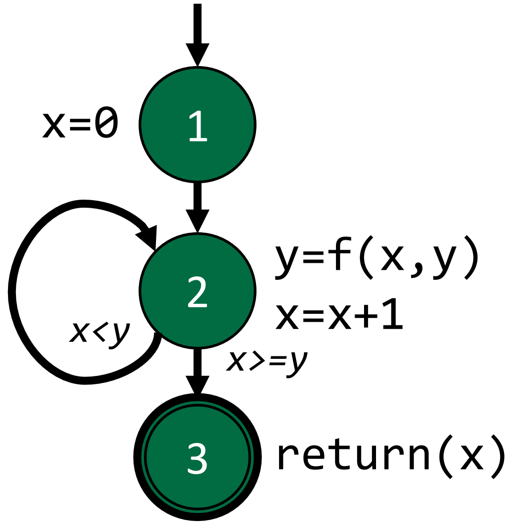


For loops have additional implicit nodes for initialization and incrementing

Increment node (4) could be combined with node (3), but is often left separate to indicate that (4) is part of the loop structure

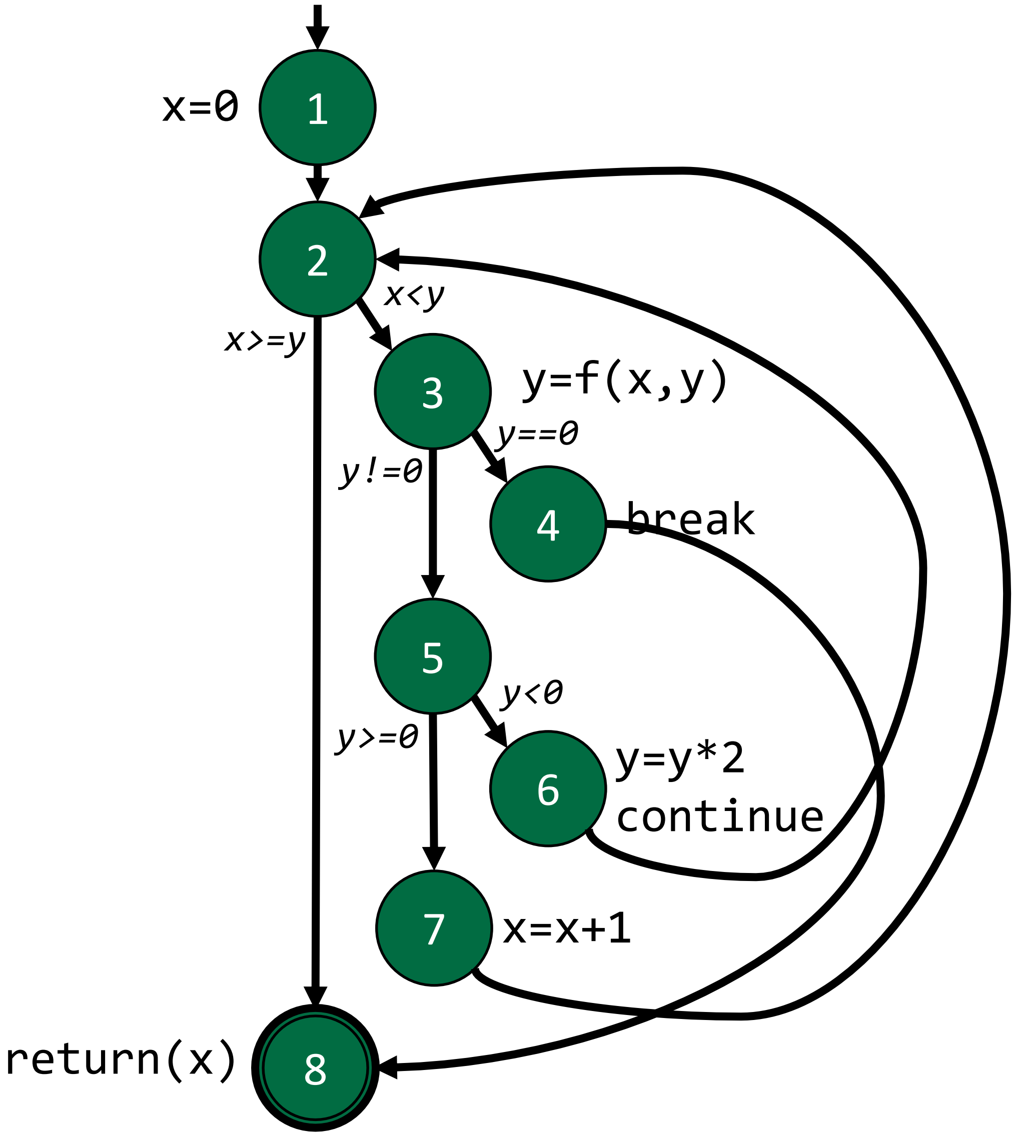
CFG Example: `do loop`

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



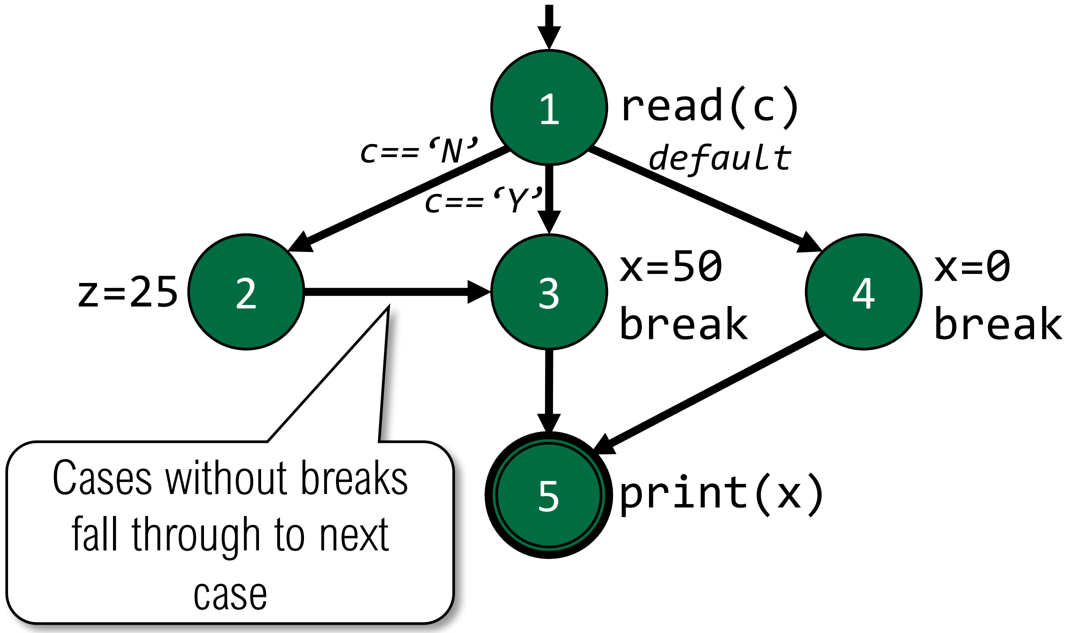
CFG Example: break and continue

```
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 (x);
```



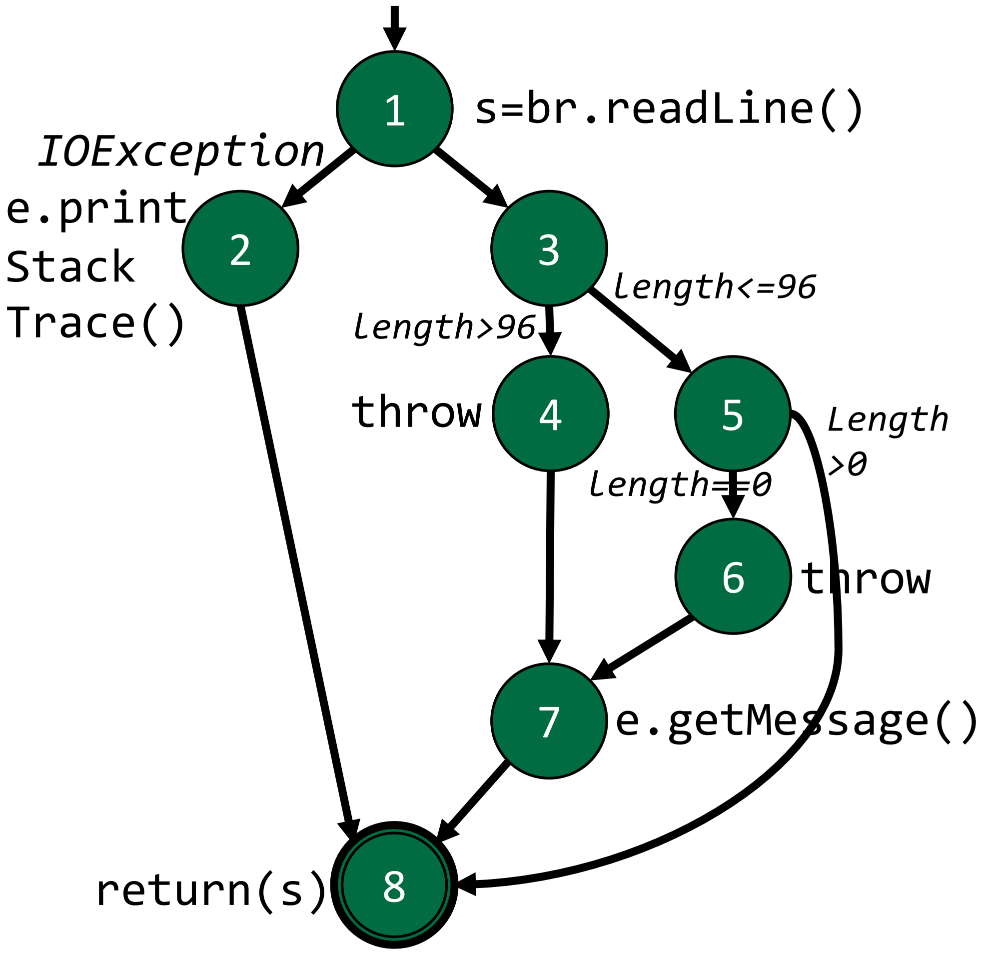
CFG Example: switch/case

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



CFG Example: exceptions

```
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);
```

