Intro to Software Testing

Chapter 7.3

Graph Coverage from Source Code

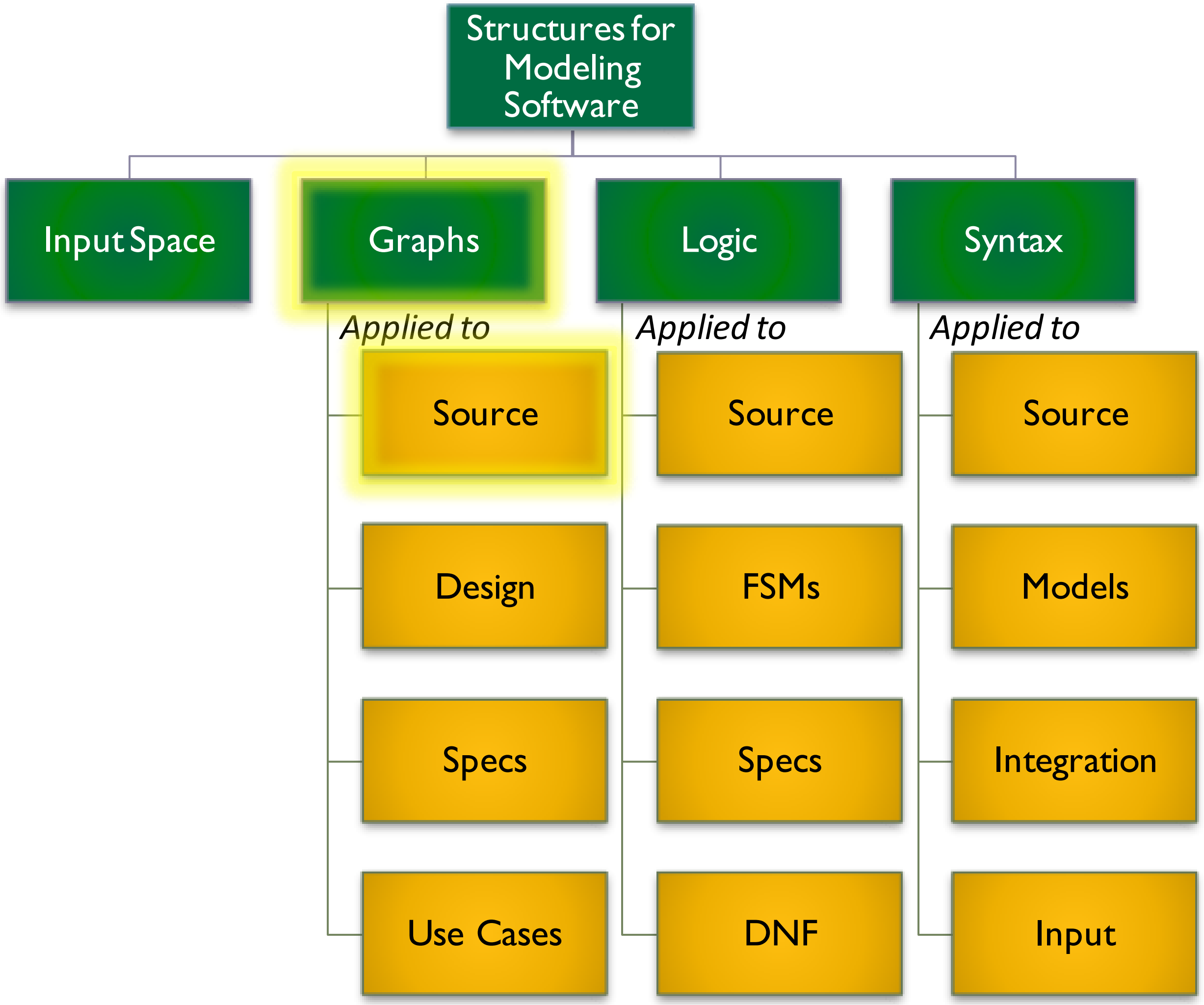
Dr. Brittany Johnson-Matthews

(Dr. B for short)

https://go.gmu.edu/SWE637

Provided by Bob Kurtz

Graph Coverage



Structures for Modeling Software

Input Space

Graphs

Logic

Syntax

Applied to

Source

Design

Specs

Use Cases

Applied to

Source

FSMs

Specs

DNF

Applied to

Source

Models

Integration

Input

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

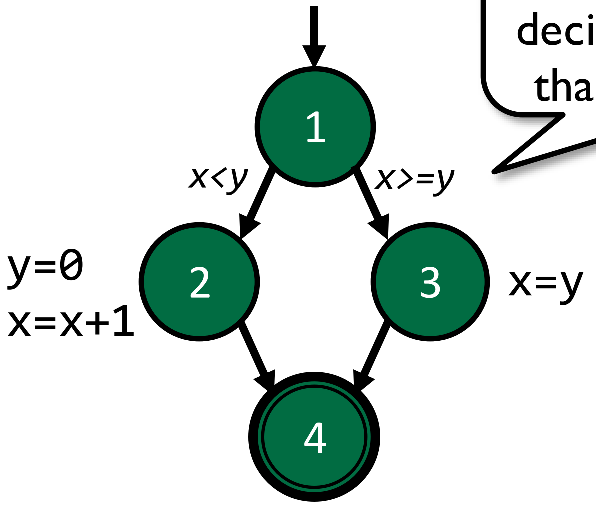
Control Flow Graphs

- 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

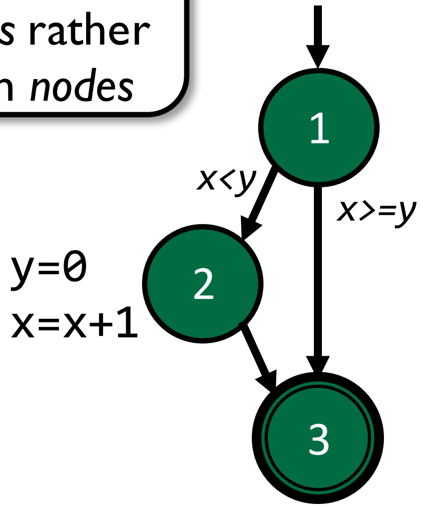
CFG Example: If

```
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*

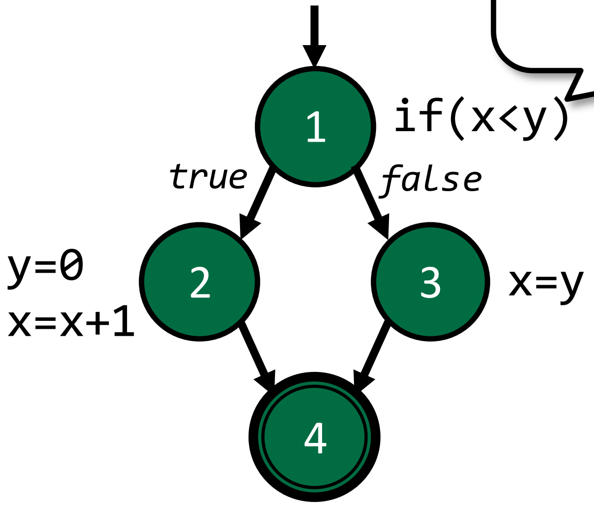


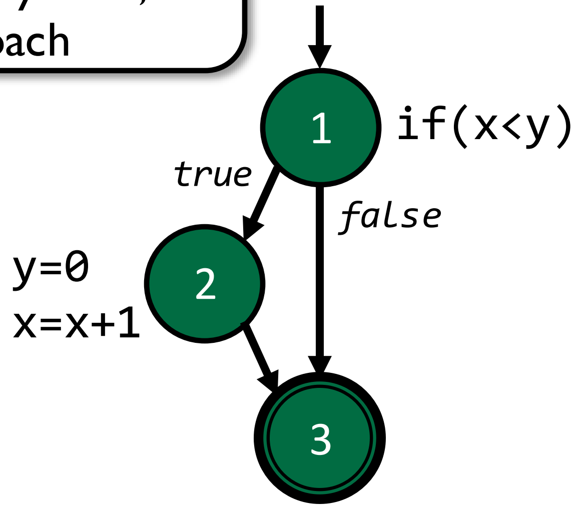
CFG Example: If

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

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

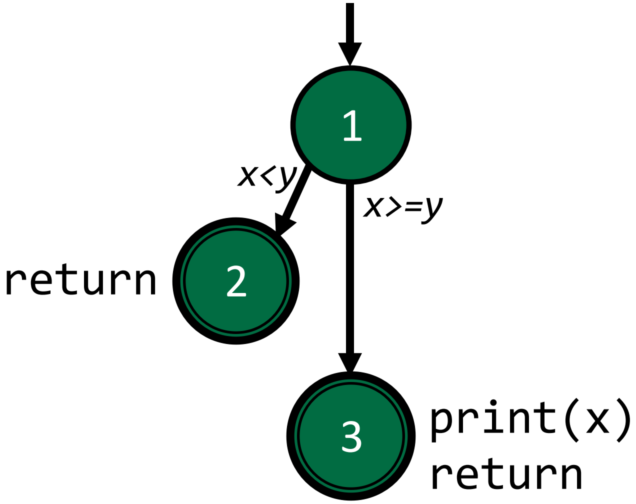
Annotating decision nodes is an alternative, and equally valid, approach





CFG Example: If-Return

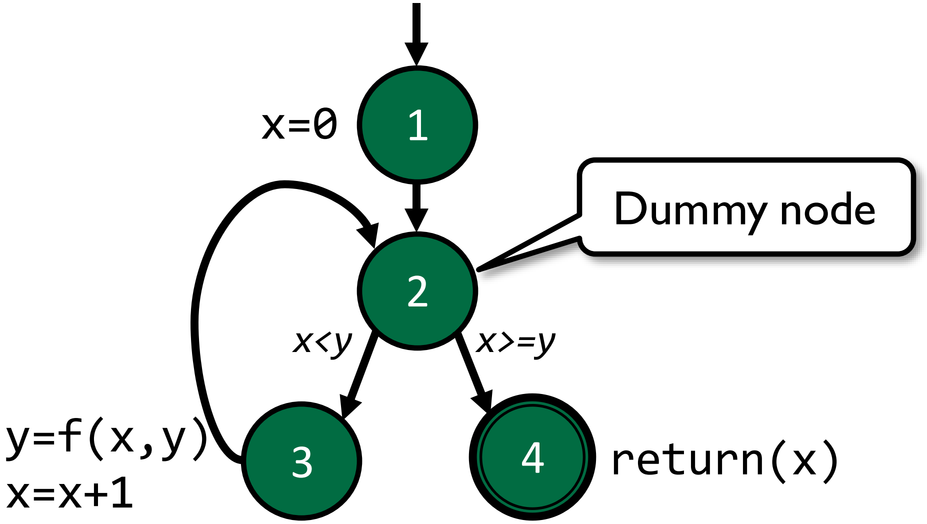
```
if (x < y) {
  return;
}
print (x);
return;
```



- 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

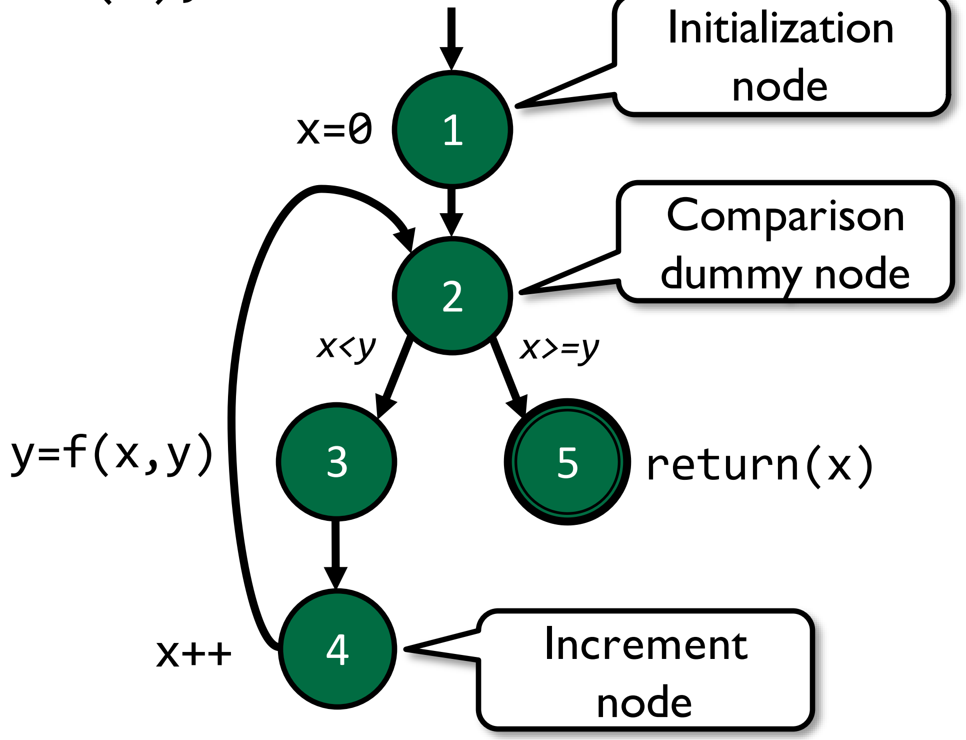
```
x = 0;
while (x < y) {
  y = f (x, y);
  x = x + 1;
}
return (x);
```



- 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);
```



Initialization node

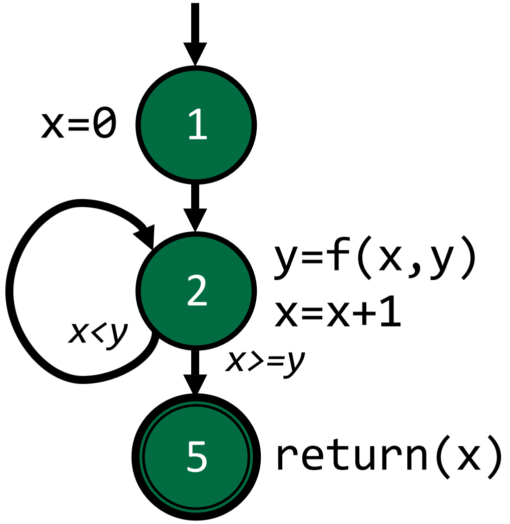
Comparison dummy node

Increment node

- 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

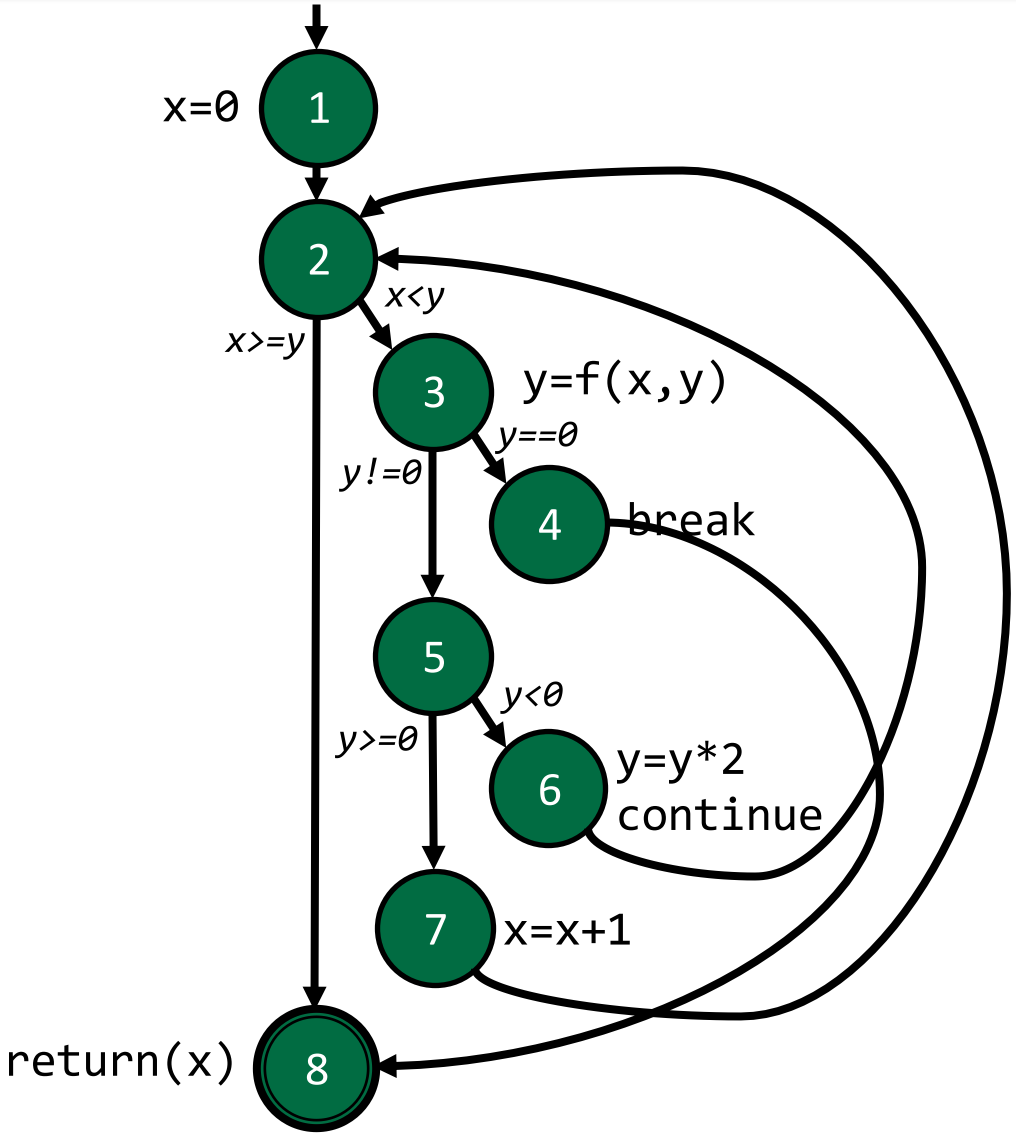
CFG Example: Do Loop

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



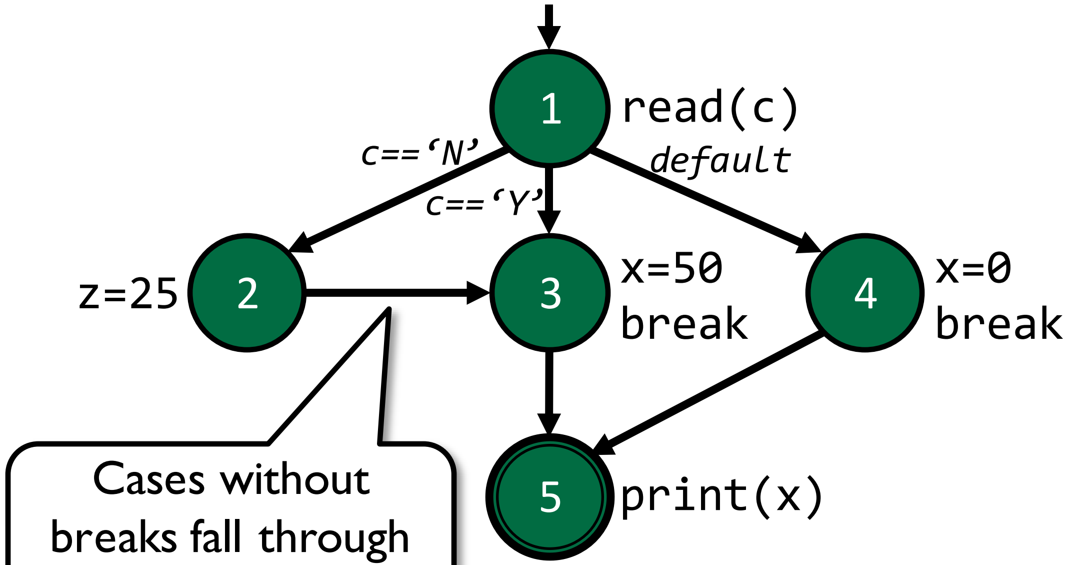
CFG Example: Break and Continue

```
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

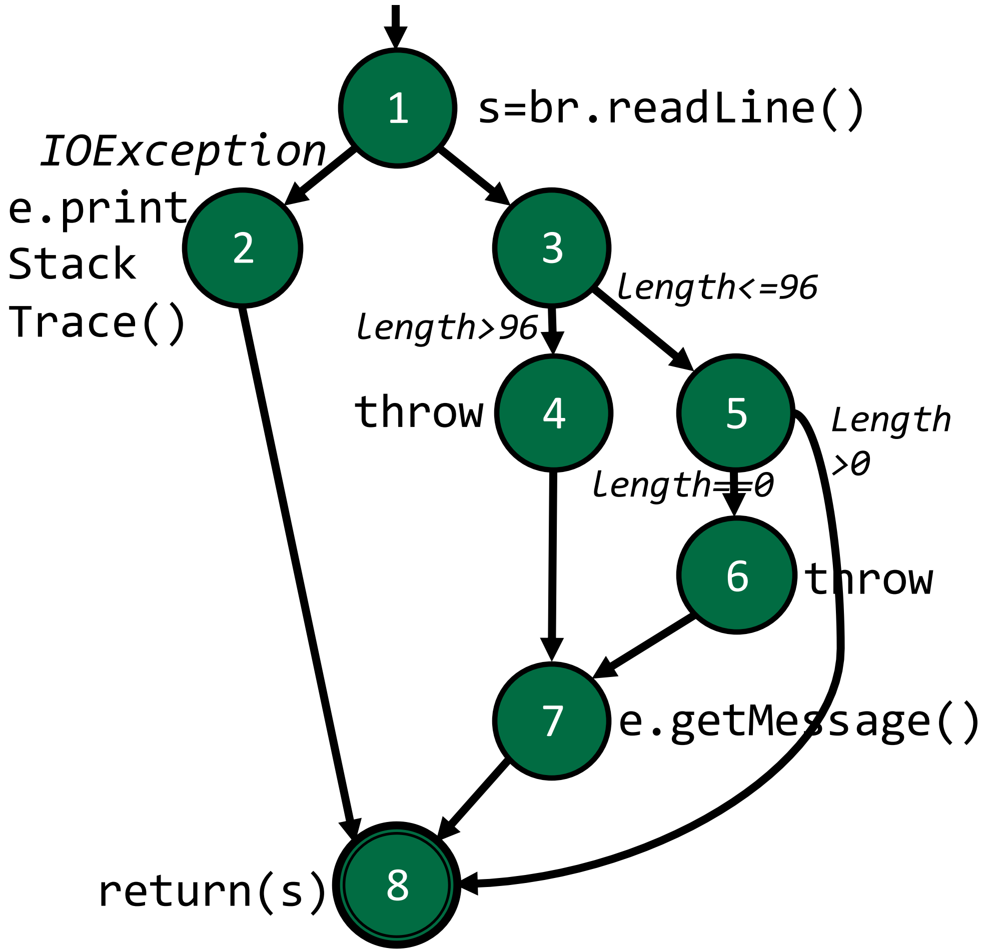
```
read (c);
switch (c) {
  case 'N':
    z = 25;
  case 'Y':
    x = 50;
    break;
  default:
    x = 0;
    break;
}
print (x);
```



Cases without breaks fall through to next case

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);
```