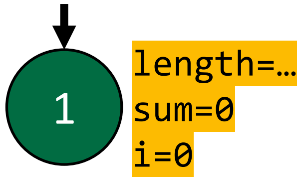


CFG Example: computeStats

```
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

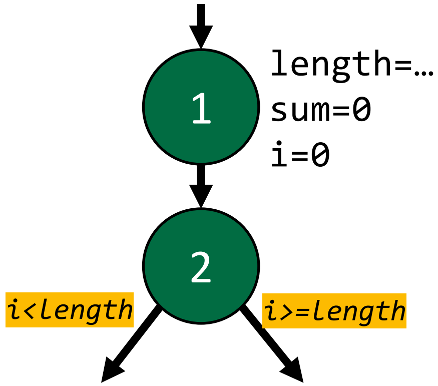
```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  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);

}
```



Here I've combined the initialization node to keep the graph smaller

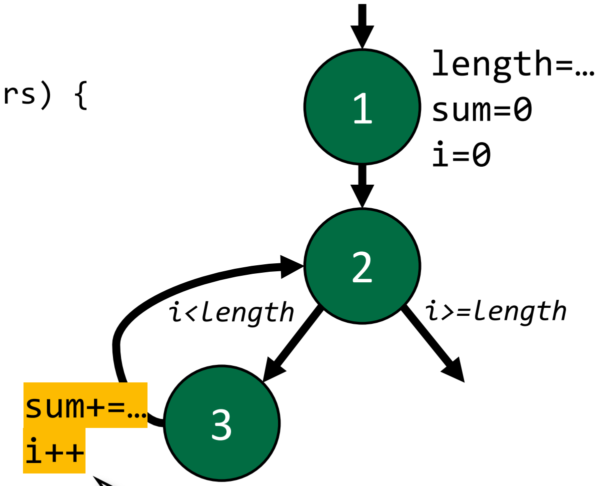
CFG Example: computeStats

```
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

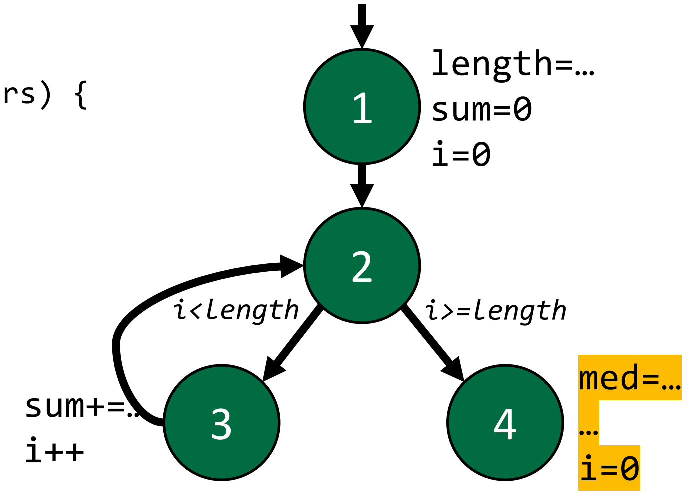
```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  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);

}
```



Here I've combined the increment node to keep the graph smaller

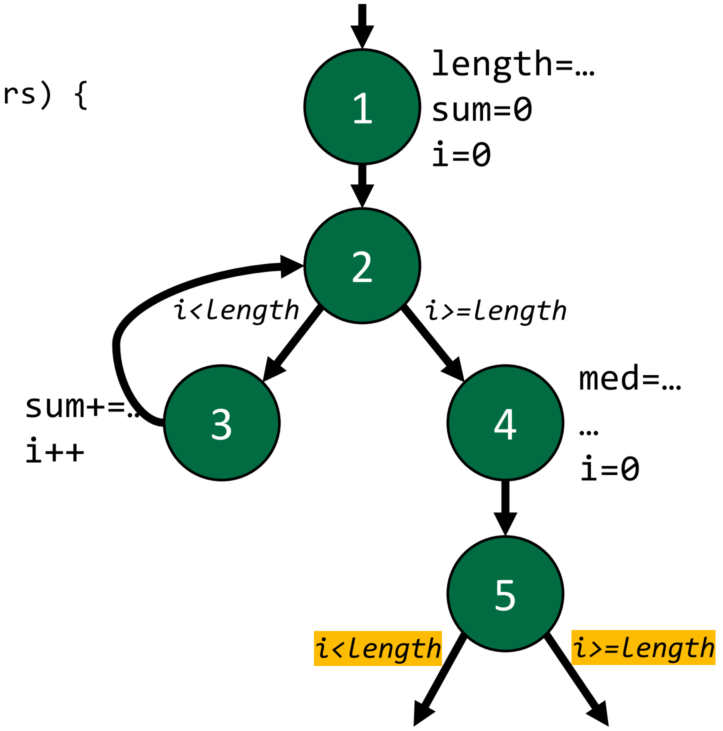
CFG Example: computeStats

```
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);

}
```



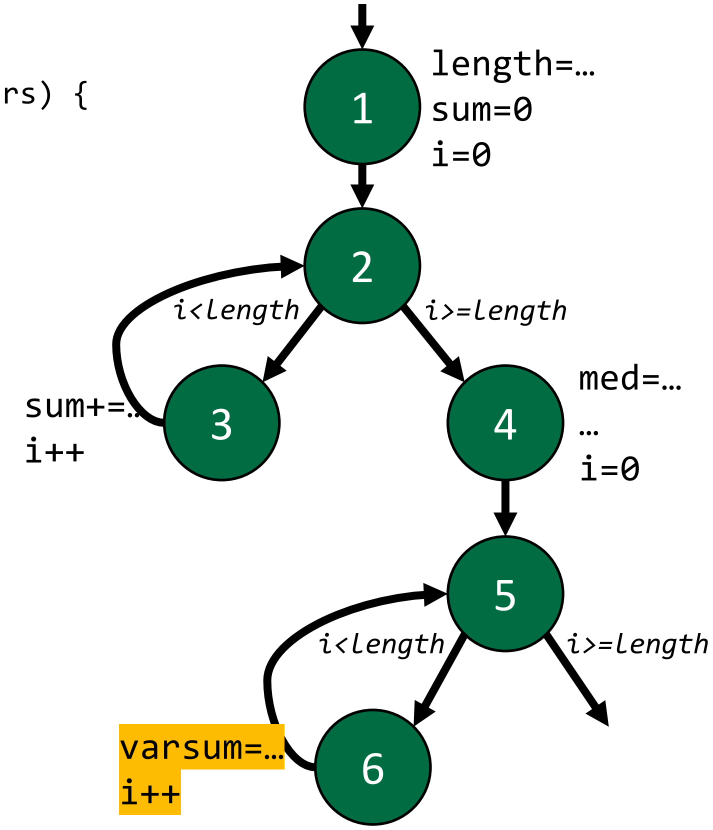
CFG Example: computeStats

```
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);

}
```



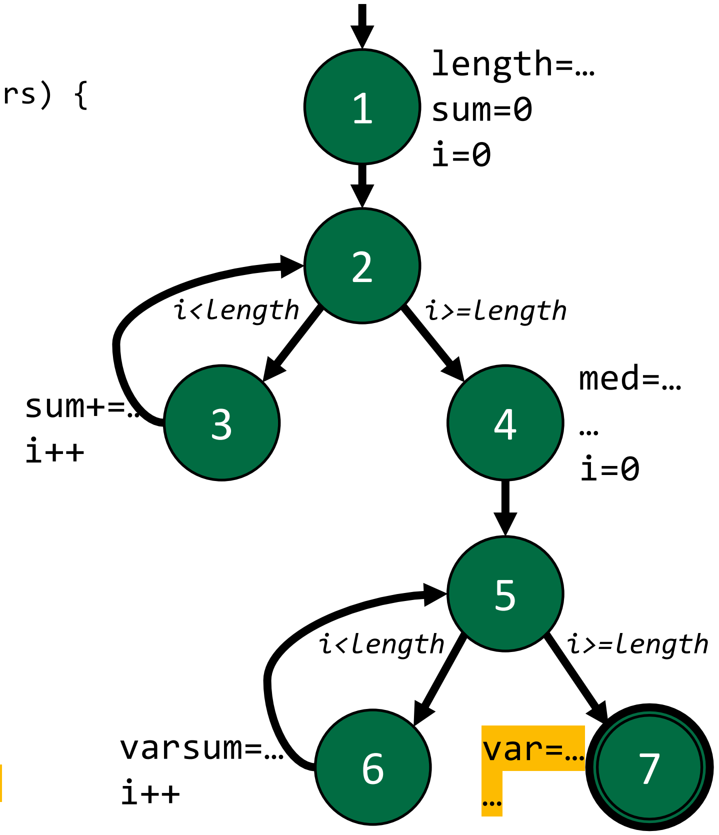
CFG Example: computeStats

```
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);

}
```

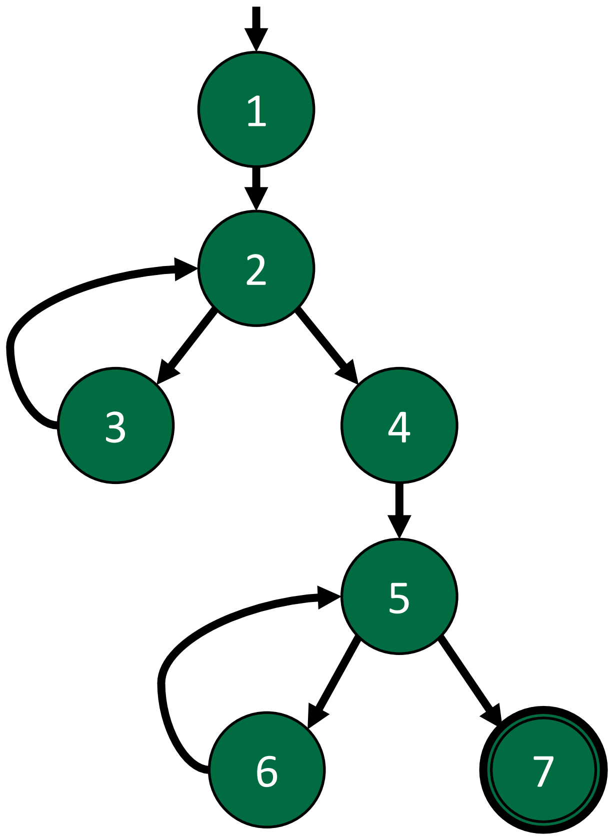


CFG Example: computeStats

```
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: EC

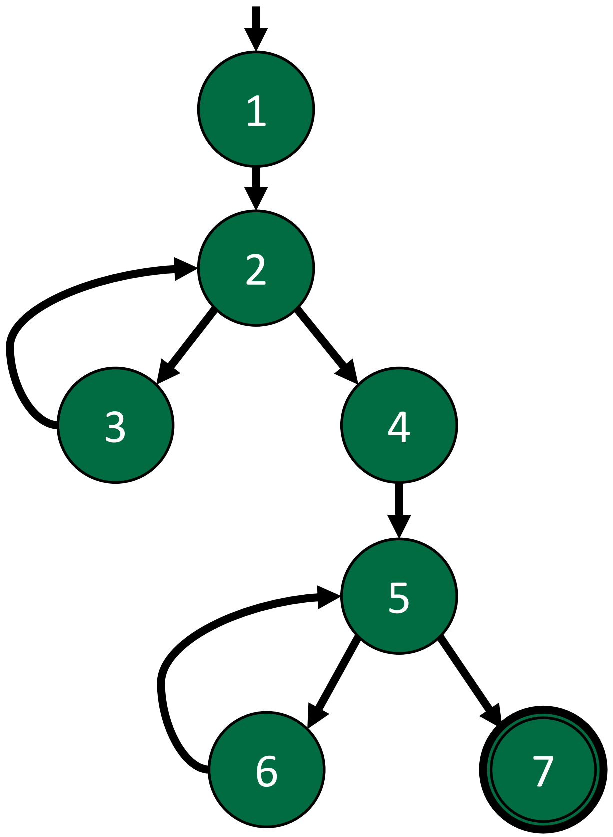


Edge Coverage TRs

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

Test paths

TRs and Test Paths: EC



Edge Coverage TRs

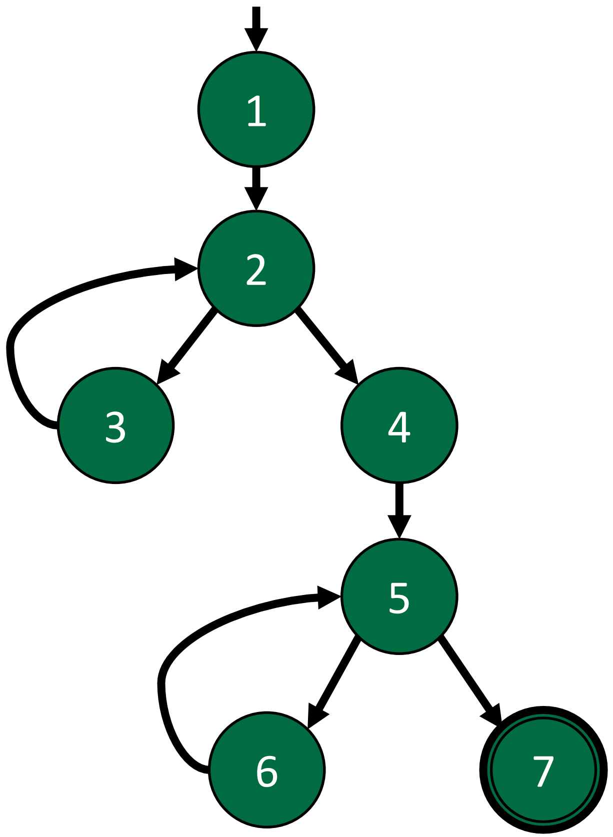
[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

[1,2

Start at the initial node

TRs and Test Paths: EC



Edge Coverage TRs

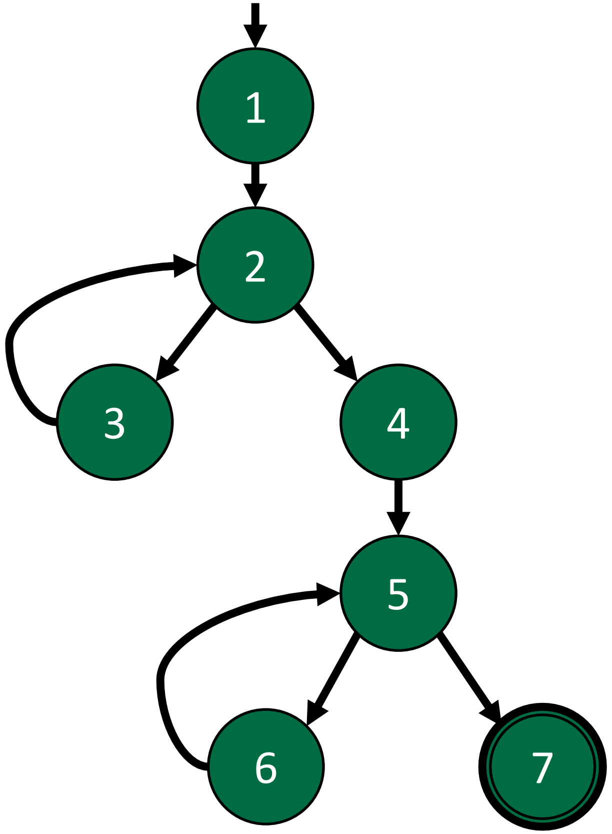
[1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]

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: EC



Edge Coverage TRs

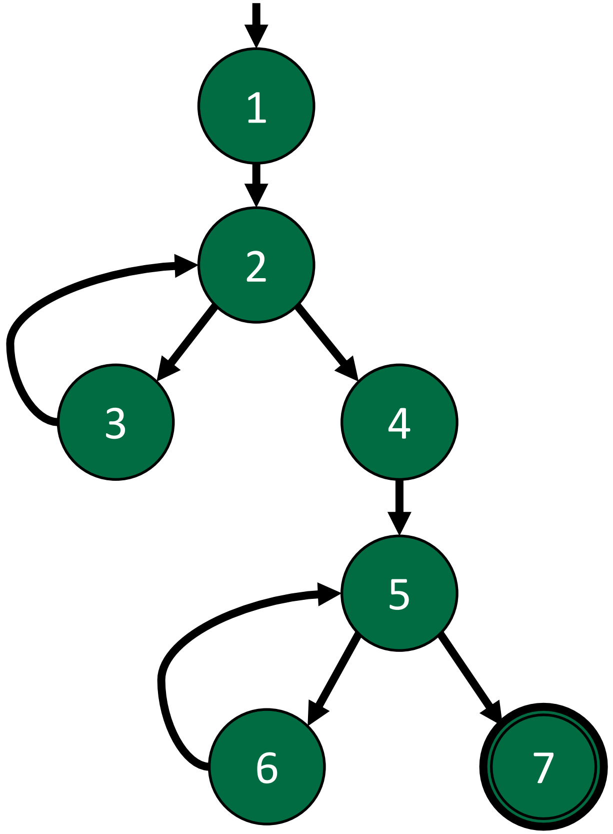
[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

[1,2,3,2

Continue to pick edges that increase coverage

TRs and Test Paths: EC



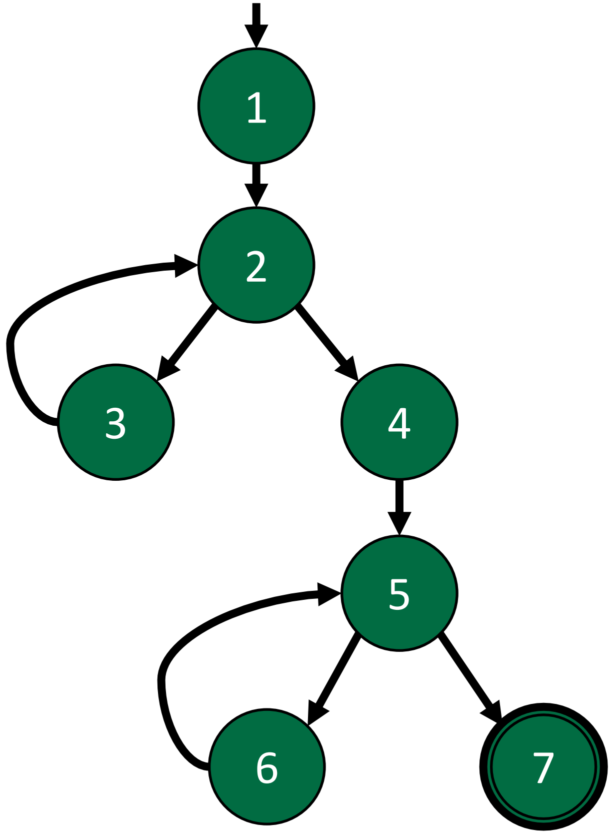
Edge Coverage TRs