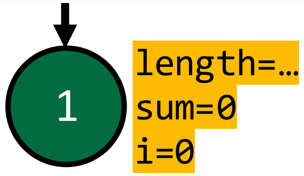


CFG Example: computeStats

```
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

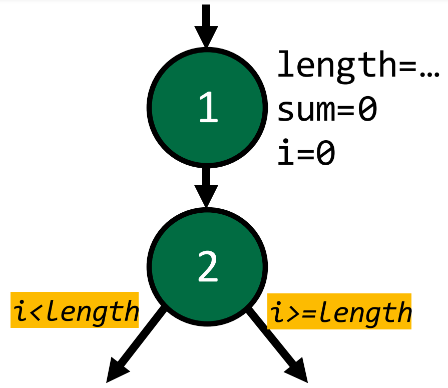
```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  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

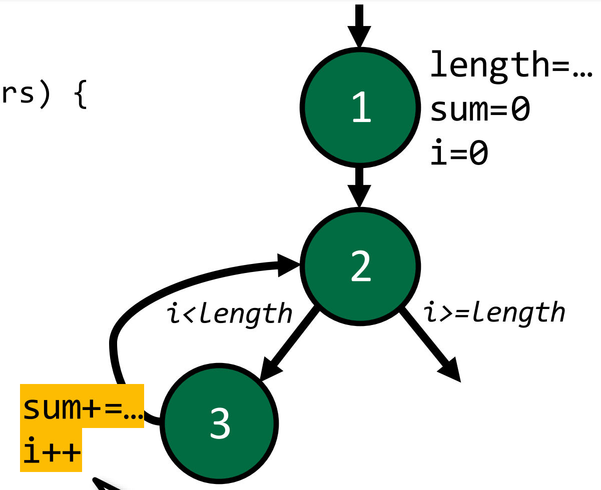
CFG Example: computeStats

```
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

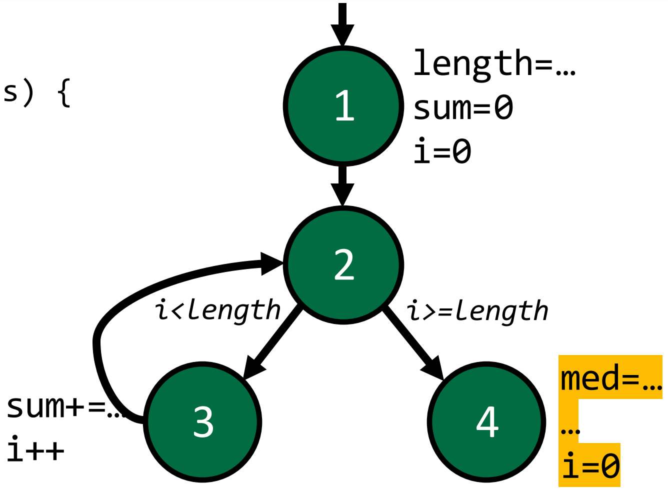
```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  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

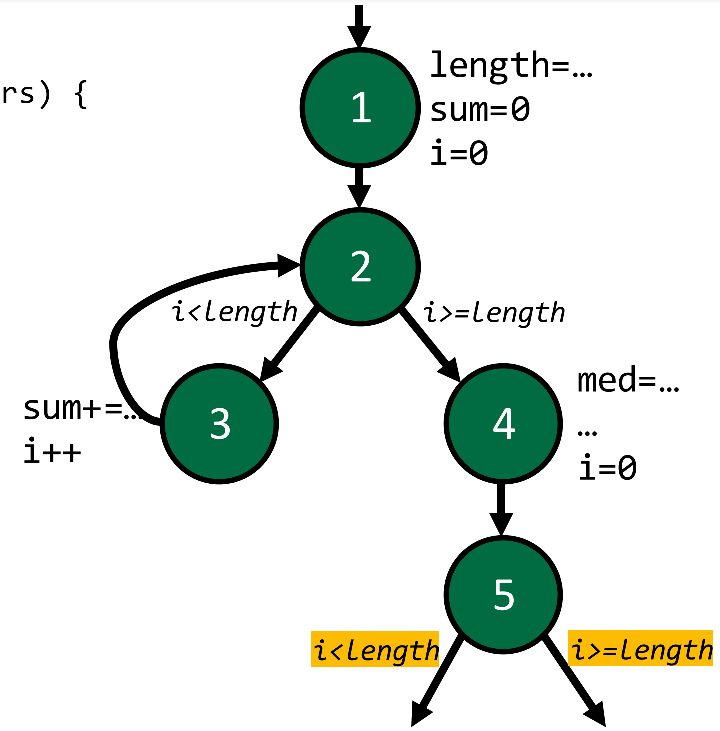
CFG Example: computeStats

```
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);
}
```



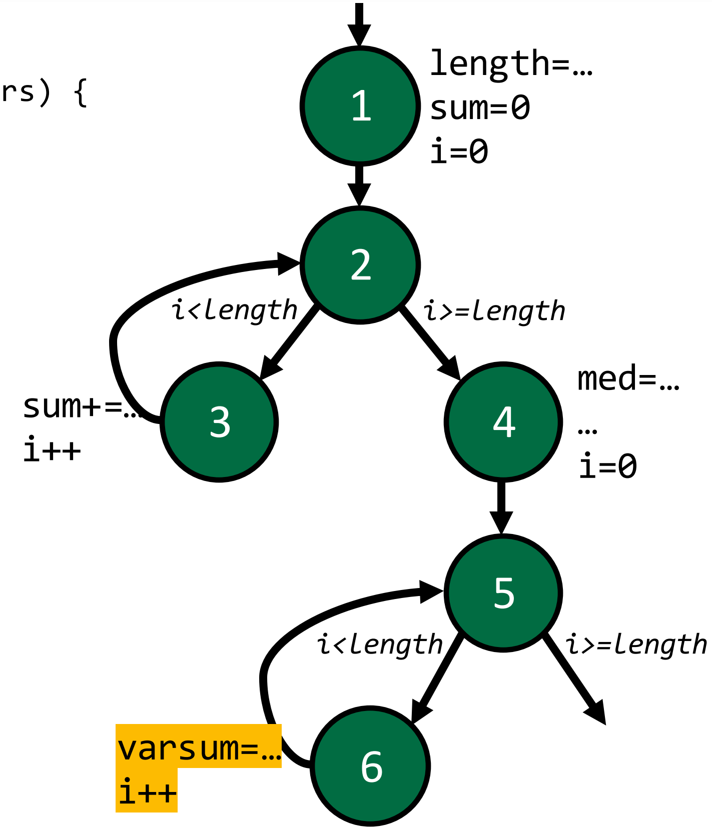
CFG Example: computeStats

```
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);

}
```



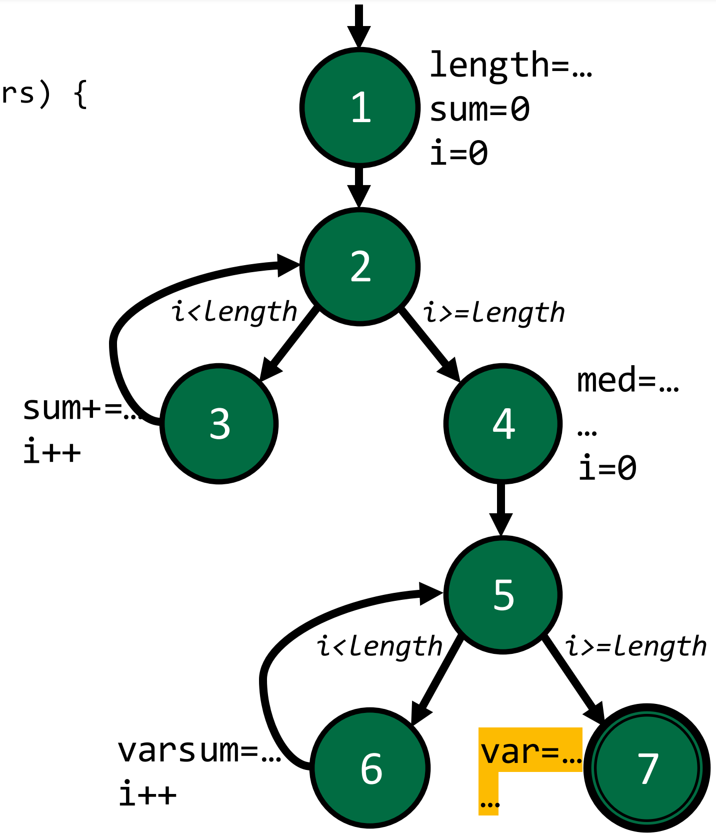
CFG Example: computeStats

```
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);
}
```

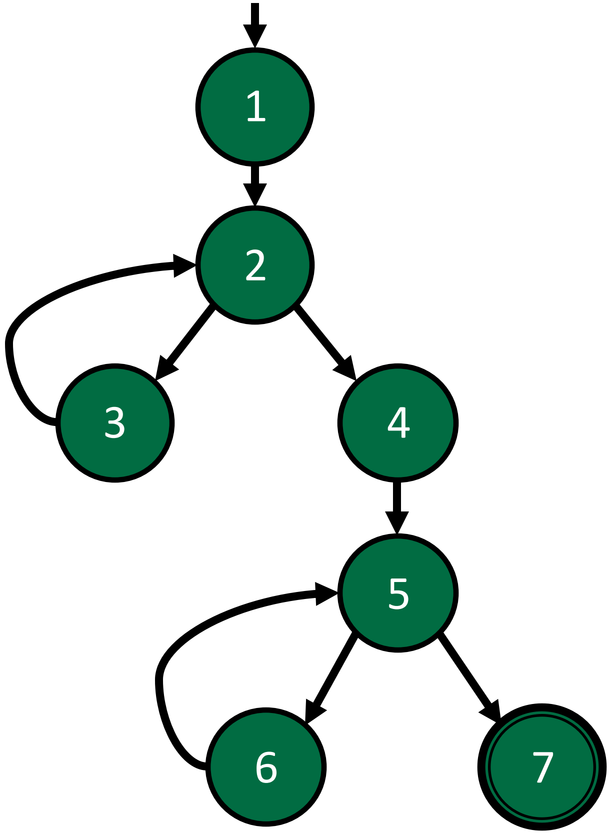


CFG Example: computeStats

```
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);
}
```

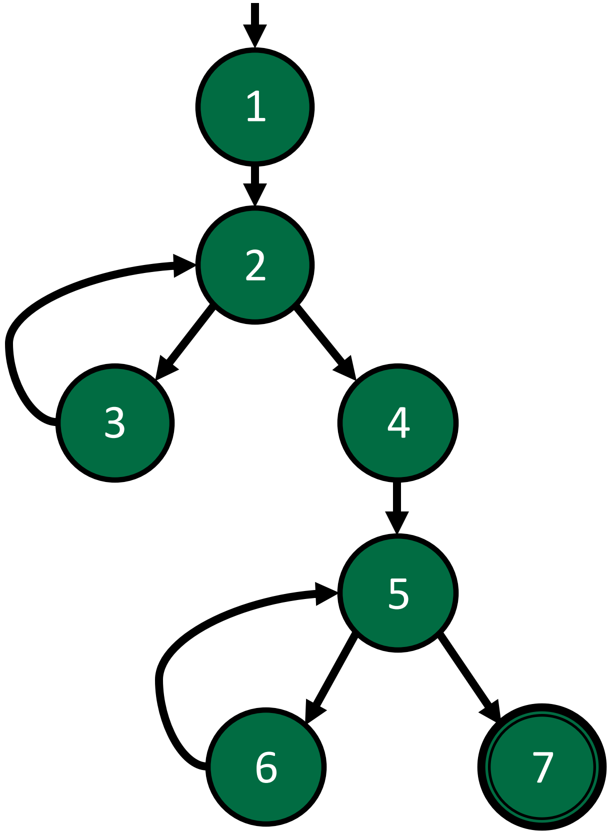


TRs and Test Paths: EC



- Edge Coverage TRs
  - [1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]
- Test paths

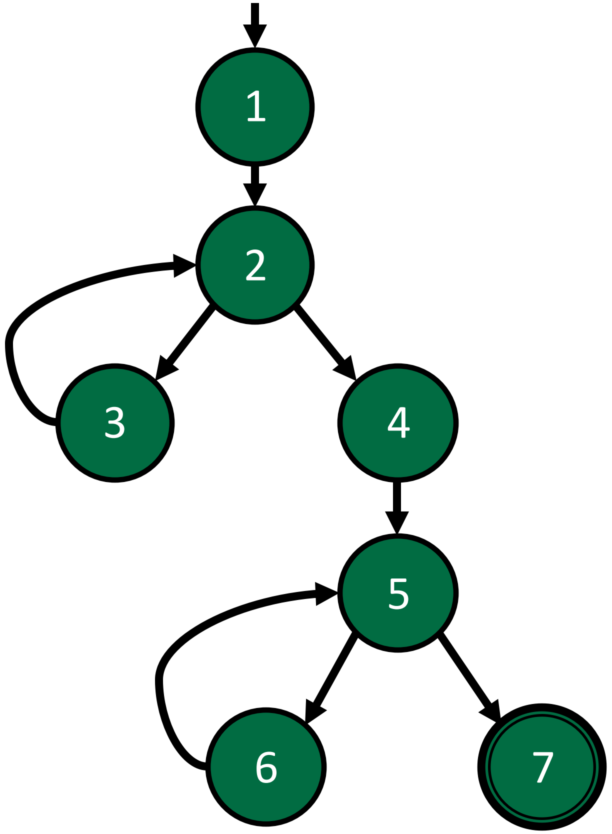
TRs and Test Paths: EC



- 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

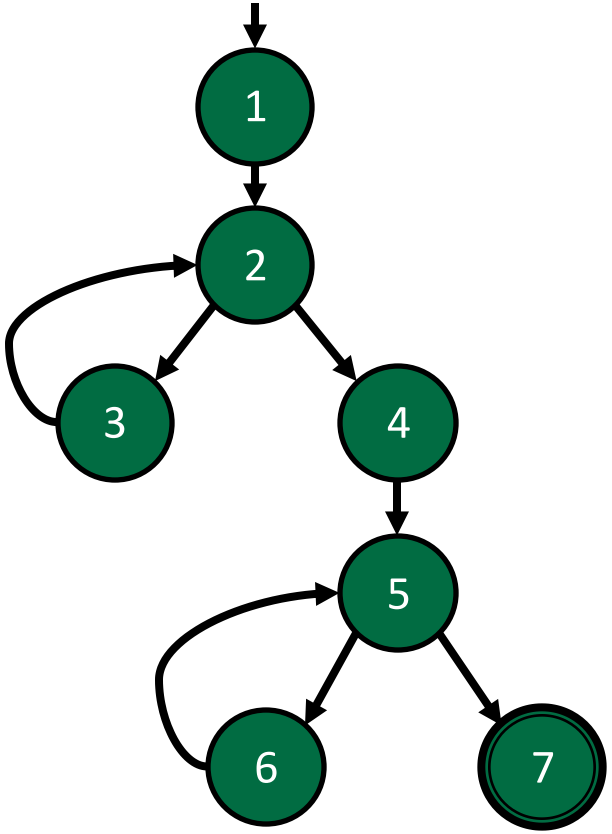
TRs and Test Paths: EC



- 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)

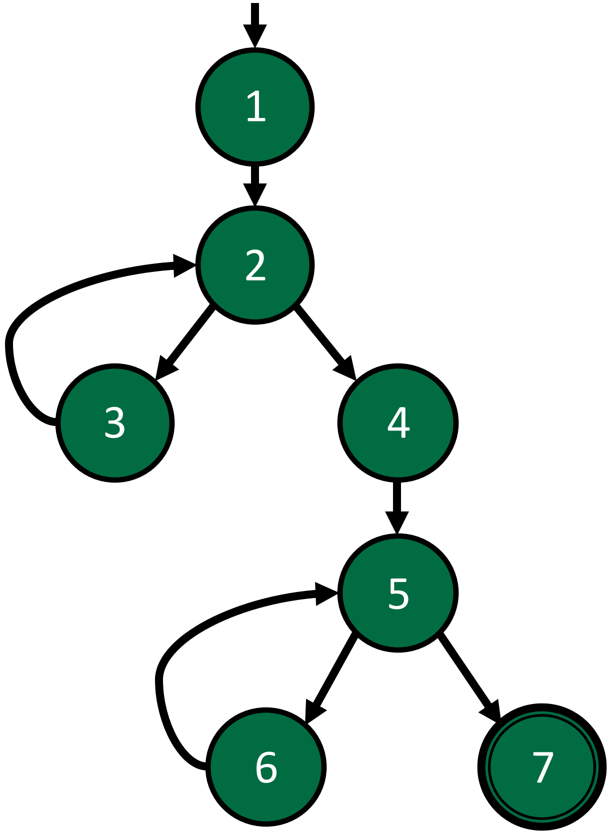
TRs and Test Paths: EC



- 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

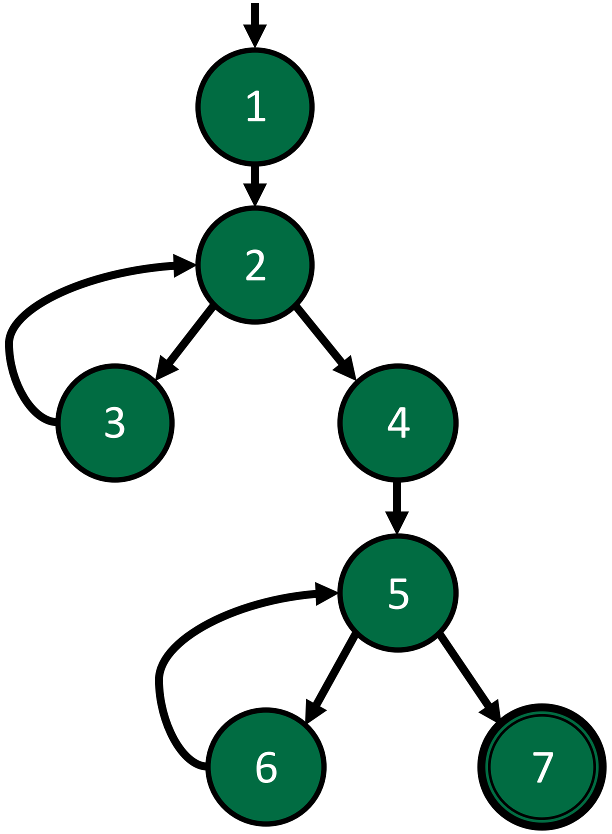
Continue to pick edges that increase coverage