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

Test paths

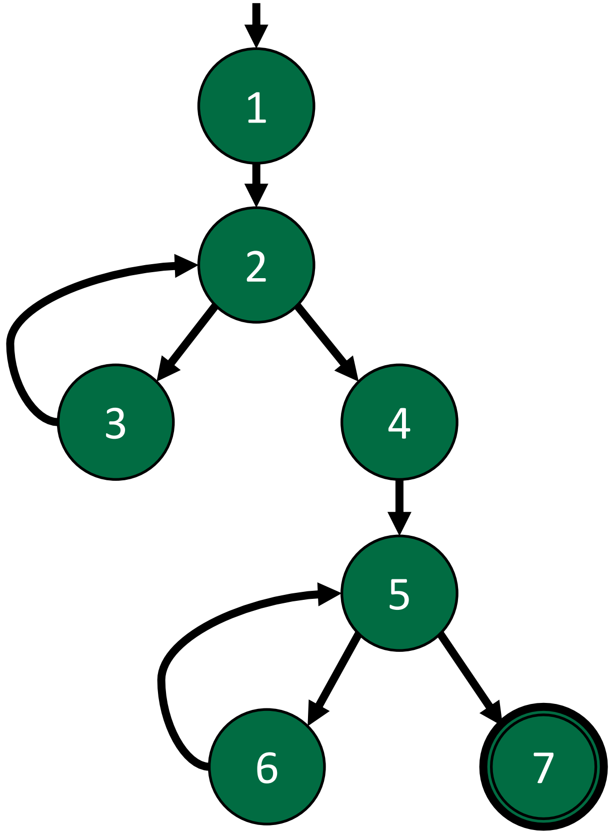
[1,2,3,2,4

TRs and Test Paths: EC



- 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: EC



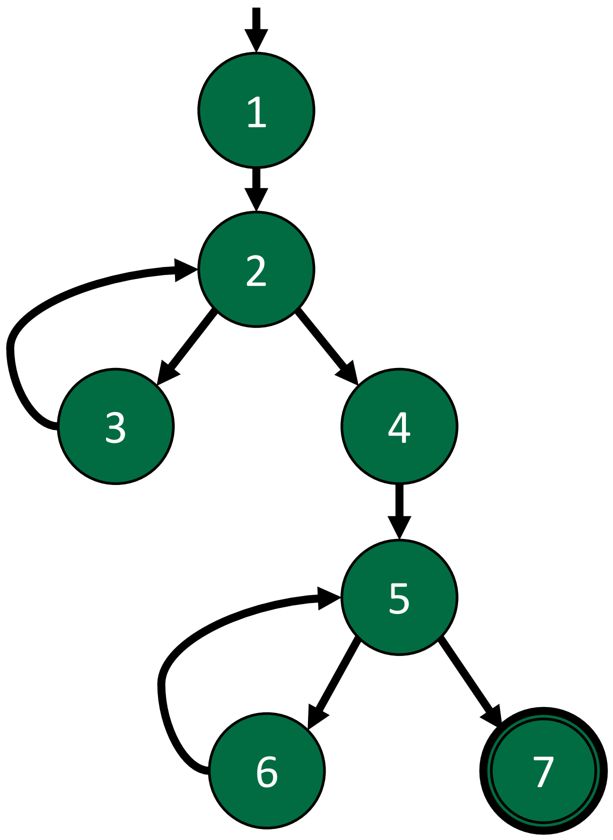
Edge Coverage TRs

[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: EC



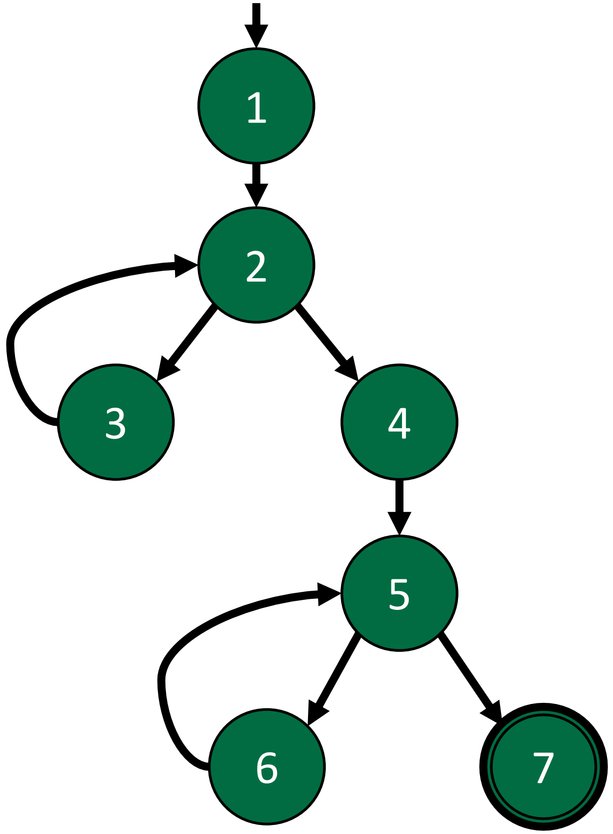
Edge Coverage TRs

[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

TRs and Test Paths: EC



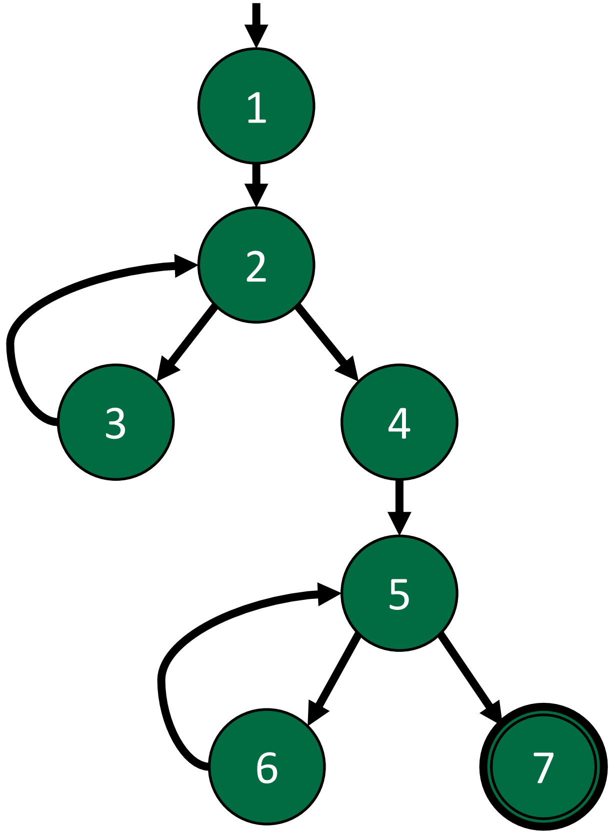
Edge Coverage TRs

[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: EC



Edge Coverage TRs

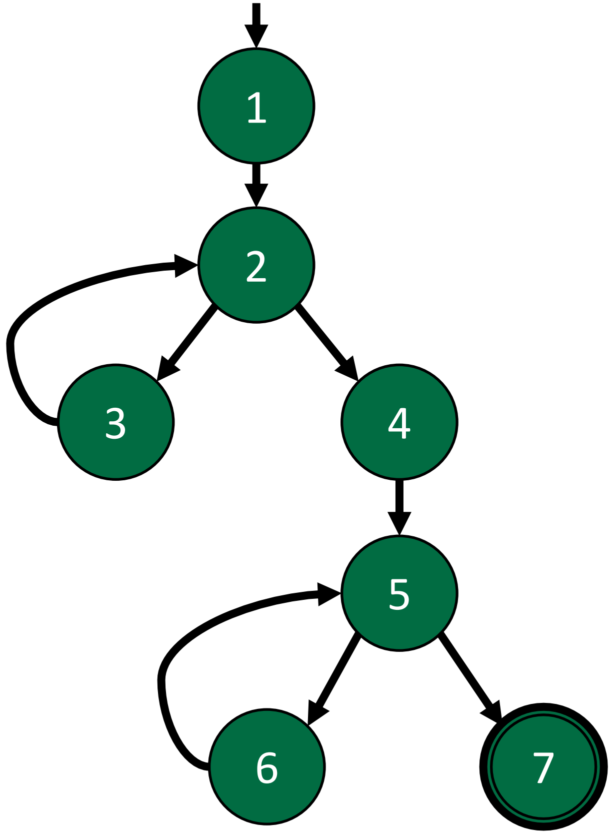
[1,2], [2,3], [2,4], [3,2],
[4,5], [5,6], [5,7], [6,5]

Test paths

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

Edge coverage is satisfied with 1 test path

TRs and Test Paths: EPC

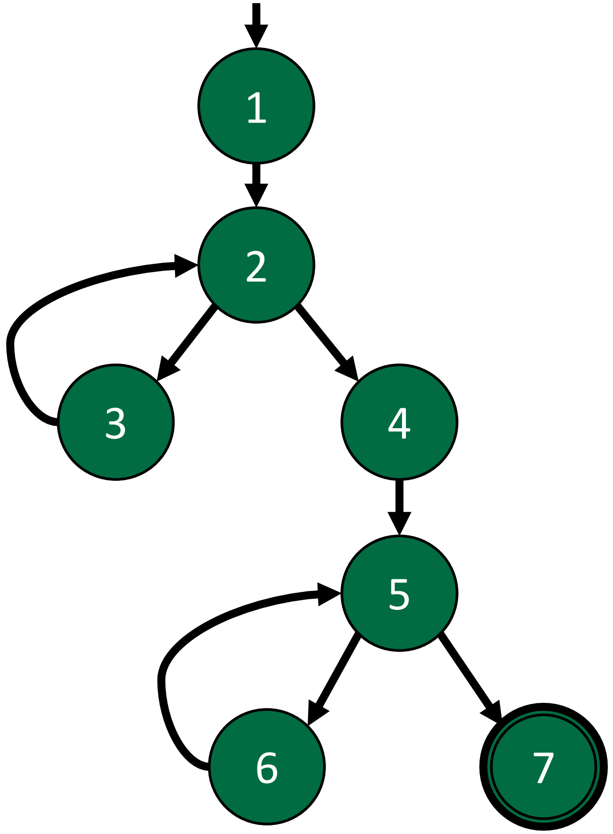


Edge-Pair TRs

[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: EPC



Edge-Pair TRs

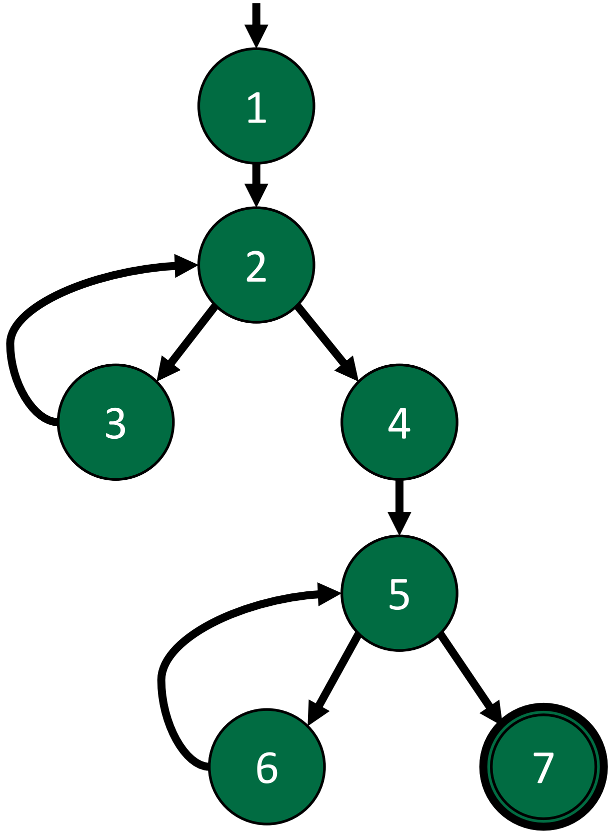
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3

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

TRs and Test Paths: EPC



Edge-Pair TRs

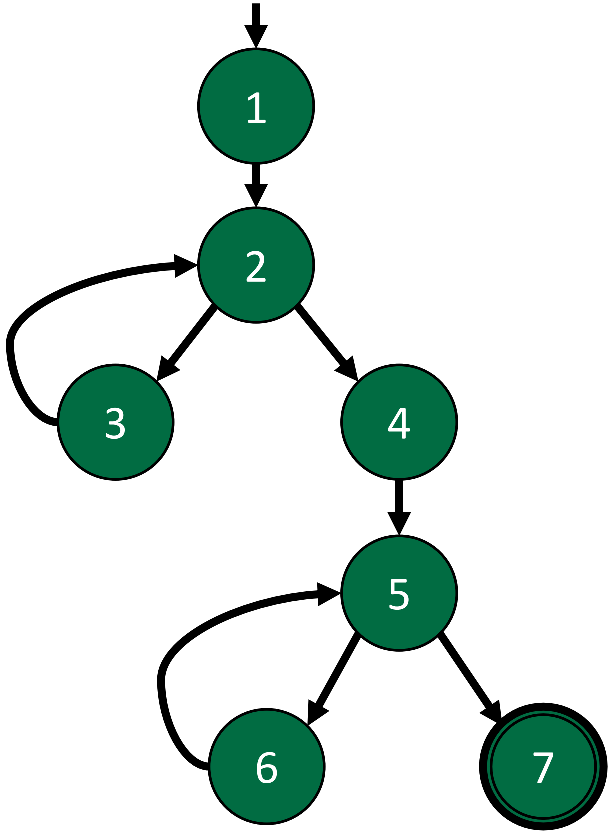
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

[1,2,3,2

Select an edge that increases edge-pair coverage

TRs and Test Paths: EPC



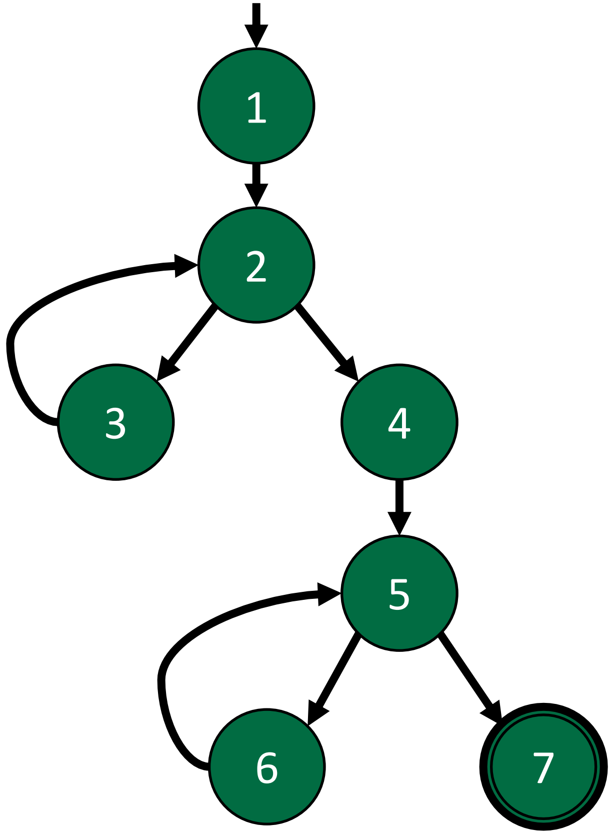
Edge-Pair TRs

[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: EPC



Edge-Pair TRs

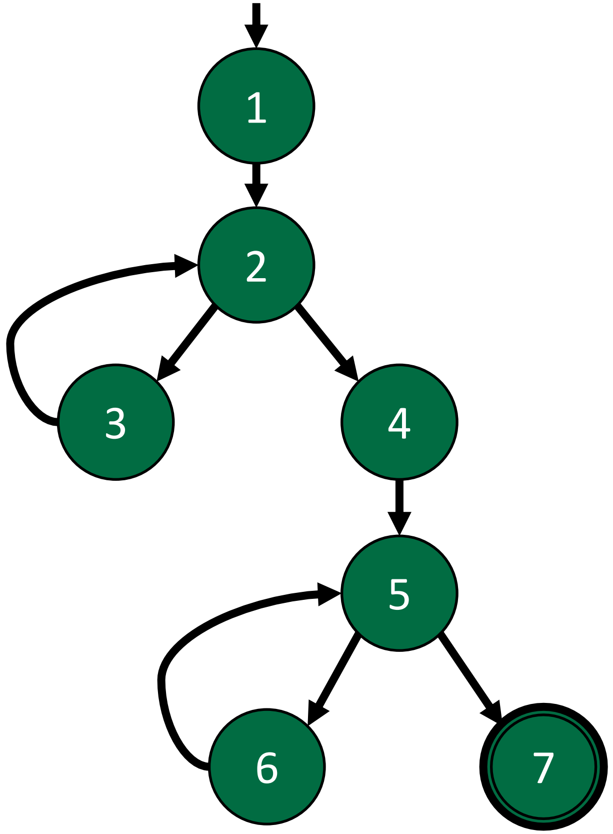
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

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

TRs and Test Paths: EPC



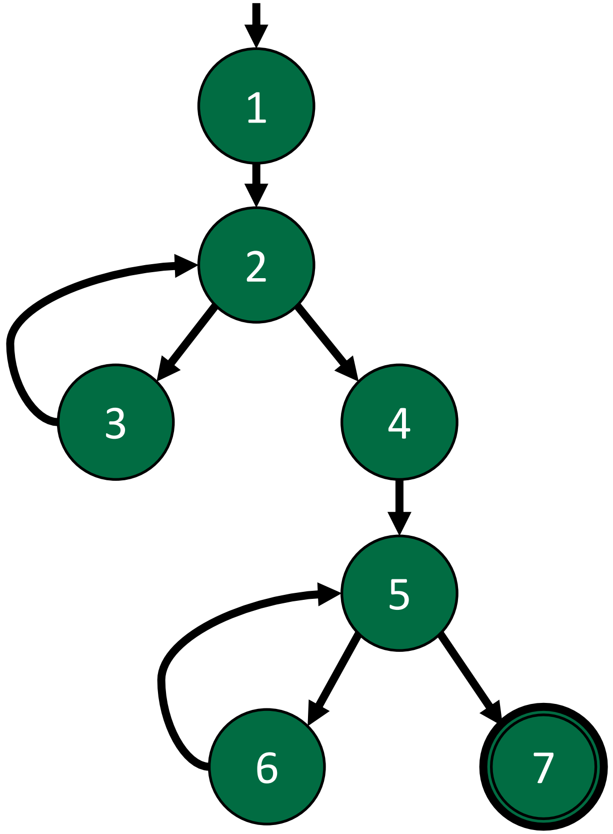
Edge-Pair TRs

[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: EPC



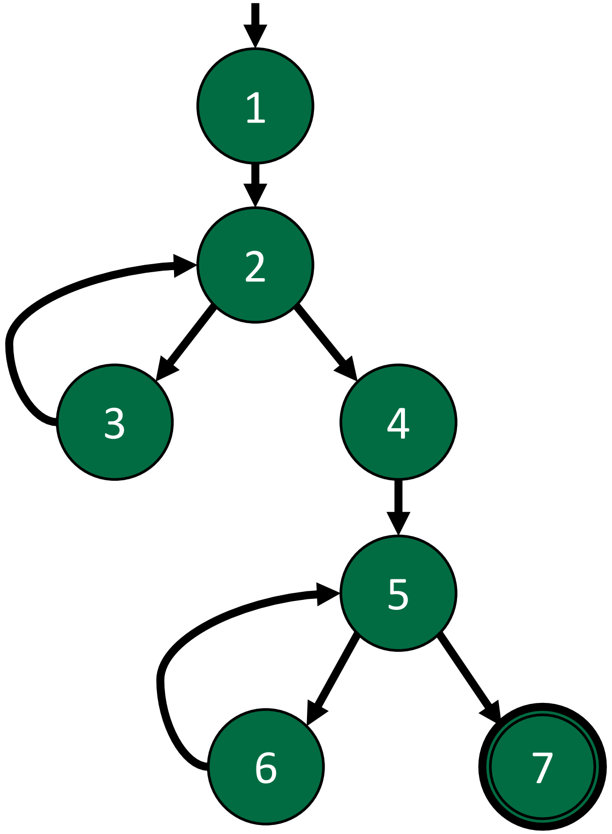