TRs and Test Paths: EC



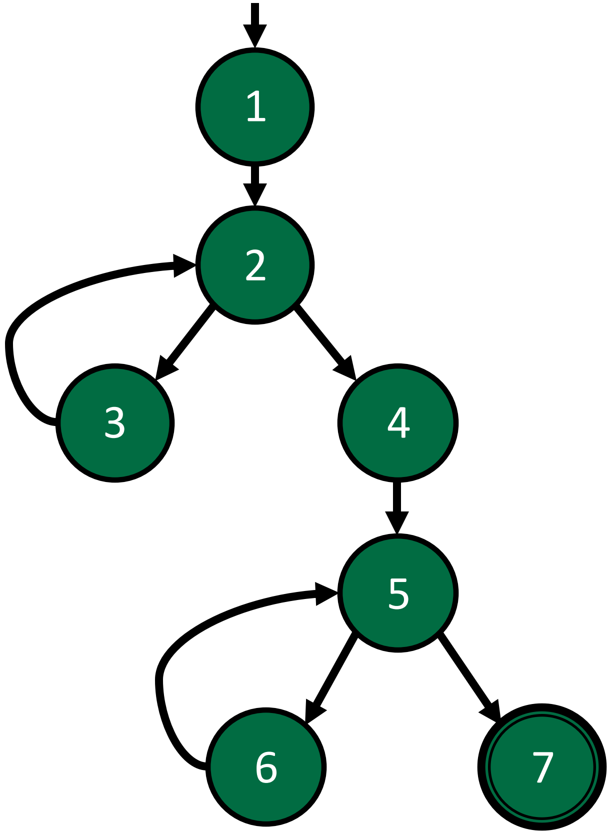
- Edge Coverage TRs
  - [1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]
- Test paths
  - [1,2,3,2,4

TRs and Test Paths: EC



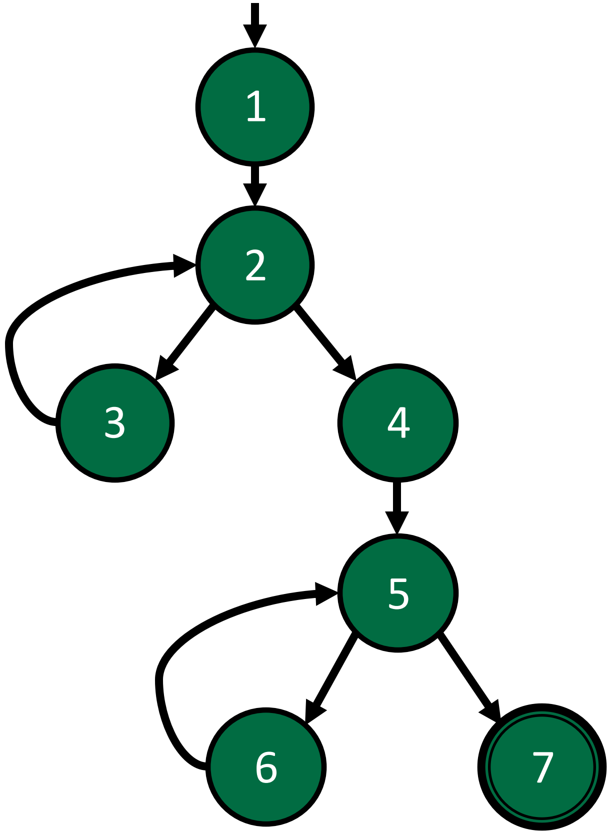
- Edge Coverage TRs
  - [1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]
- Test paths
  - [1,2,3,2,4,5

TRs and Test Paths: EC



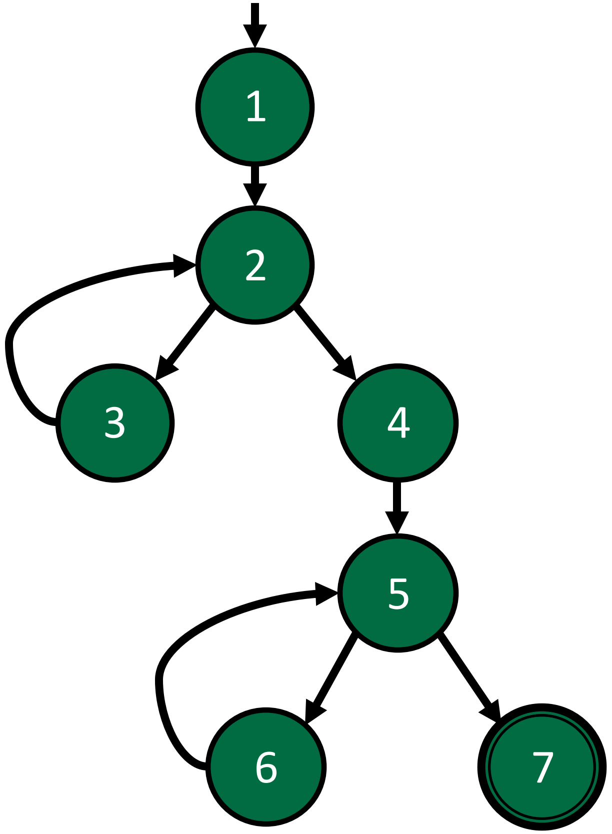
- Edge Coverage TRs
  - [1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]
- Test paths
  - [1,2,3,2,4,5,6

TRs and Test Paths: EC



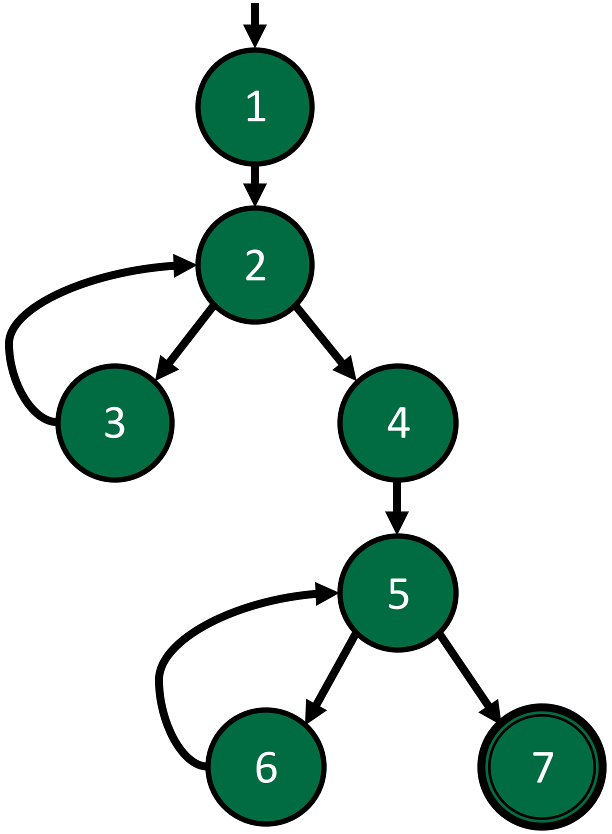
- Edge Coverage TRs
  - [1,2], [2,3], [2,4], [3,2], [4,5], [5,6], [5,7], [6,5]
- Test paths
  - [1,2,3,2,4,5,6,5

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]

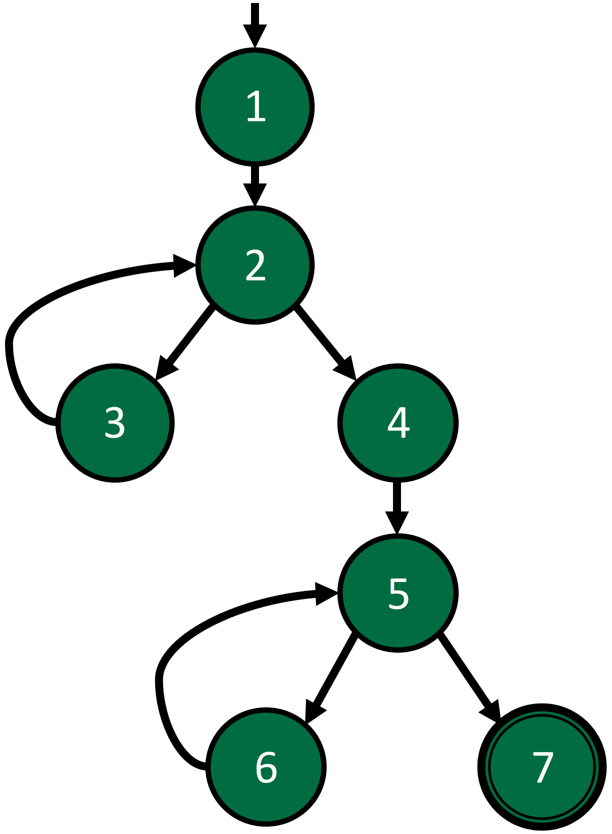
TRs and Test Paths: EC



- 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]

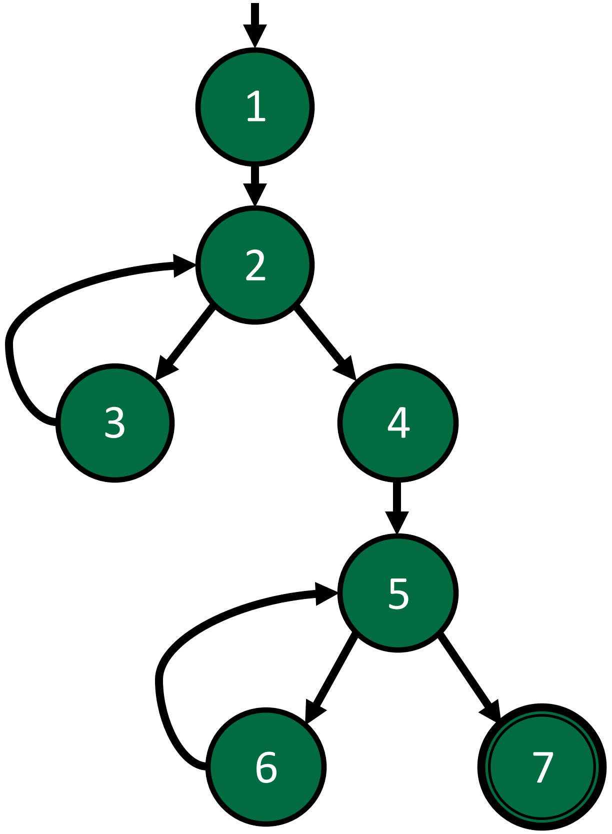
Edge coverage is satisfied with 1 test path

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

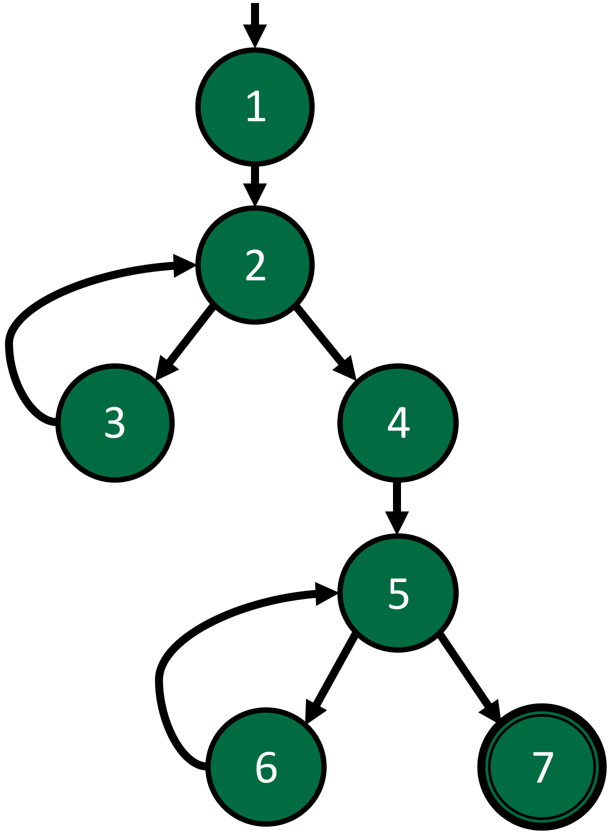
TRs and Test Paths: EPC



- 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

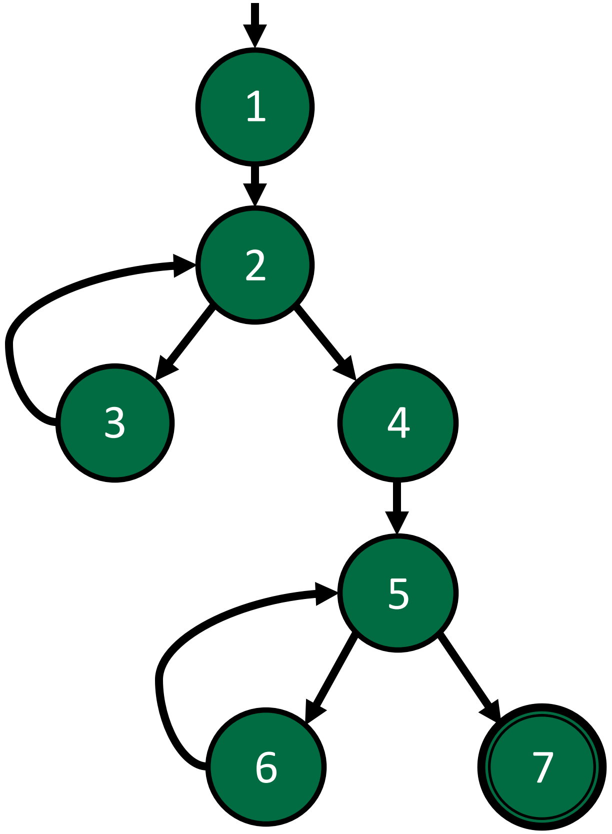
TRs and Test Paths: EPC



- 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
  -

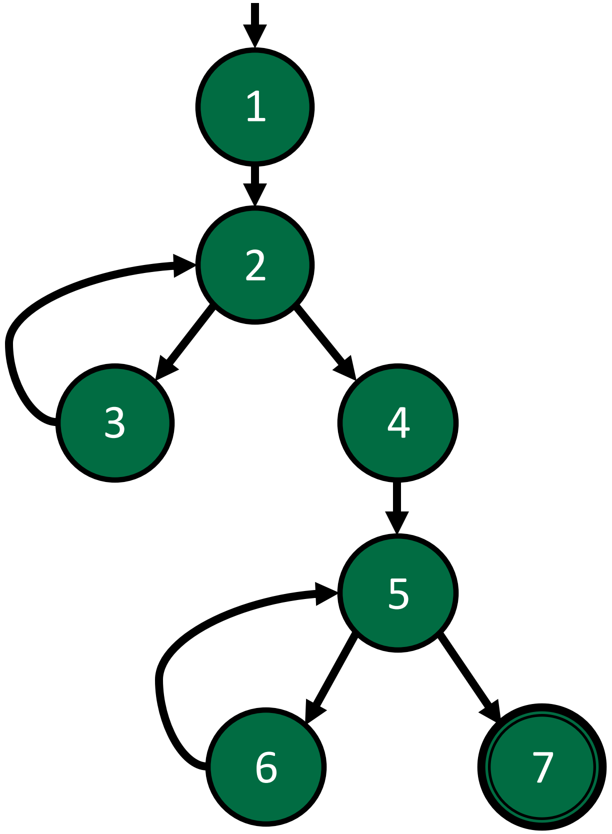
Select an edge that increases edge-pair coverage

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
  -

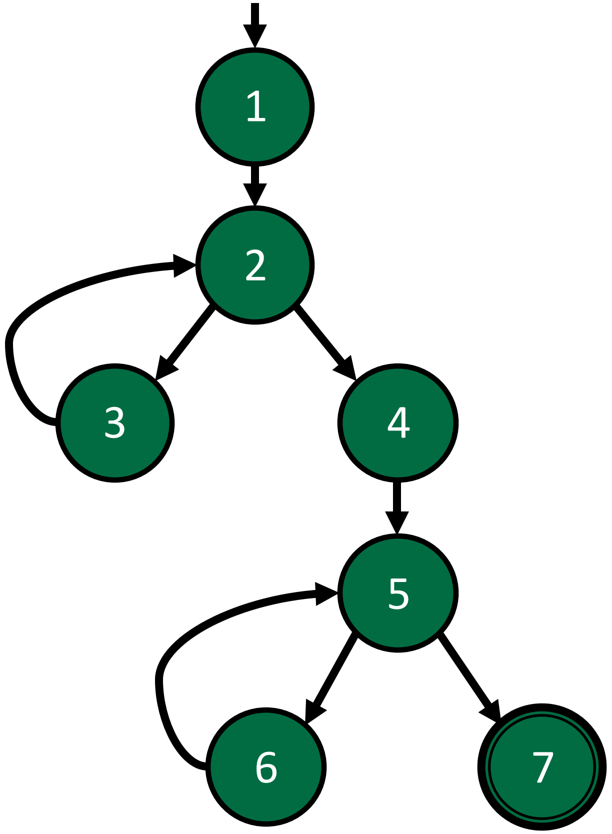
TRs and Test Paths: EPC



- 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
  -

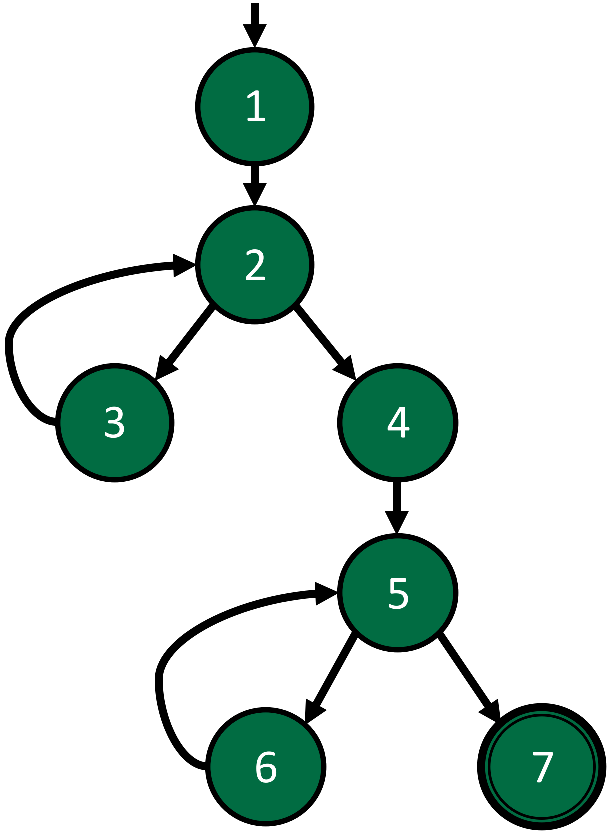
It's not always possible to increase coverage with every selected edge

TRs and Test Paths: EPC



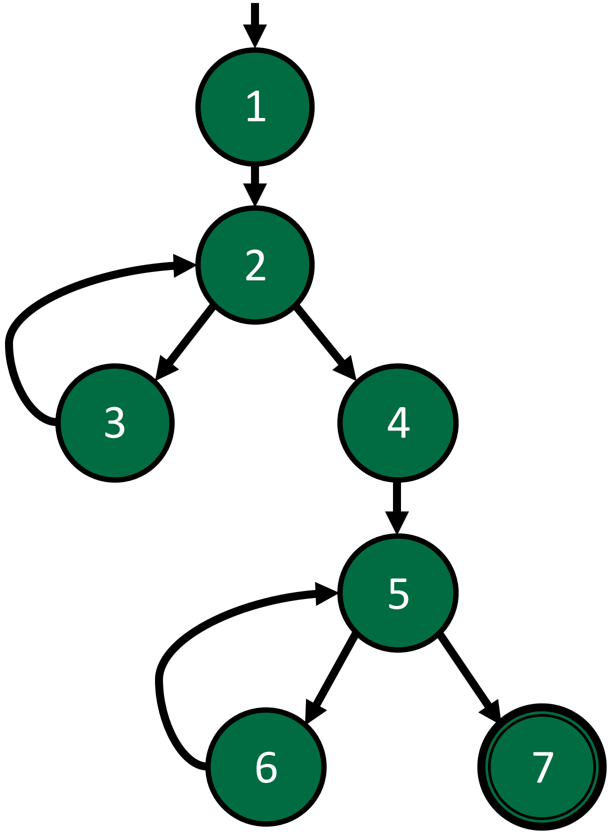
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4
  -