Edge-Pair TRs

[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: EPC



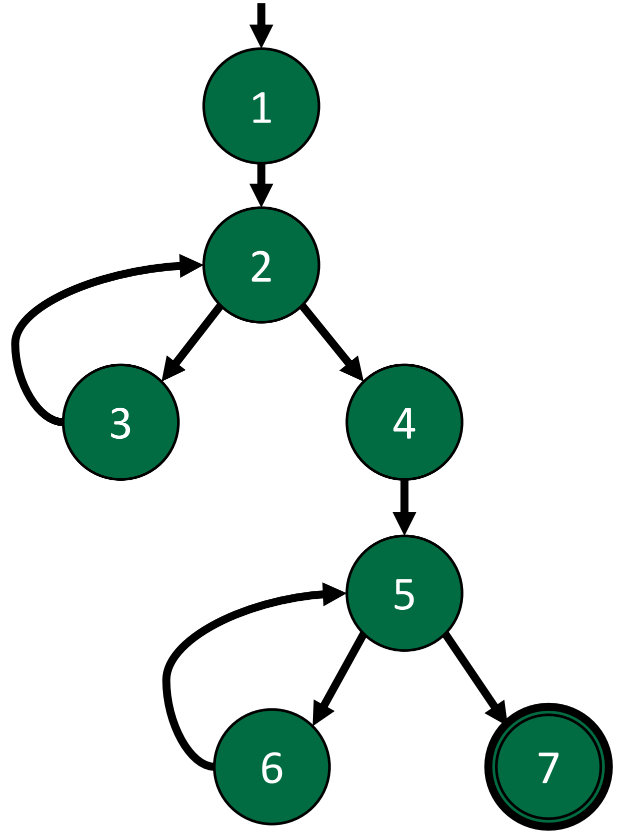
Edge-Pair TRs

[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: EPC



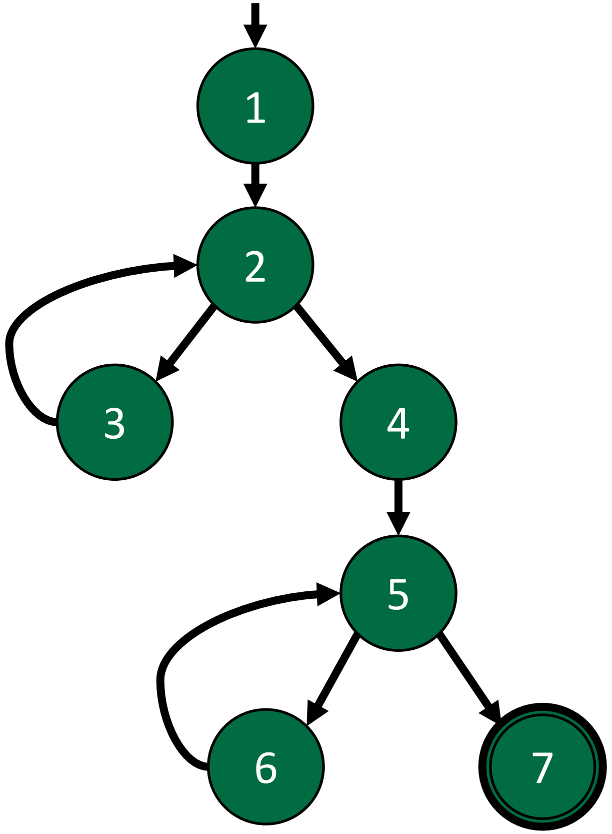
Edge-Pair TRs

[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: EPC



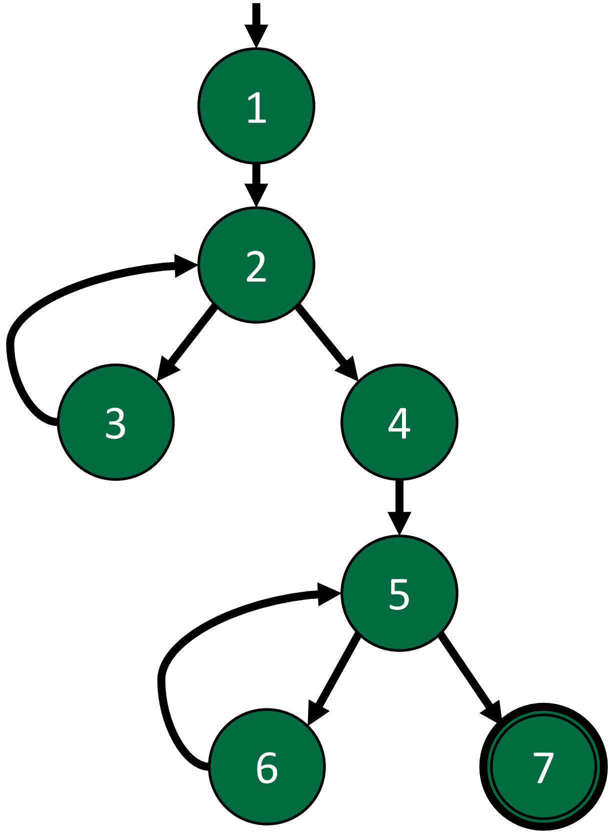
Edge-Pair TRs

[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: EPC



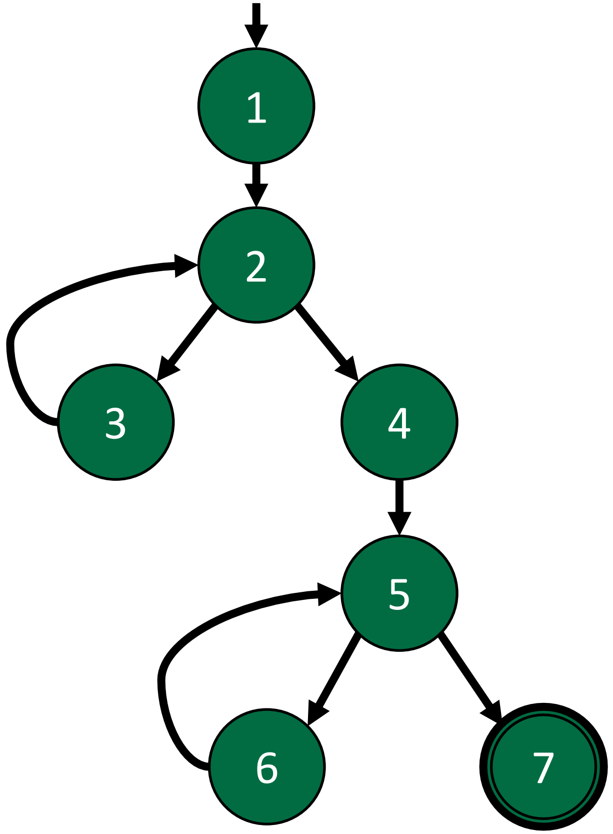
Edge-Pair TRs

[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: EPC



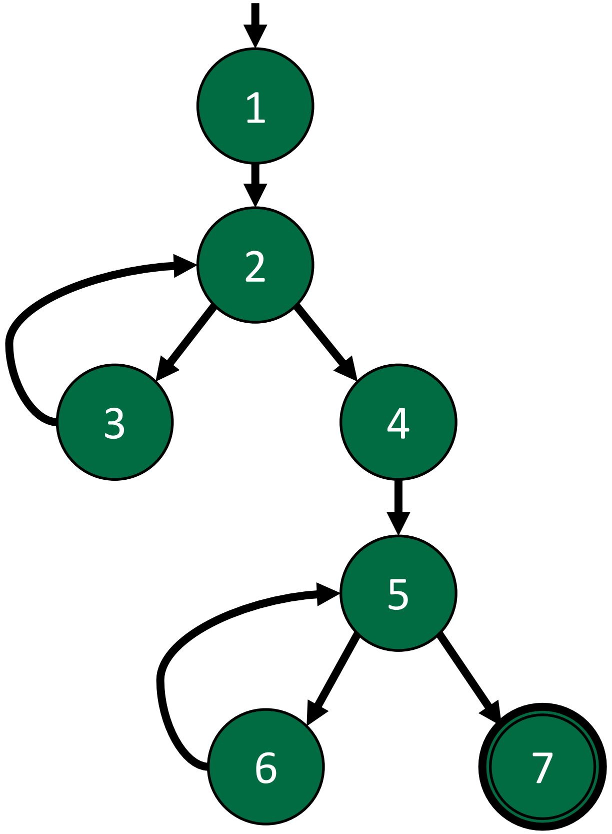
Edge-Pair TRs

[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: EPC



Edge-Pair TRs

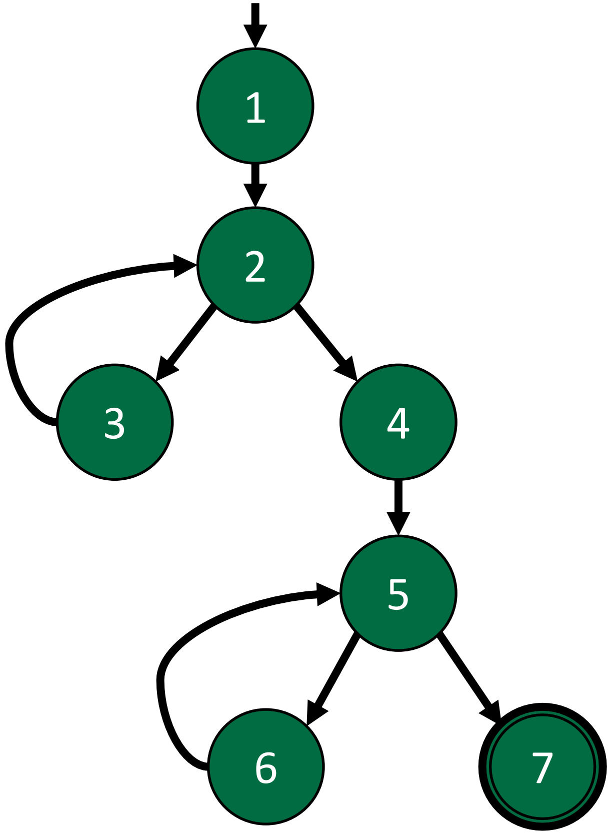
[1,2,3], [1,2,4], [2,3,2], [2,4,5],
[3,2,3], [3,2,4],
[4,5,6], [4,5,7], [5,6,5], [6,5,6],
[6,5,7]

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: EPC