TRs and Test Paths: EPC



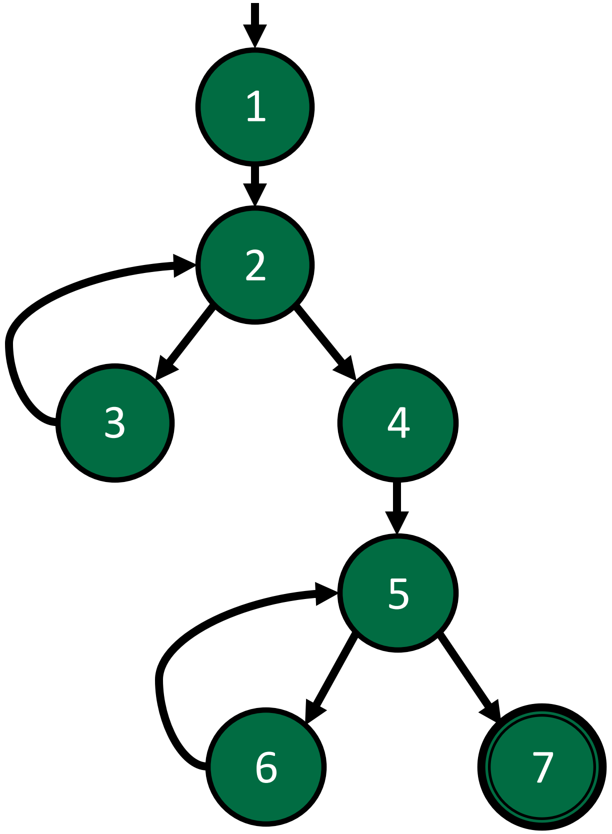
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5
  -

TRs and Test Paths: EPC



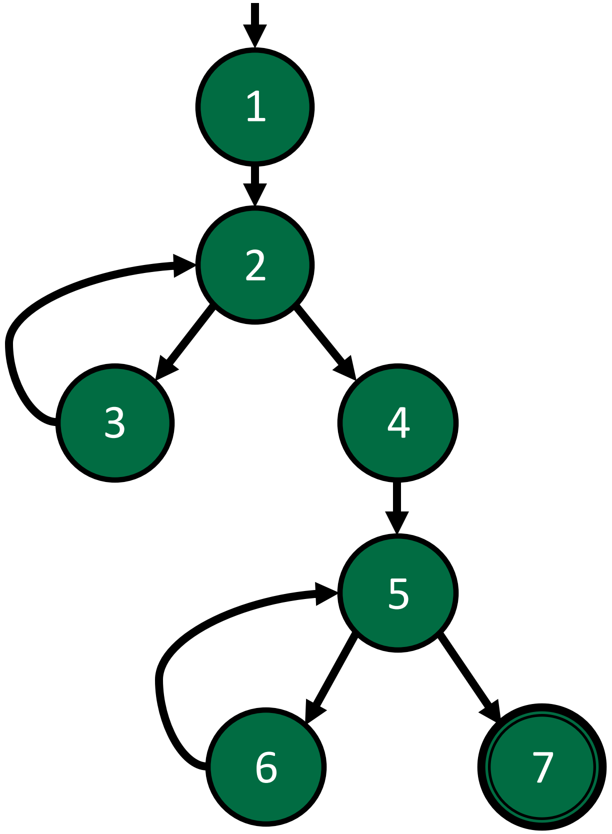
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6
  -

TRs and Test Paths: EPC



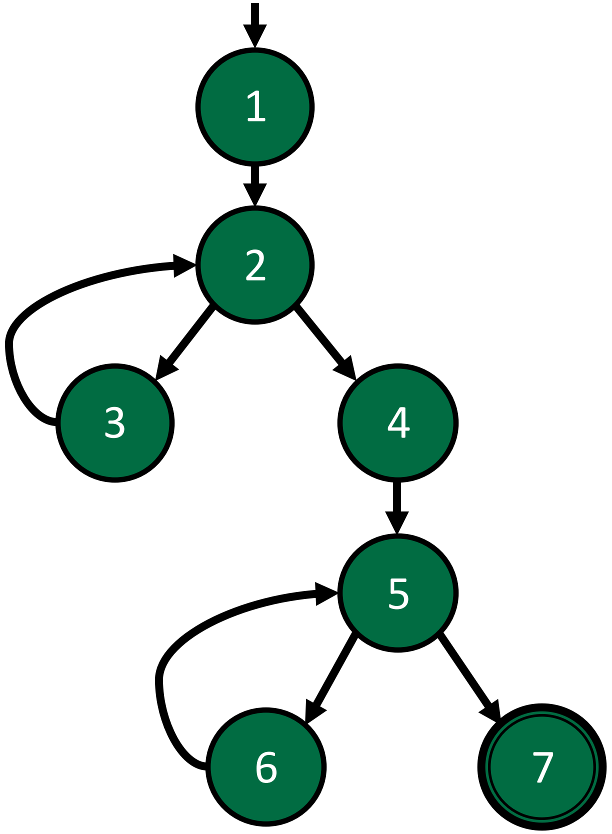
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6,5
  -

TRs and Test Paths: EPC



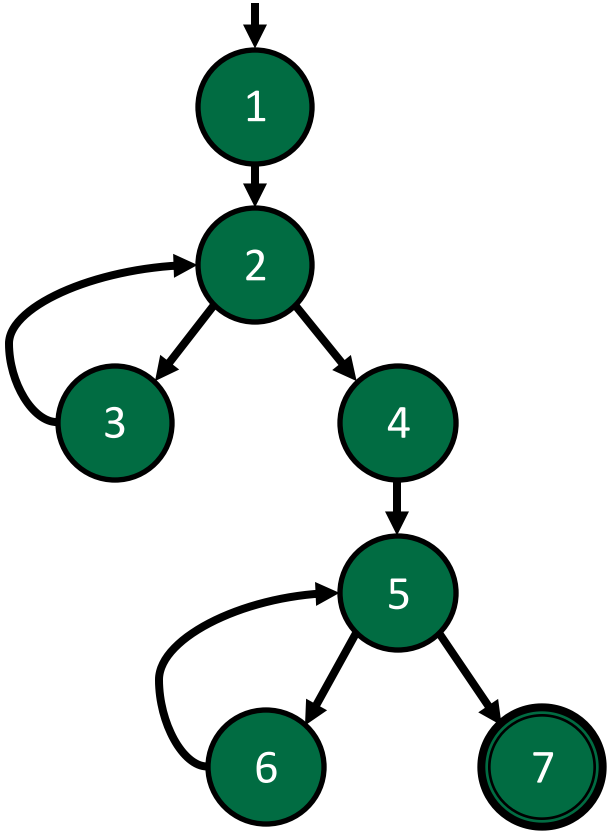
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6,5,6
  -

TRs and Test Paths: EPC



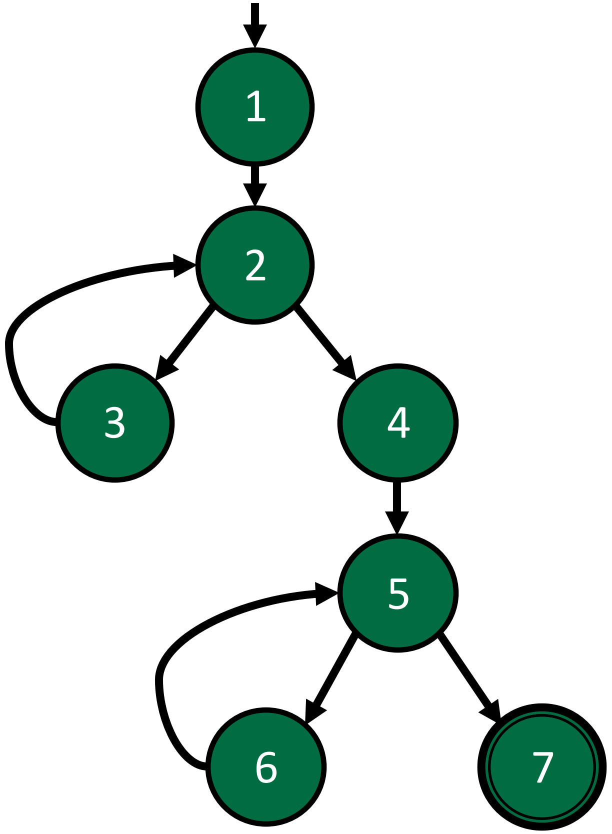
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6,5,6,5
  -

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]
  -

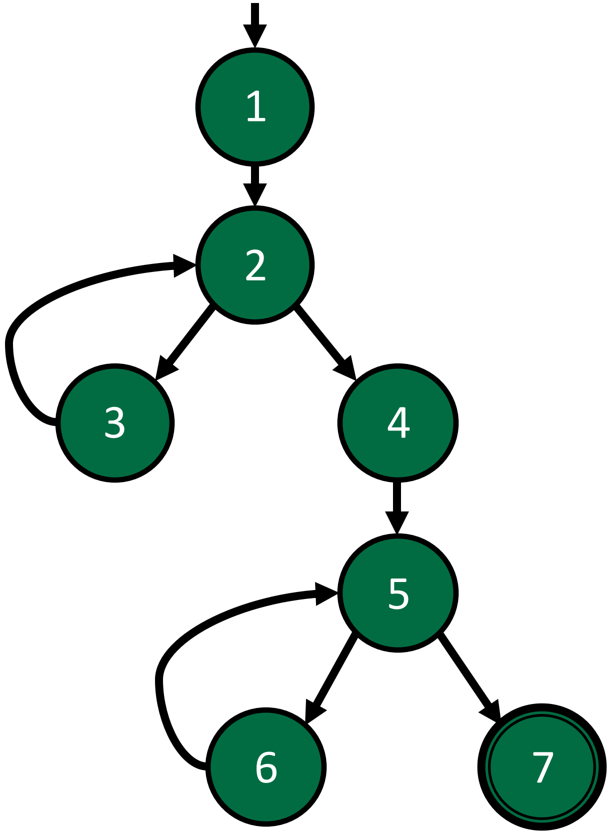
TRs and Test Paths: EPC



- 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]

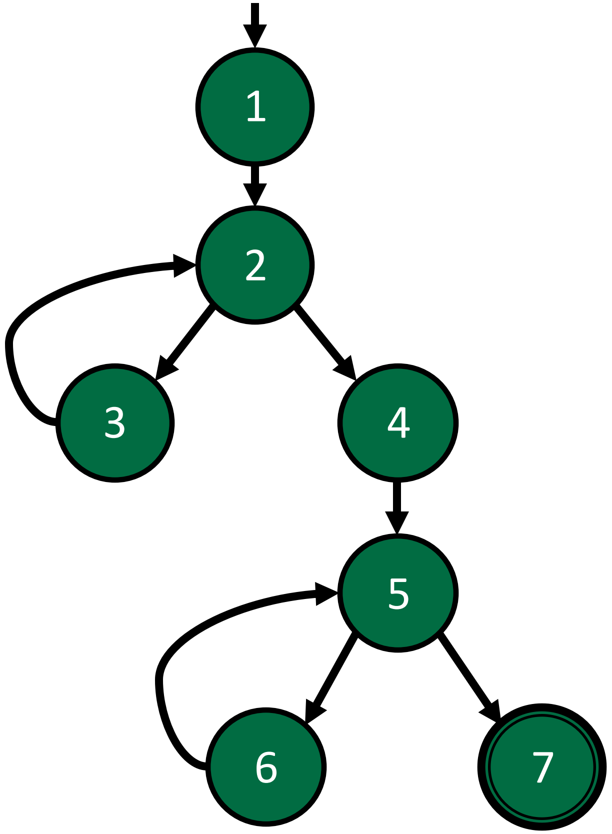
We need another test path to achieve edge-pair coverage

TRs and Test Paths: EPC



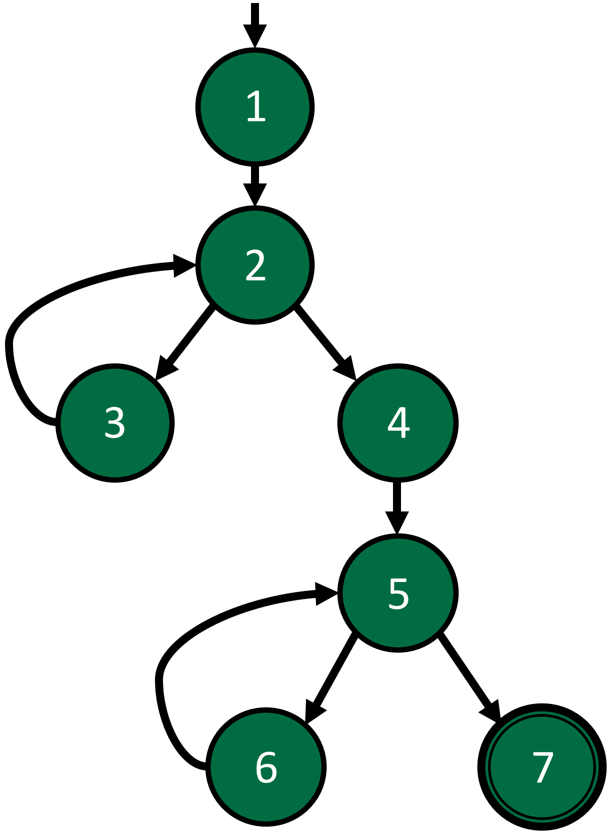
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6,5,6,5,7]
  - [1,2,4

TRs and Test Paths: EPC



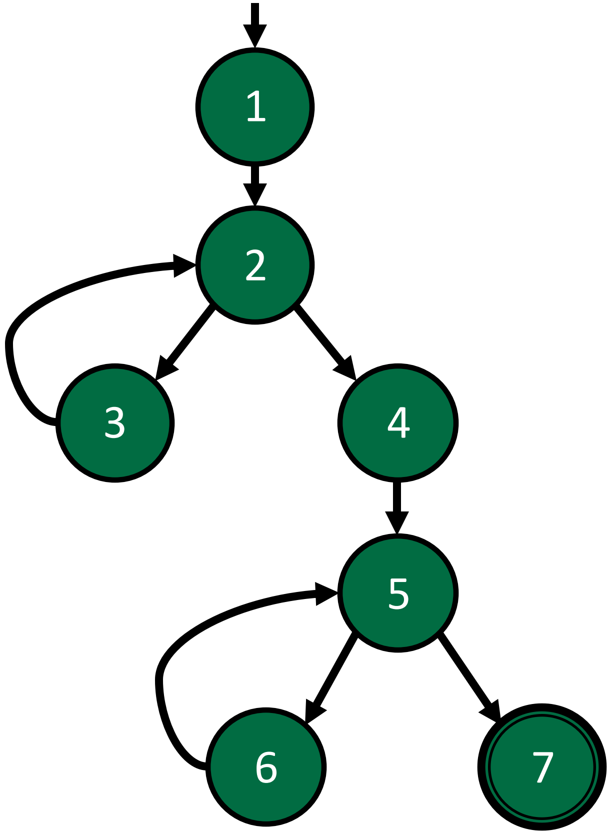
- Edge-Pair TRs
  - [1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6,5,6,5,7]
  - [1,2,4,5

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]

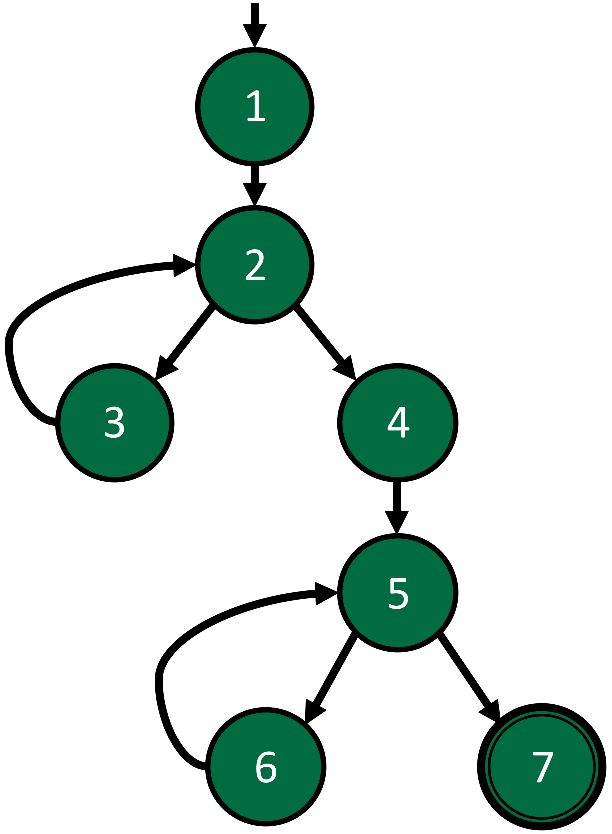
TRs and Test Paths: EPC



- 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]

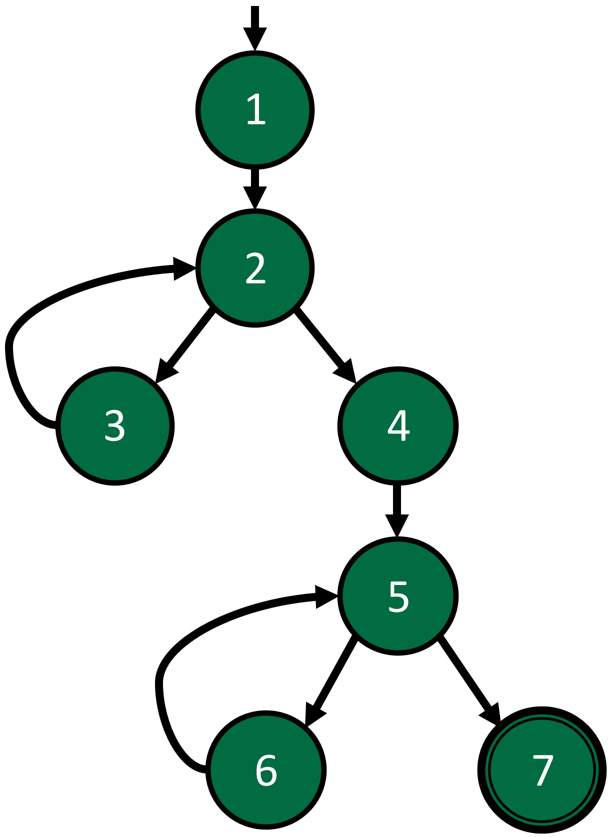
Edge-pair coverage is satisfied with 2 test paths

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
  -
  -
  -
  -

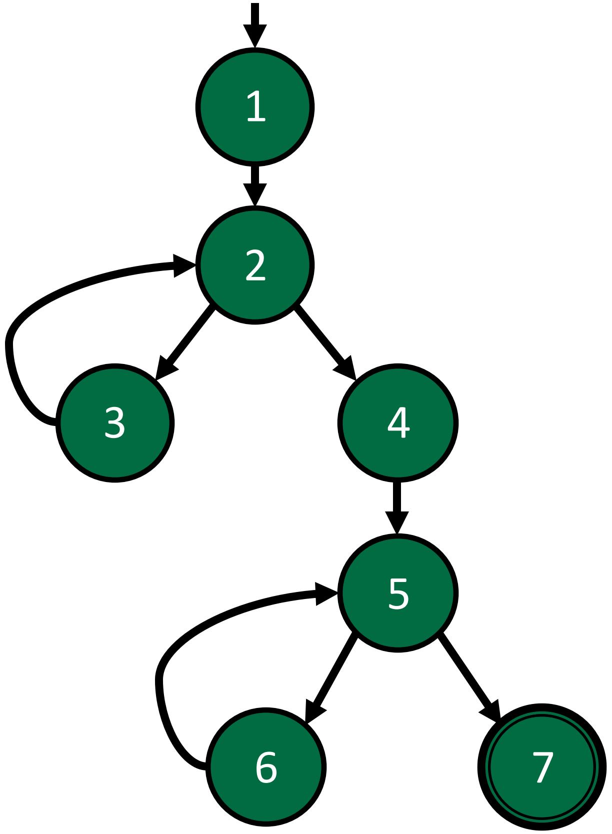
TRs and Test Paths: PPC



- 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

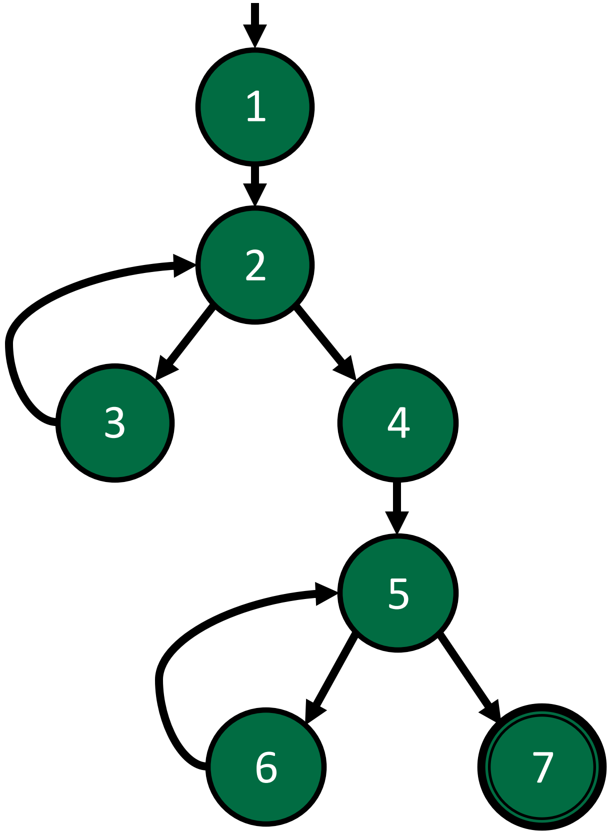
TRs and Test Paths: PPC



- 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]
  -
  -

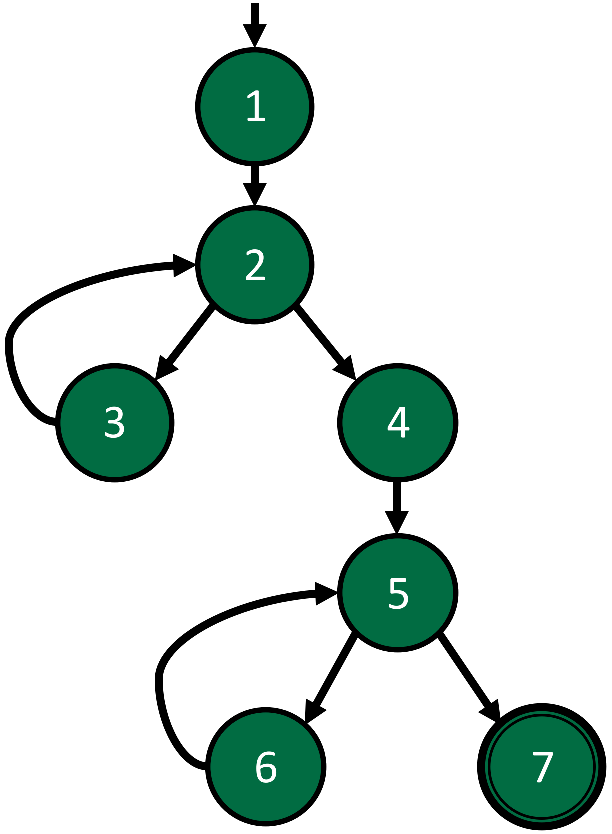
Add additional test paths to capture the remaining TRs

TRs and Test Paths: PPC



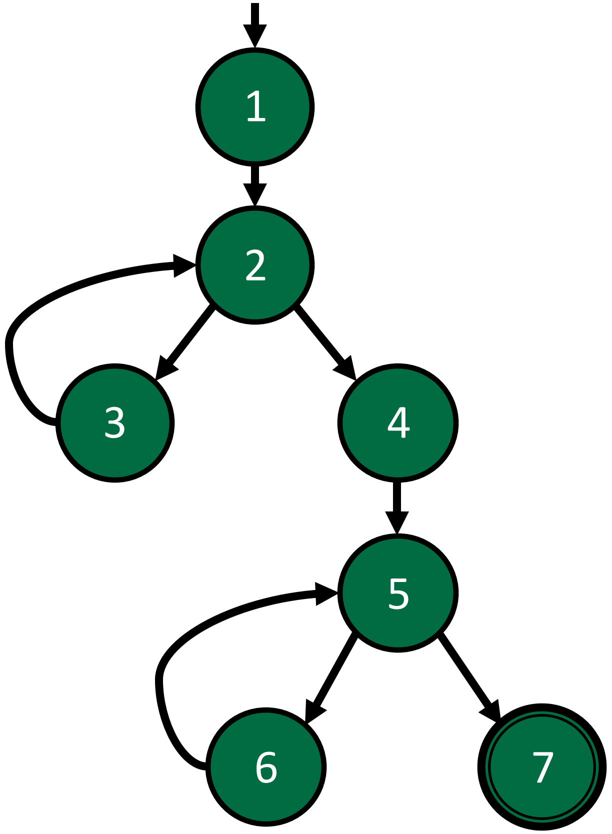
- Prime Path TRs
  - [1,2,3], [1,2,4,5,6], [1,2,4,5,7], [2,3,2], [3,2,3], [3,2,4,5,6], [3,2,4,5,7], [5,6,5], [6,5,6], [6,5,7]
- Test paths
  - [1,2,3,2,3,2,4,5,6,5,6,5,7]
  - [1,2,4,5,7]
  - [1,2,4,5,6,5,7]

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]

Data Flow Coverage for Source

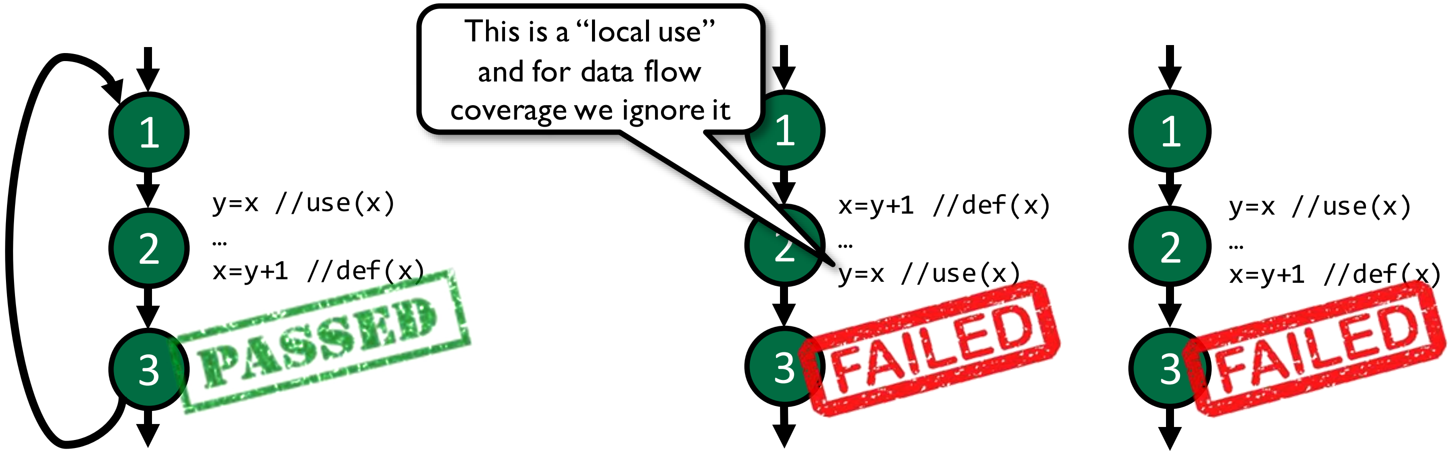
- ***Def***: a location where a value is *stored* into memory
  - Variable appears on the *left side* of an assignment (e.g. x=44)
  - Variable is an *actual parameter* in a call and the method *changes* its value
  - Variable is a *formal parameter* of a method (implicit def when the method is called)
- ***Use***: a location where a variable is *accessed*
  - Variable appears on the *right side* of an assignment
  - Variable appears in a *conditional* test
  - Variable is an *actual parameter* in a call
  - Variable is an *output* of the program
  - Variable is used in a *return* statement

Data Flow Definitions

- ***DU-pair***: a related *def* and *use*, where the *use* can be reached from the *def*
  - The pair does not need to be *def-clear*
- ***Def-clear***: a path from a *def* to a *use* is *def-clear* if there are no redefinitions of the variable along the path
- ***DU-path***: a *simple path* from a *def* to a *use* that is *def-clear*

DU-Pairs in the Same Node

- A def and use **are** a DU-pair only if:
  - The *def* comes after the *use* within the node, and the node is in a loop
- A def and use **are not** a DU-pair if:
  - The *use* comes after the *def*, or…
  - The *def* comes after the *use*, but the node is not in a loop

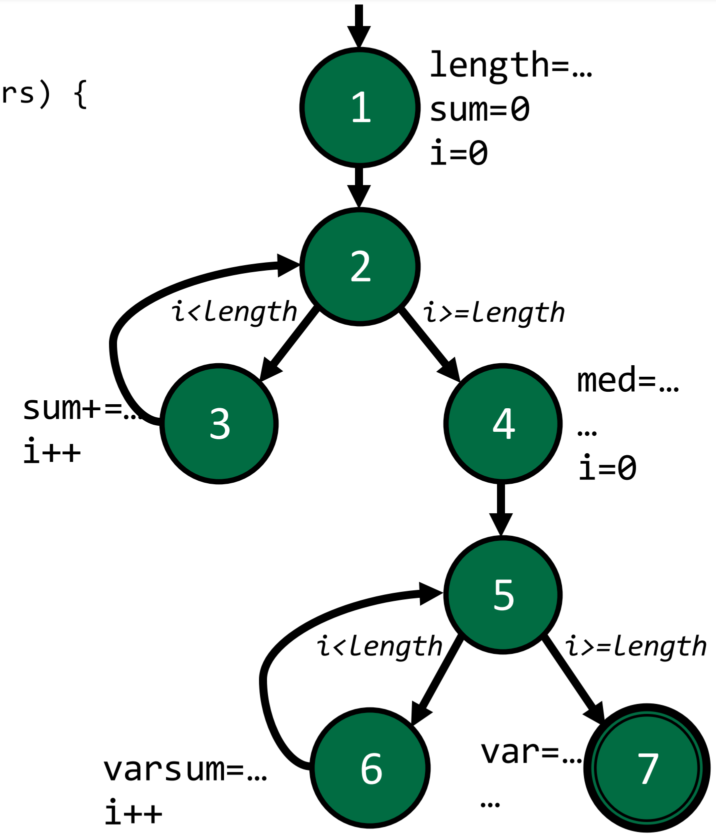


This is a "local use" and for data flow coverage we ignore it

Collaborative Example

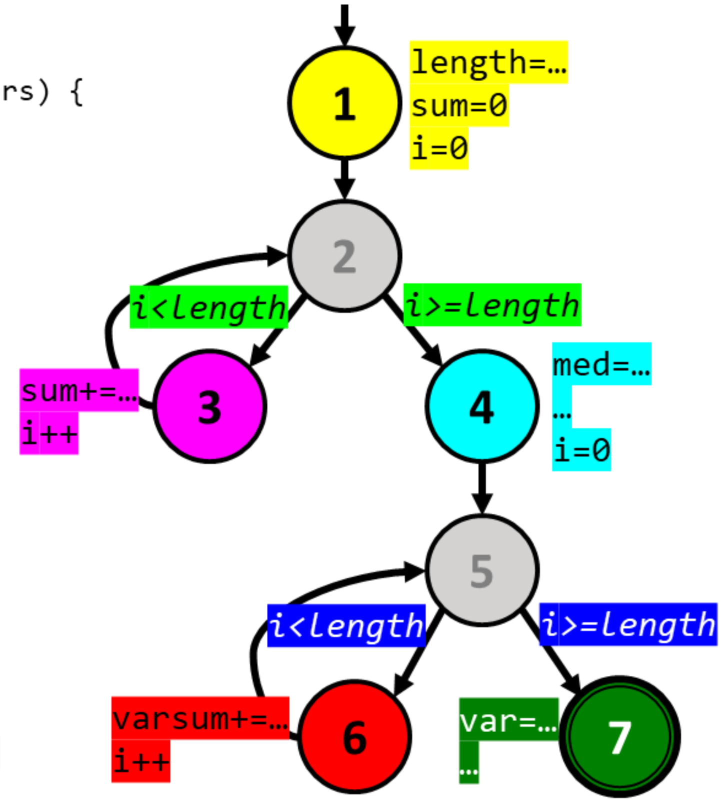
Data Flow Example: computeStats

```
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);
}
```



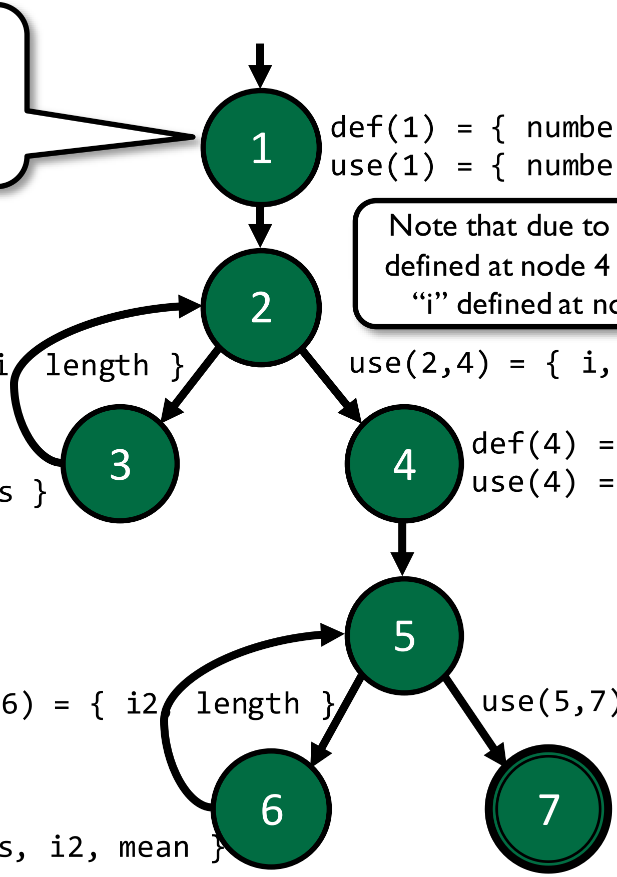
Data Flow Example: computeStats

```
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);
}
```



Data Flow Example: computeStats

Convert the code annotations into def and use sets



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

Note that due to Java scoping rules, variable "i" defined at node 4 is a different variable than the "i" defined at node 1; we'll call this one "i2"

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

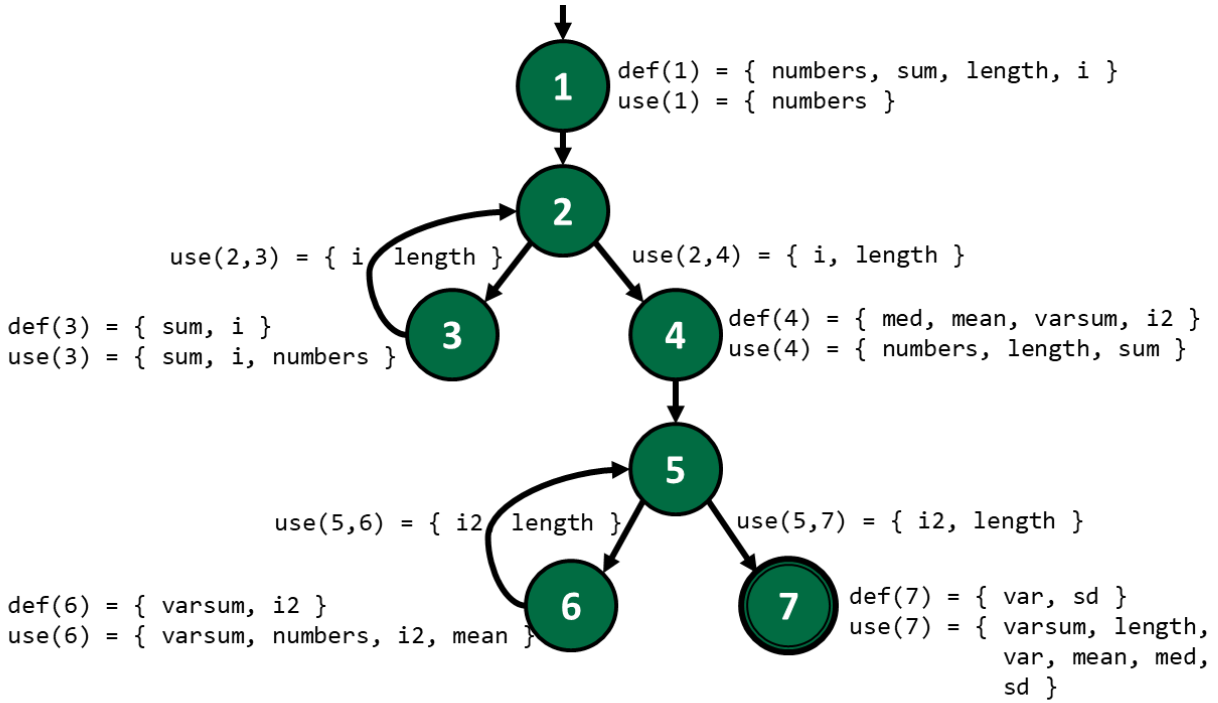
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Def/Use Tables for computeStats

Node	Defs	Uses
1		
2		
3		
4		
5		
6		
7		

Edge	Uses
(1,2)	
(2,3)	
(2,4)	
(3,2)	
(4,5)	
(5,6)	
(5,7)	
(6,5)	

Def/Use for Node 1



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

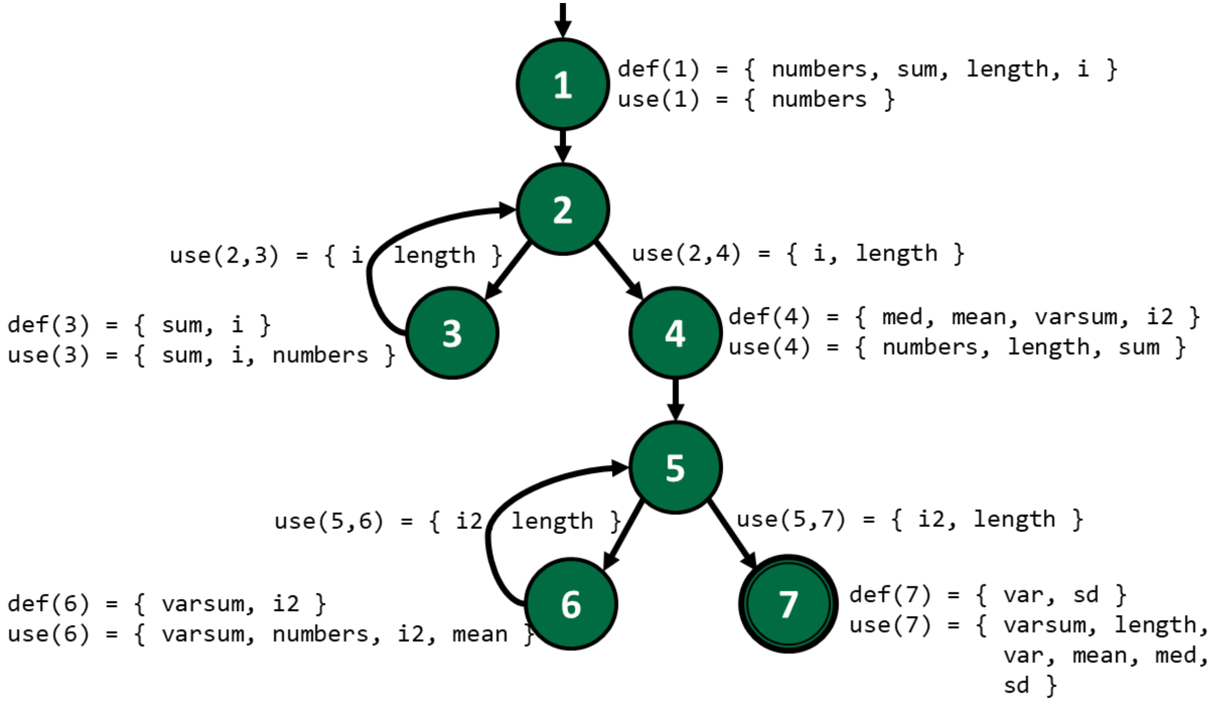
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
1		