Edge-Pair TRs

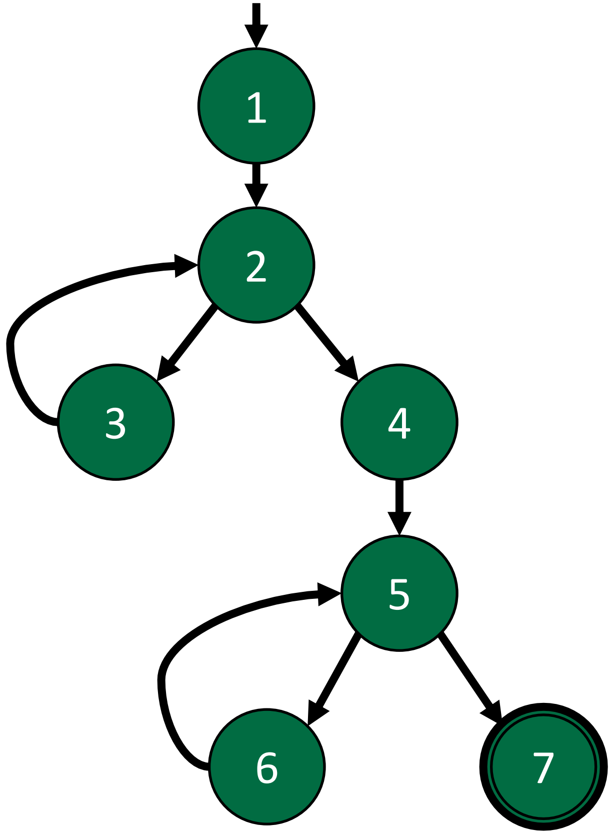
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

[1,2,4

TRs and Test Paths: EPC



Edge-Pair TRs

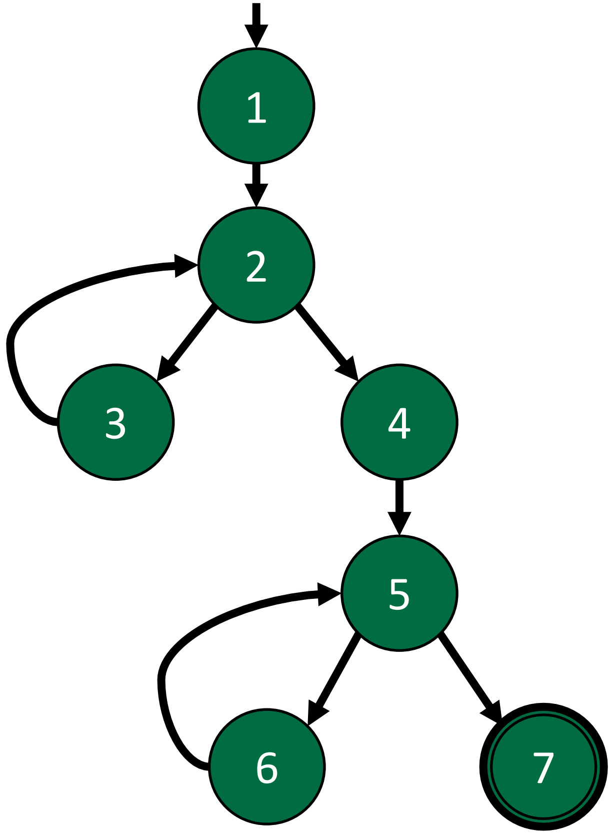
[1,2,3], [1,2,4], [2,3,2], [2,4,5],
[3,2,3], [3,2,4],
[4,5,6], [4,5,7], [5,6,5], [6,5,6],
[6,5,7]

Test paths

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

[1,2,4,5

TRs and Test Paths: EPC



Edge-Pair TRs

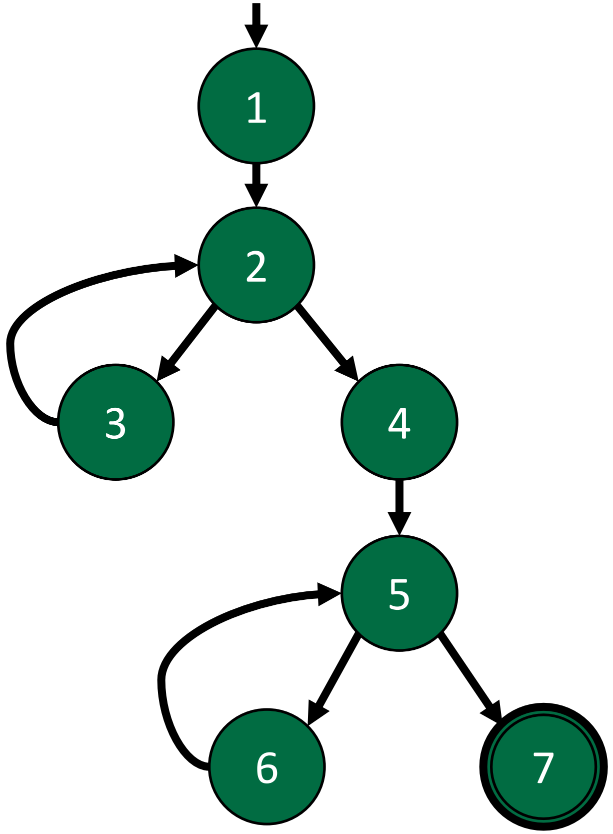
[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]

Test paths

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

[1,2,4,5,7]

TRs and Test Paths: EPC



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]

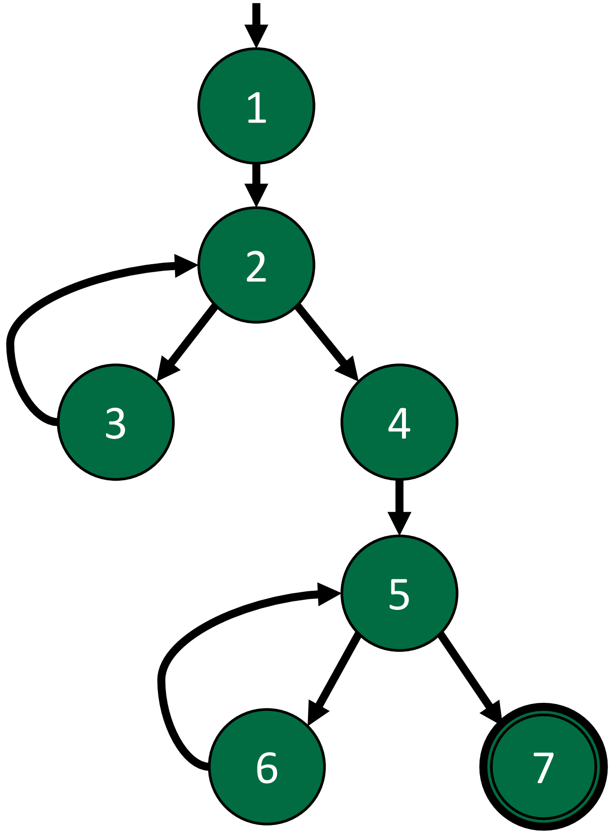
Test paths

[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: PPC

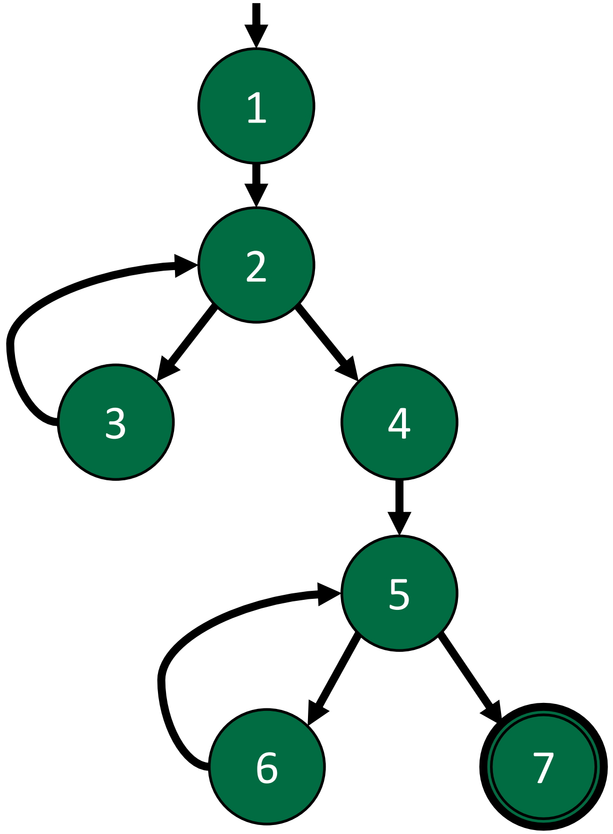


Prime Path TRs

- [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: PPC