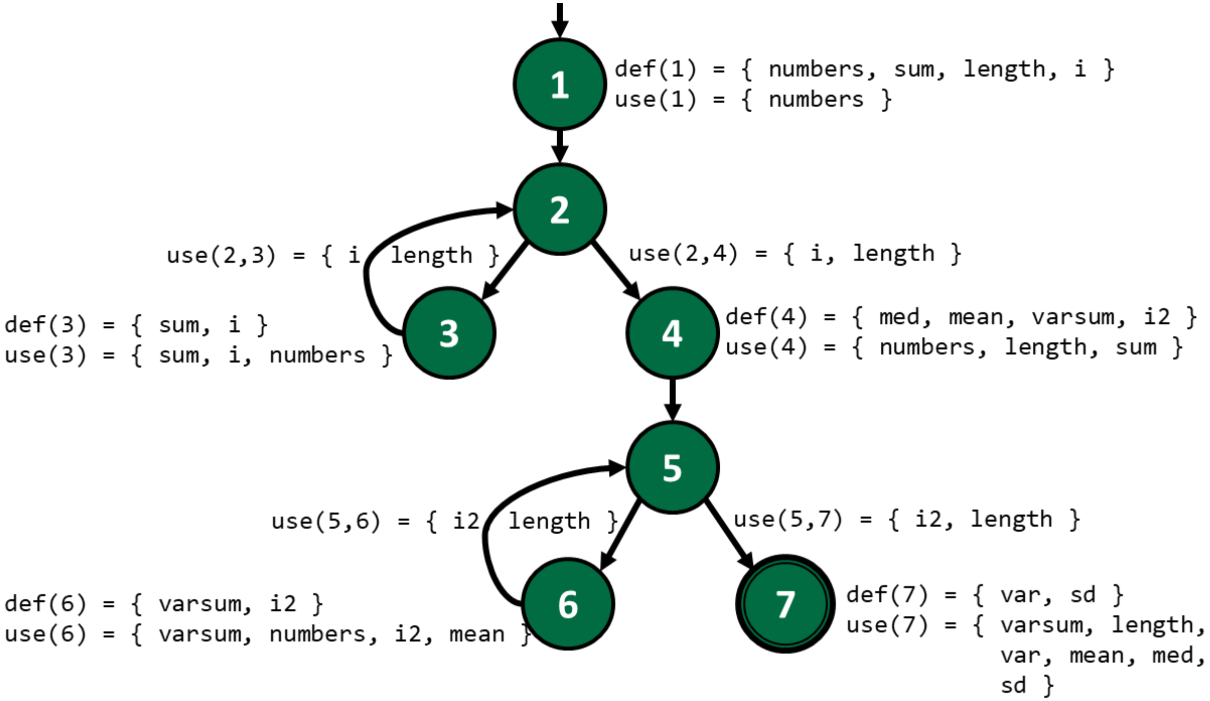
Def/Use for Node 1



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

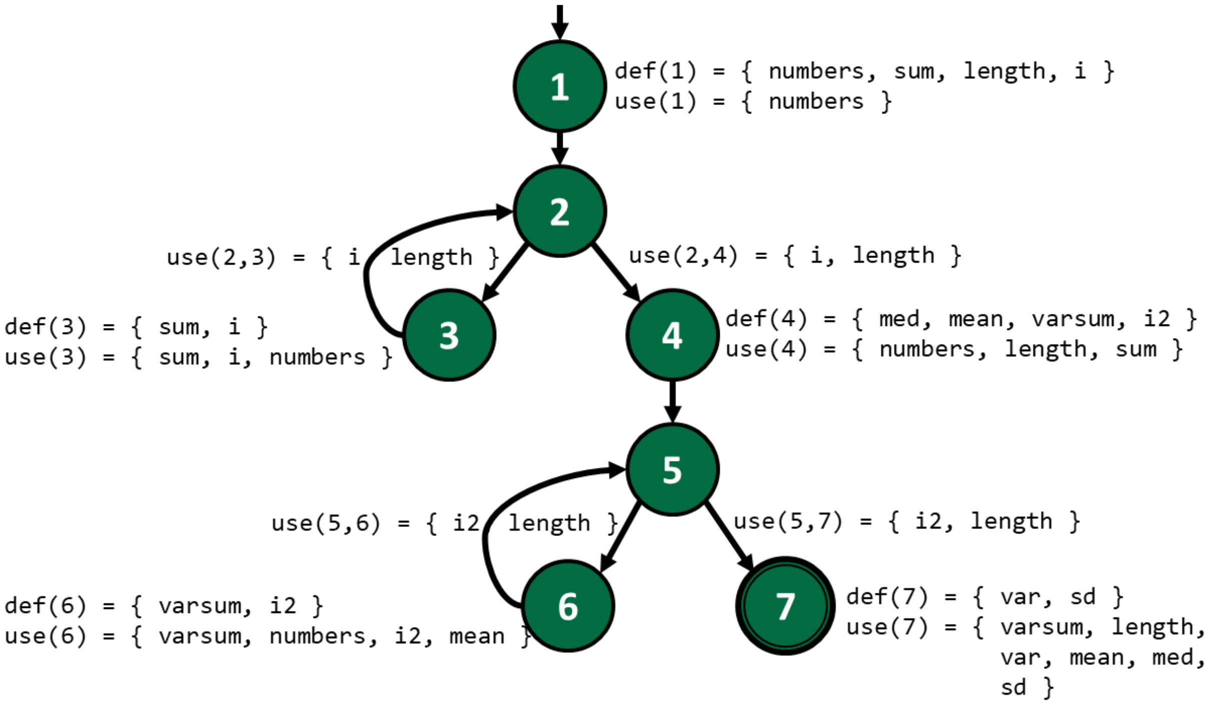
Node	Defs	Uses
1	{ numbers, sum, length, i }	

Def/Use for Node 1


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }

Def/Use for Node 2



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

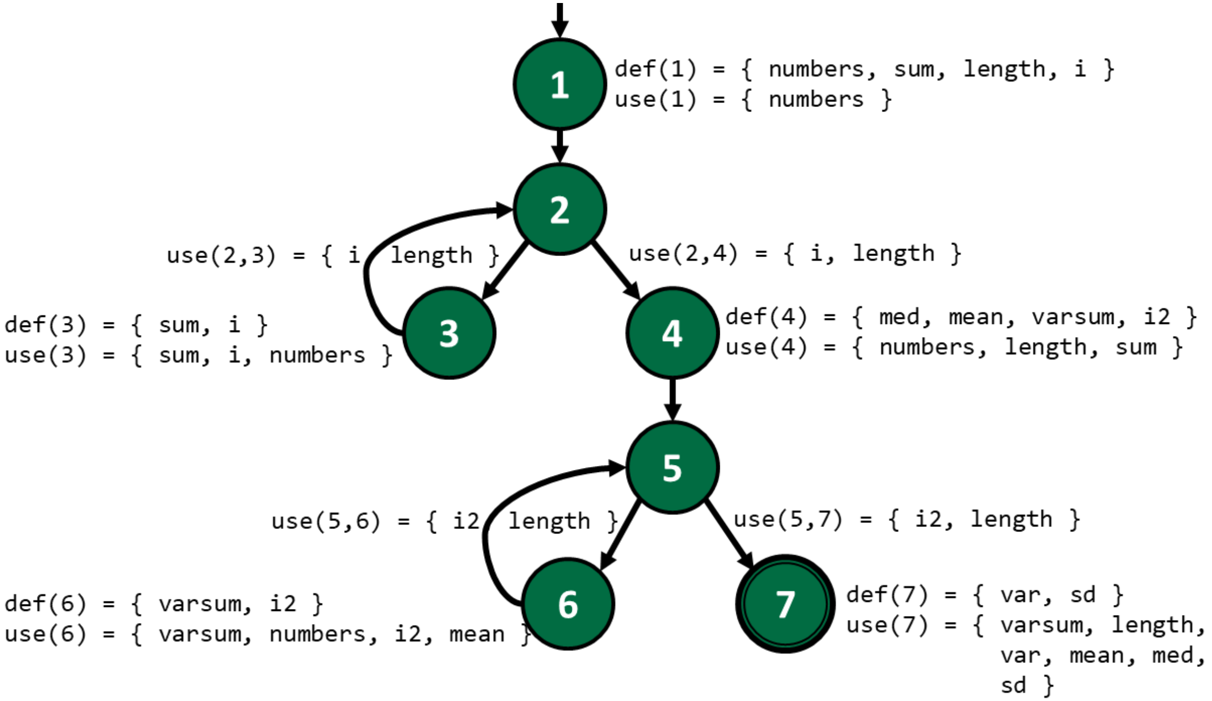
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
2		

Def/Use for Node 2



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

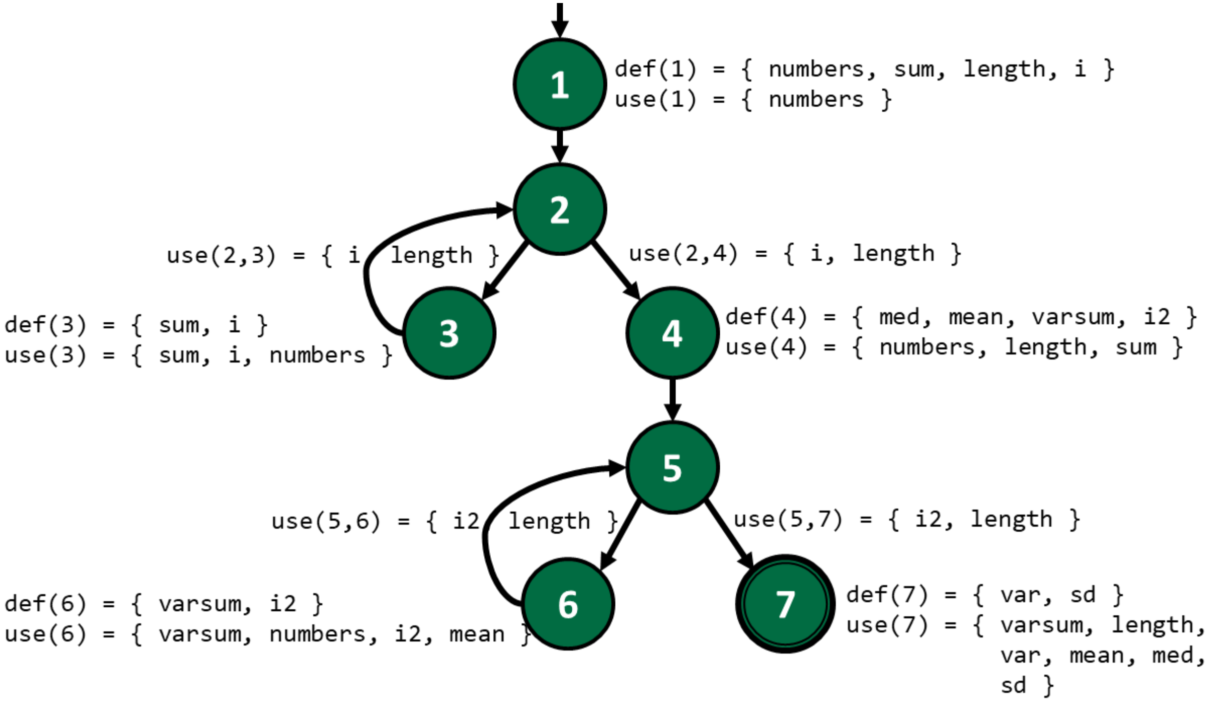
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
2	--	--

Def/Use for Node 3



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

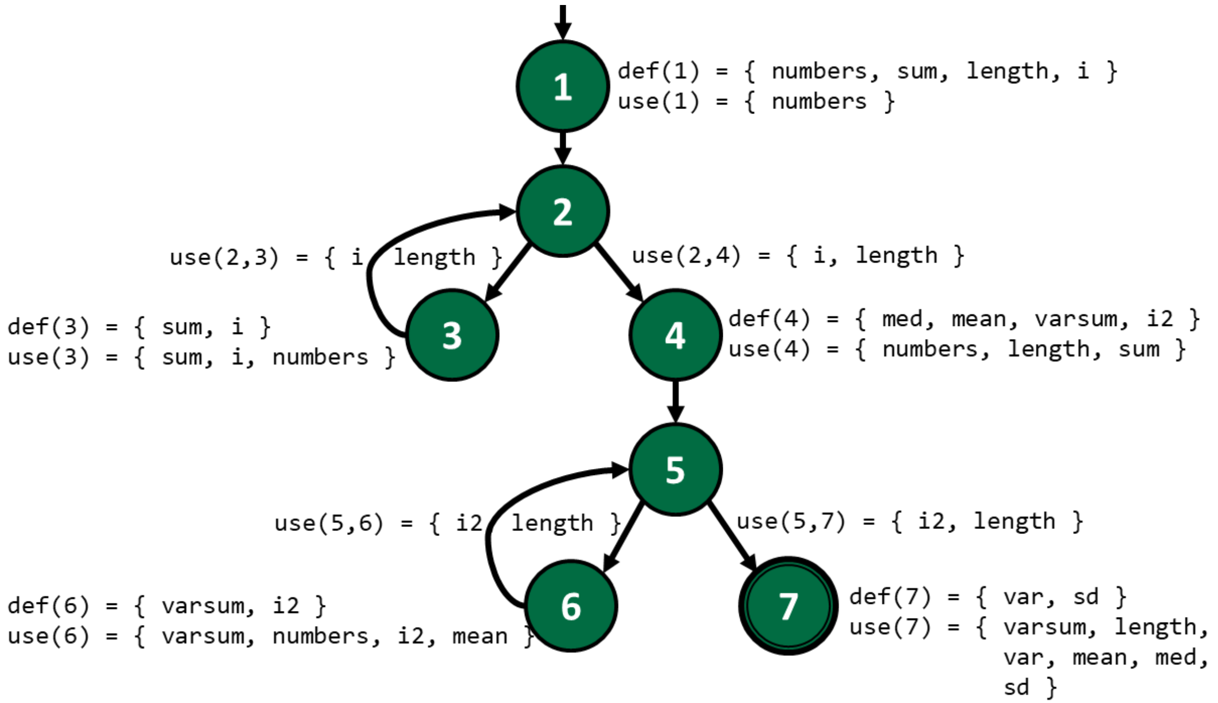
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
3		

Def/Use for Node 3



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

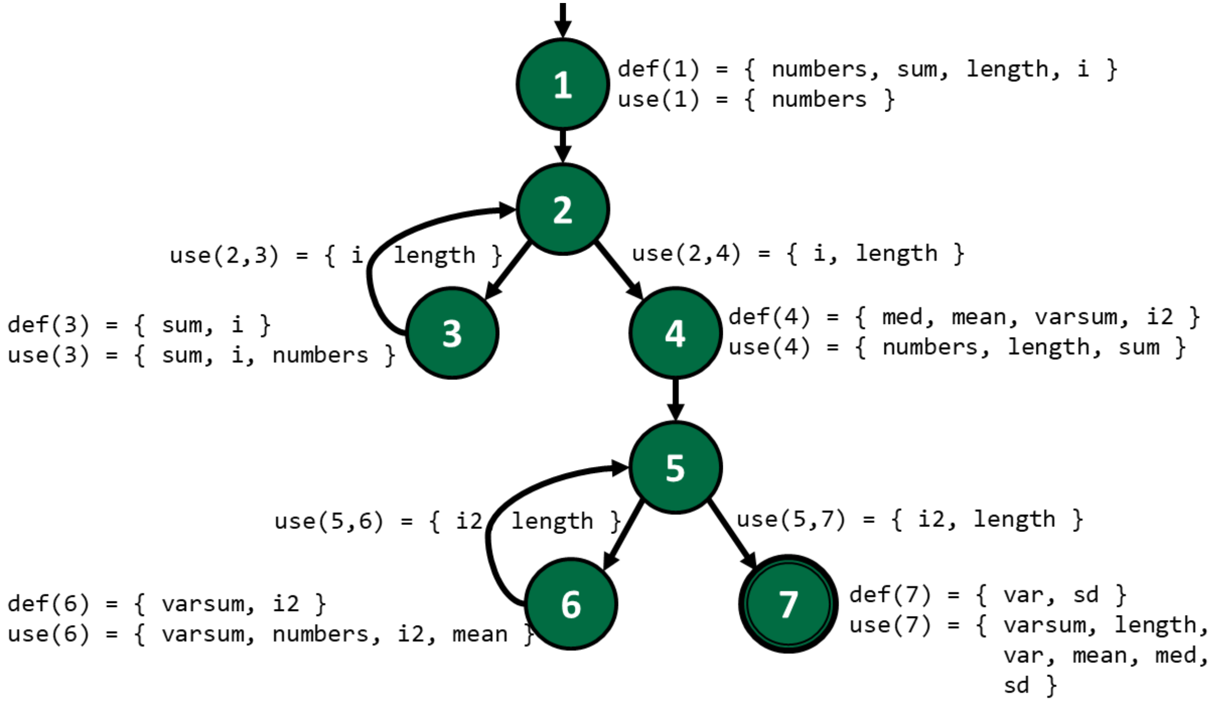
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
3	{ sum, i }	{ sum, i, numbers }

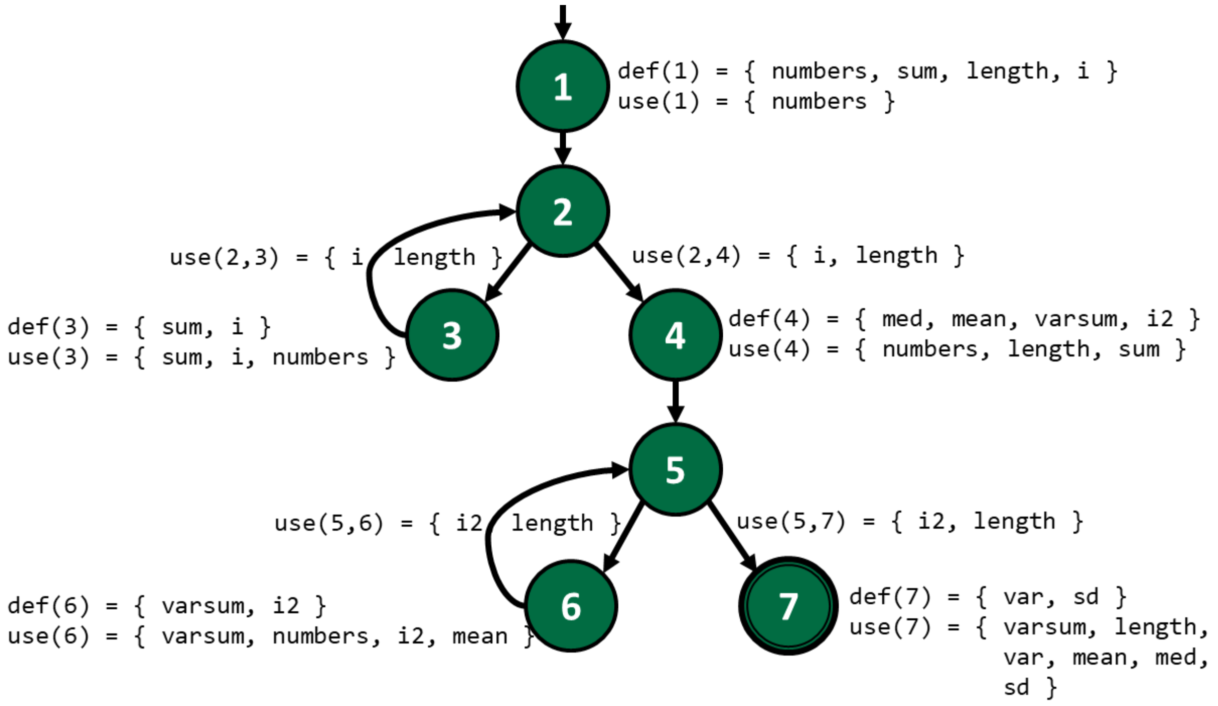
Def/Use for Node 4



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
4		

Def/Use for Node 4



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

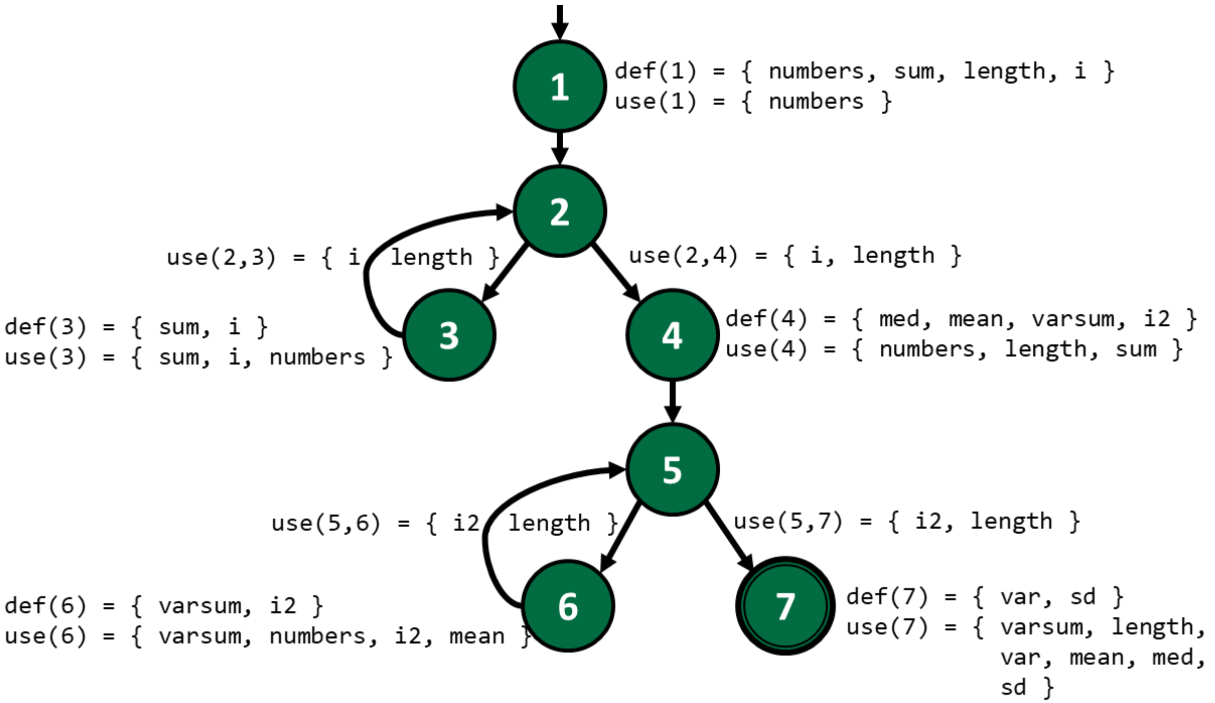
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }

Def/Use for Node 5



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

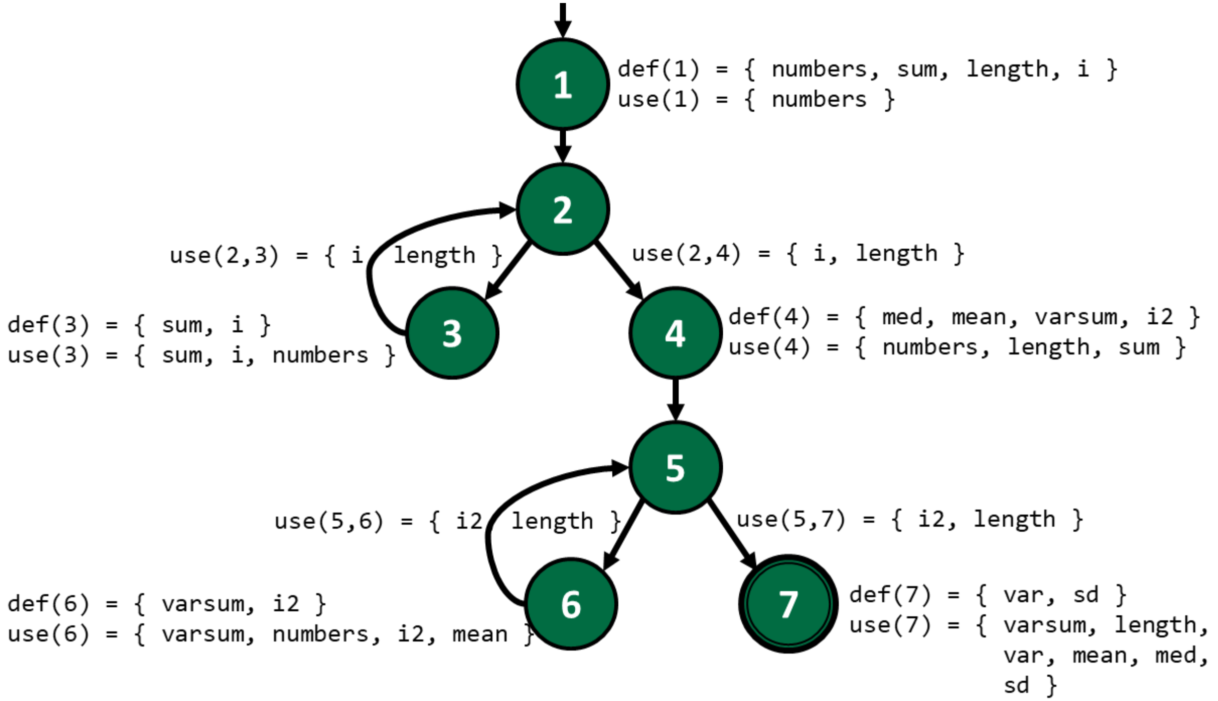
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
var, mean, med,
sd }

Node	Defs	Uses
5		

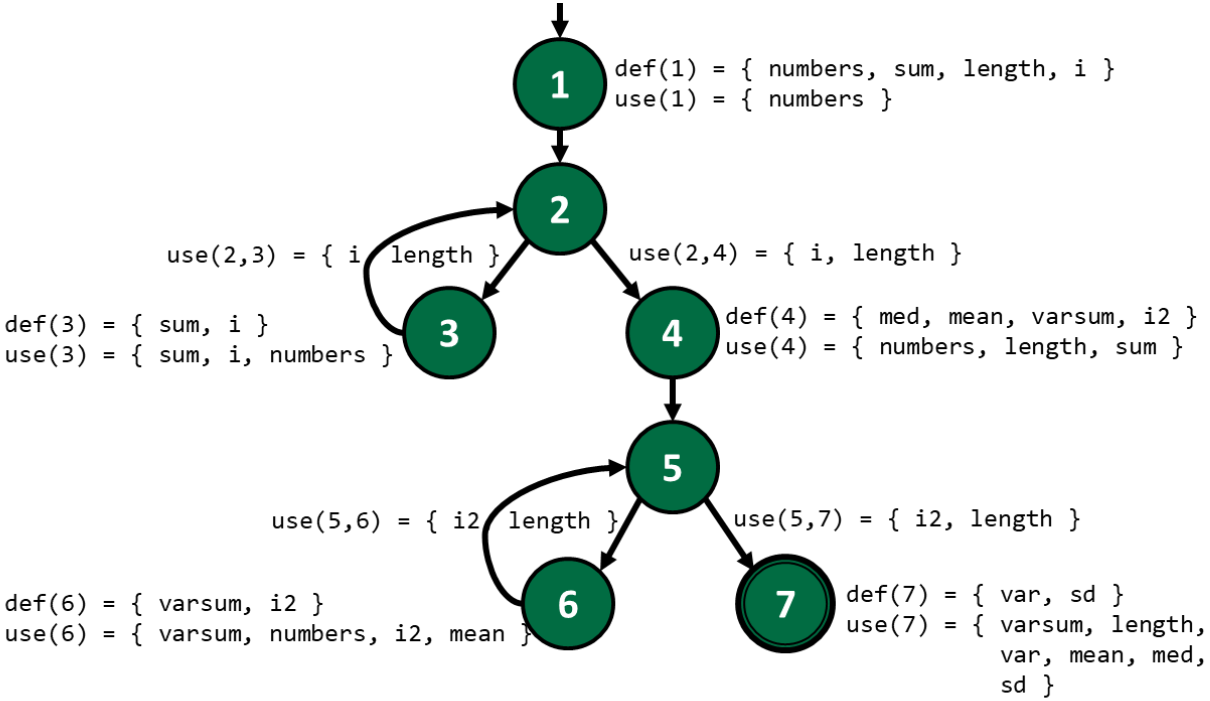
Def/Use for Node 5



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
5	--	--

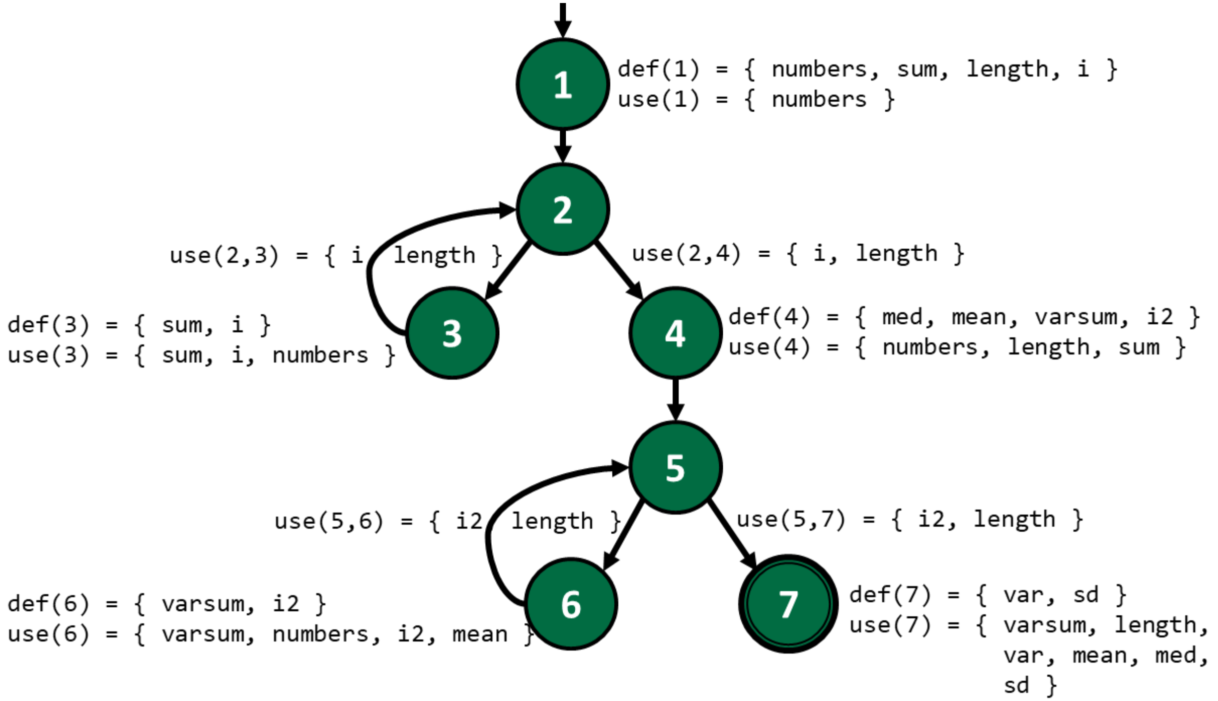
Def/Use for Node 6



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

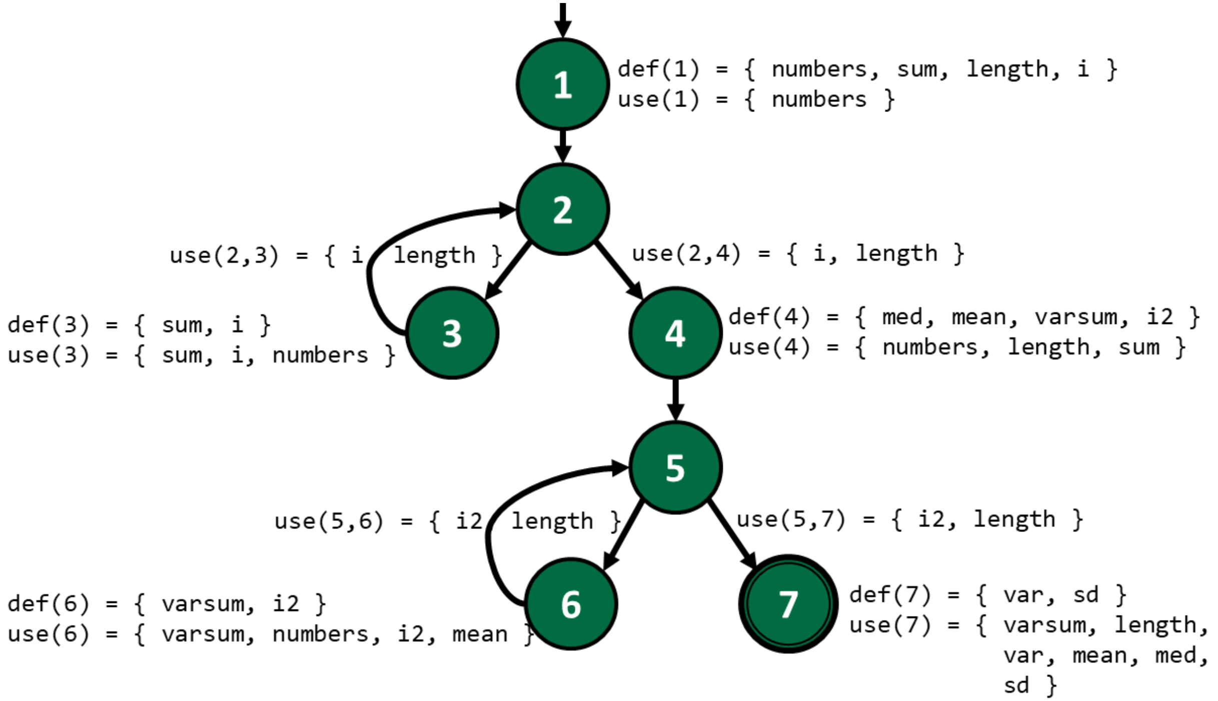
Node	Defs	Uses
6		

Def/Use for Node 6


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

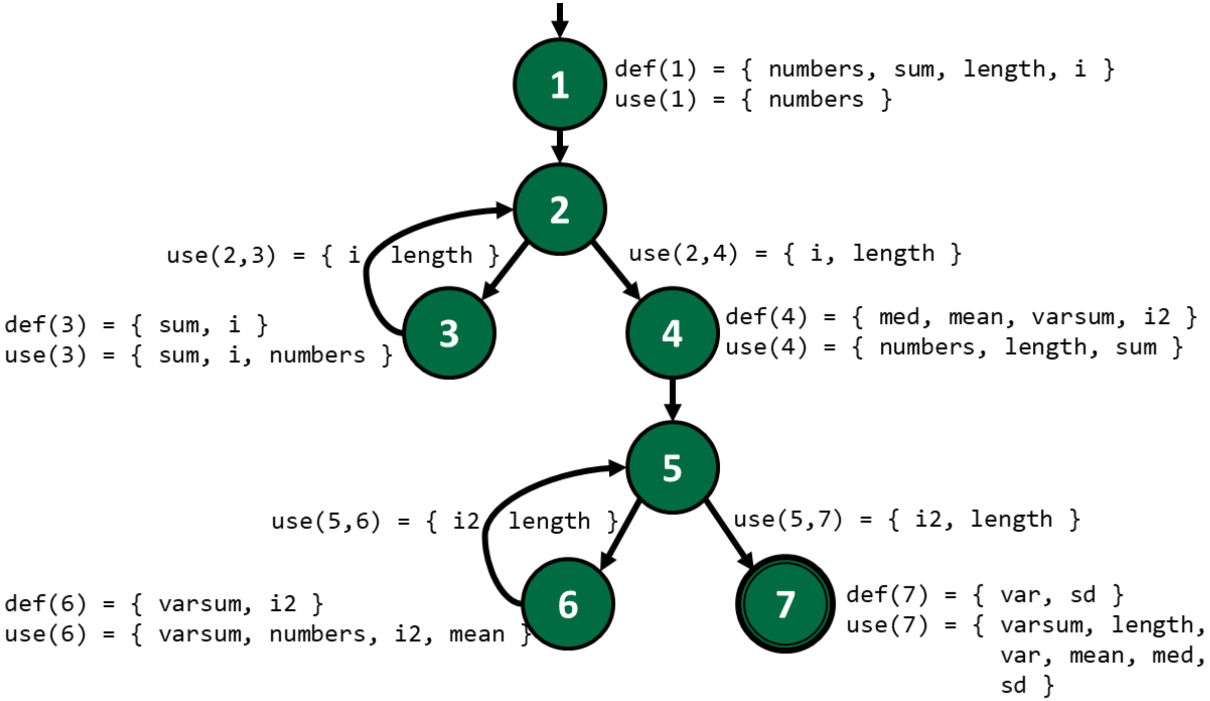
Node	Defs	Uses
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }

Def/Use for Node 7


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

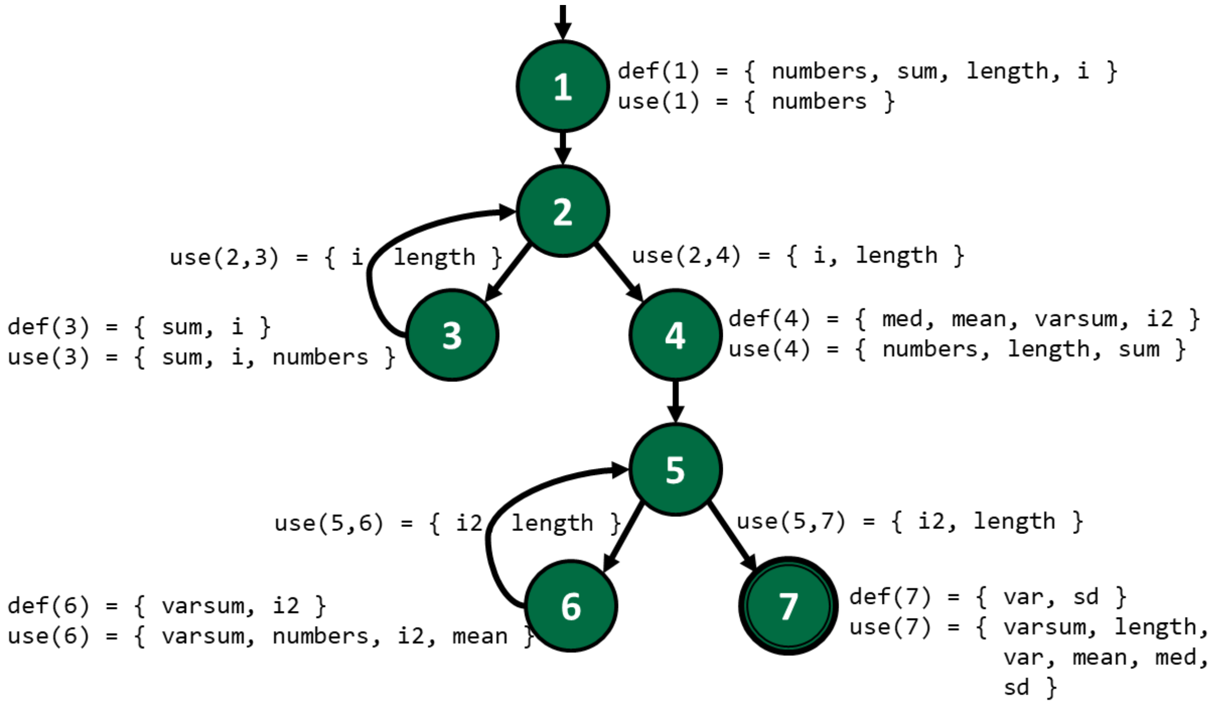
Node	Defs	Uses
7		

Def/Use for Node 7


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Node	Defs	Uses
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Uses for Edge (1,2)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

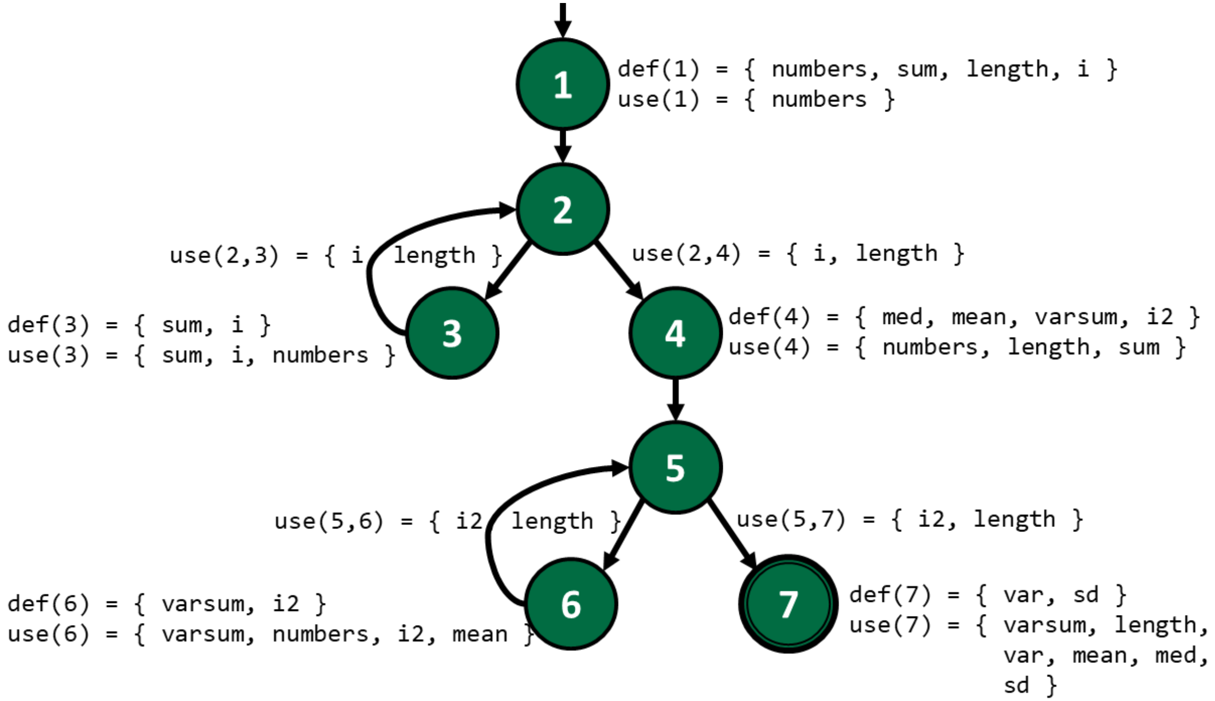
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	