Prime Path TRs

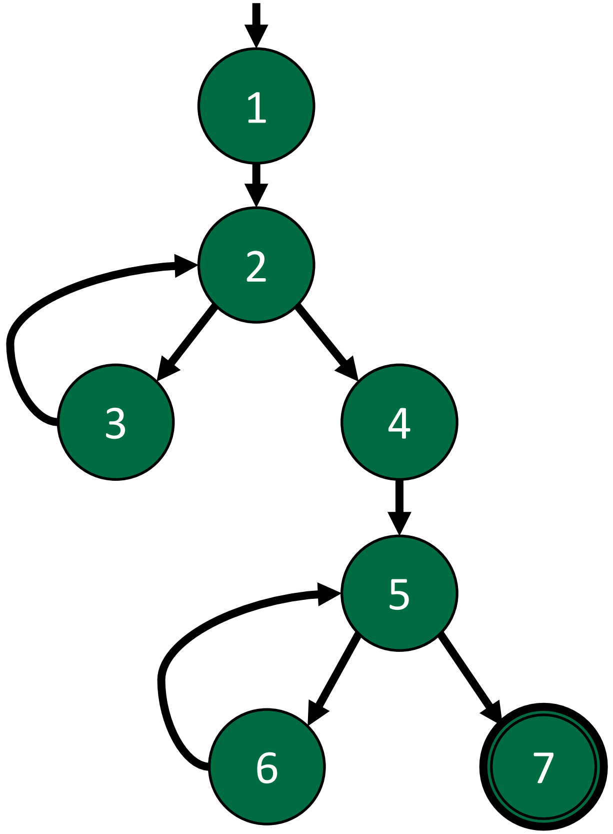
[1,2,3], [1,2,4,5,6], [1,2,4,5,7], [2,3,2], [3,2,3], [3,2,4,5,6], [3,2,4,5,7], [5,6,5], [6,5,6], [6,5,7]

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: PPC



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]

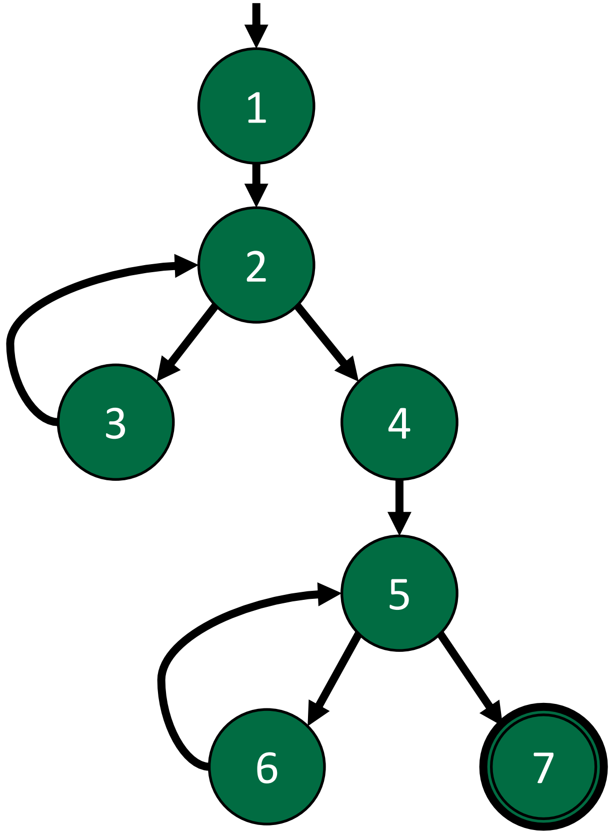
Test paths

[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: PPC



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]

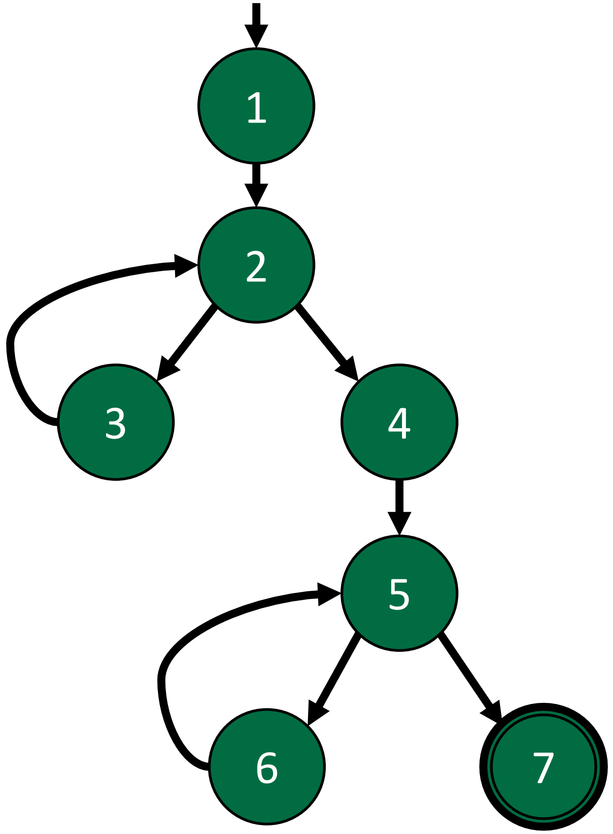
Test paths

[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: PPC



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

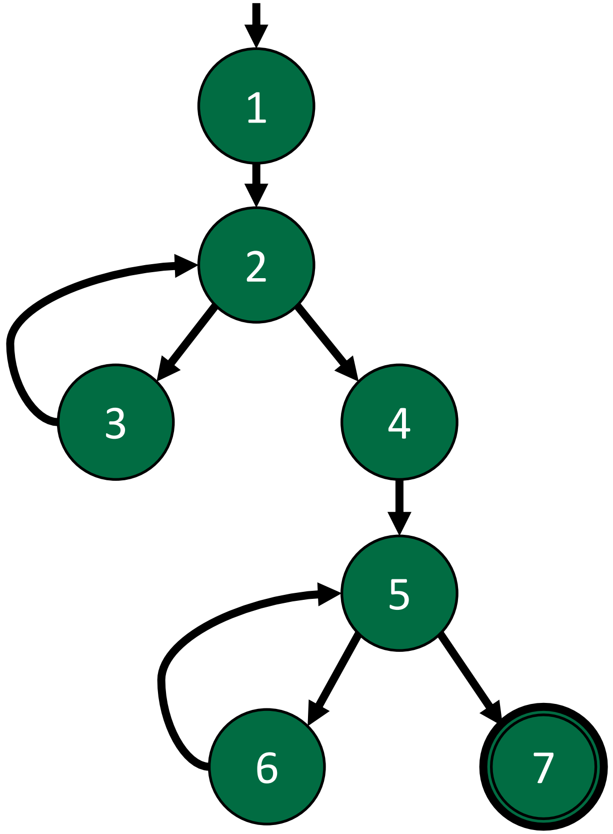
[1,2,3,2,3,2,4,5,6,5,6,5,7]

[1,2,4,5,7]

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

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

TRs and Test Paths: PPC



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]