Uses for Edge (1,2)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

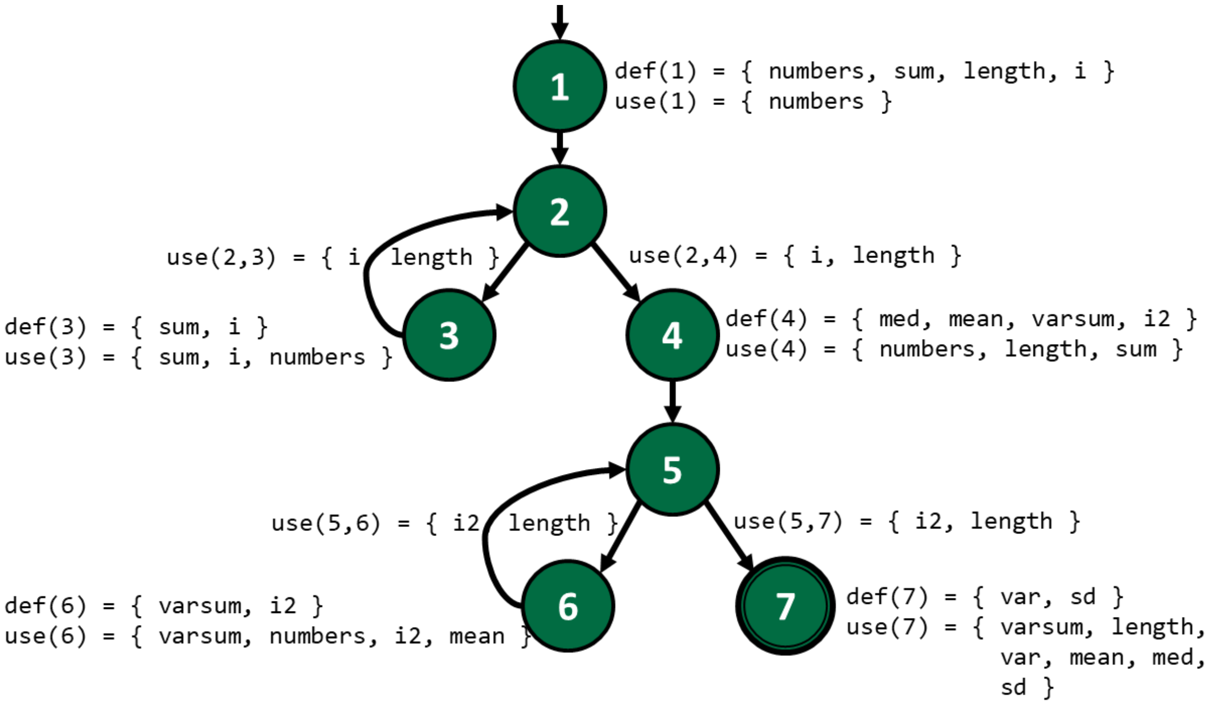
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
var, mean, med,
sd }

Edge	Uses
(1,2)	--

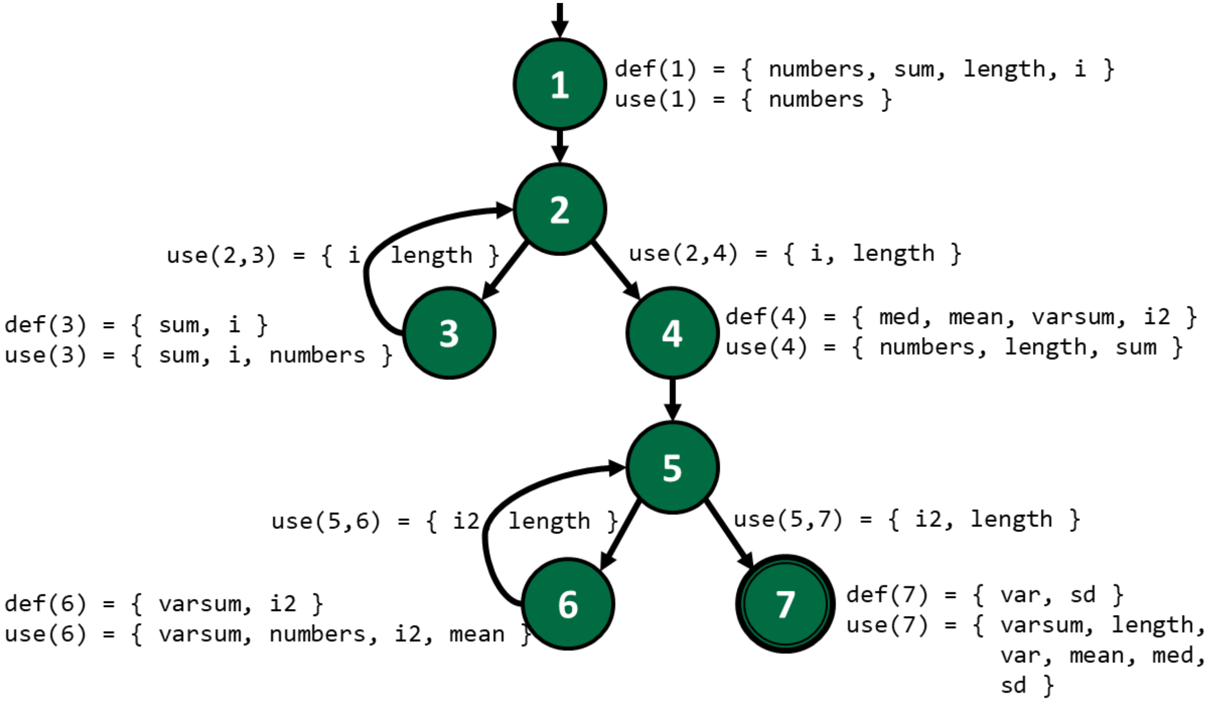
Uses for Edge (2,3)



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(2,3)	

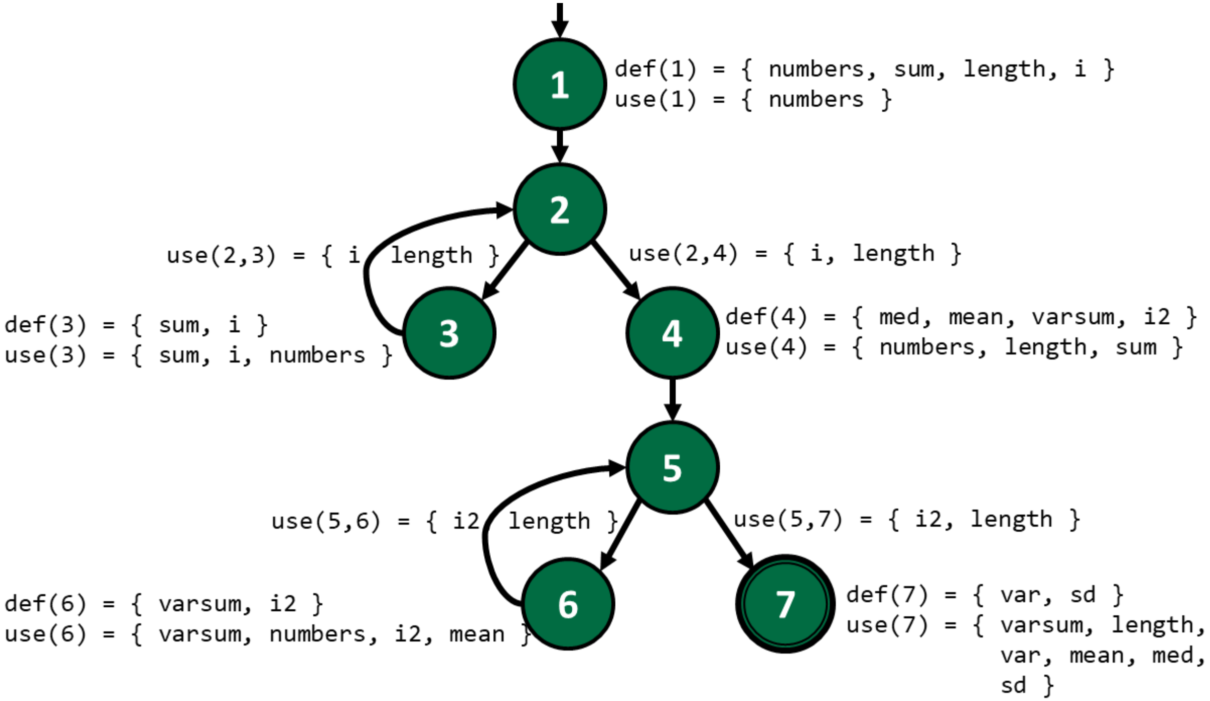
Uses for Edge (2,3)



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(2,3)	{ i, length }

Uses for Edge (2,4)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

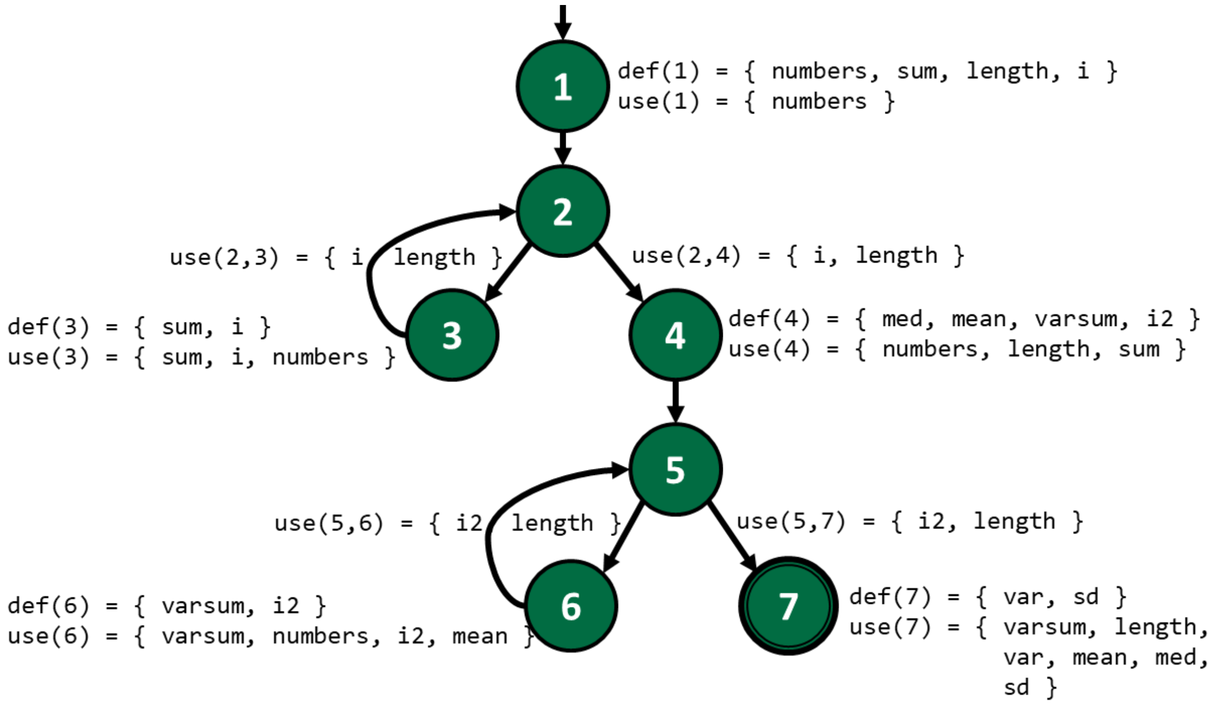
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(2,4)	

Uses for Edge (2,4)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

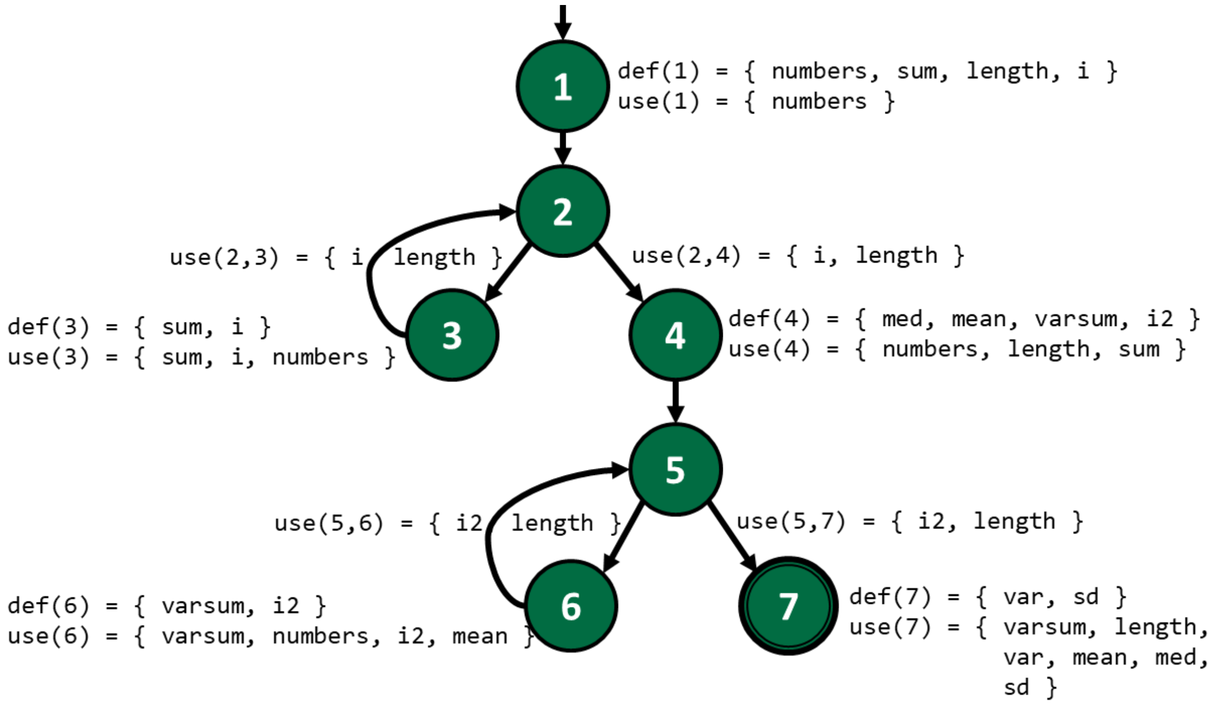
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(2,4)	{ i, length }

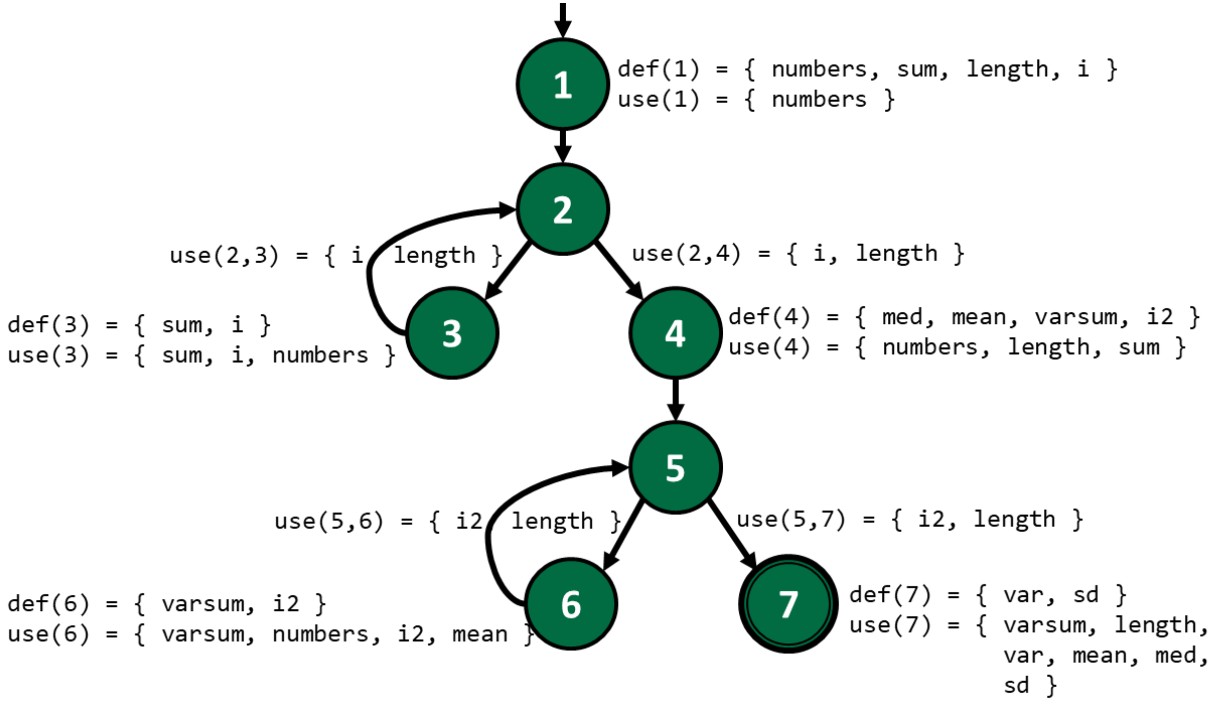
Uses for Edge (3,2)



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Edge	Uses
(3,2)	

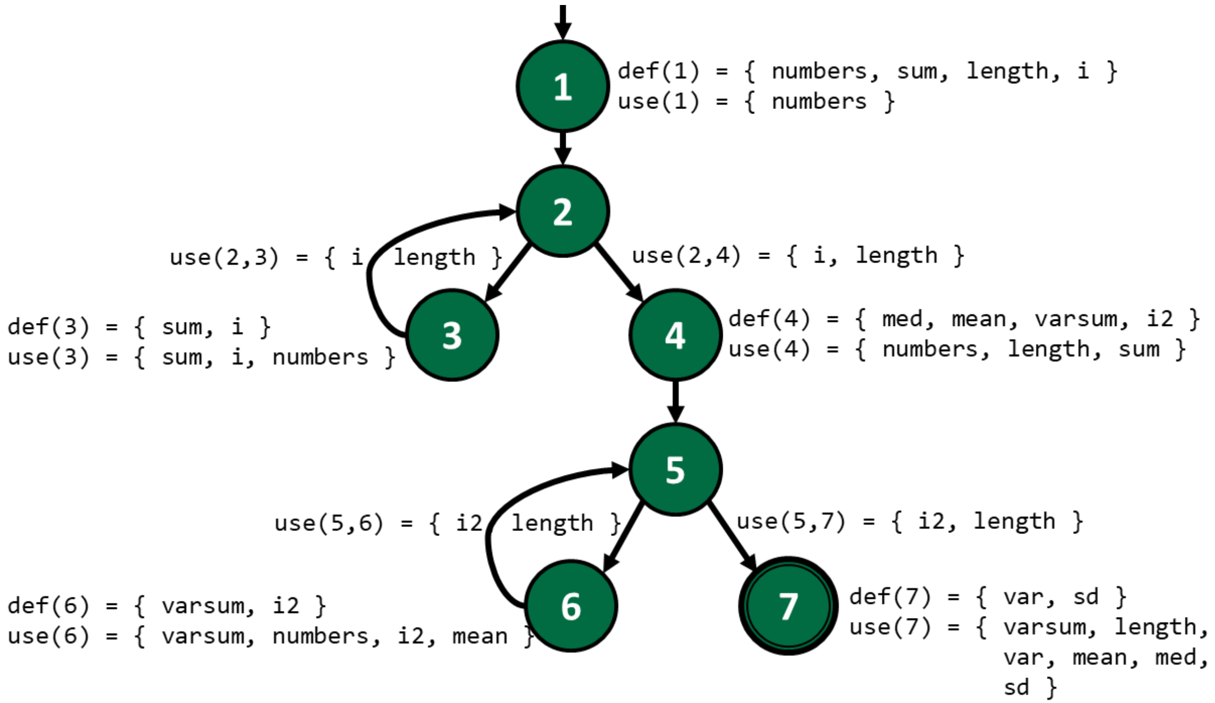
Uses for Edge (3,2)



1
2
3
4
5
6
7
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(3,2)	--

Uses for Edge (4,5)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

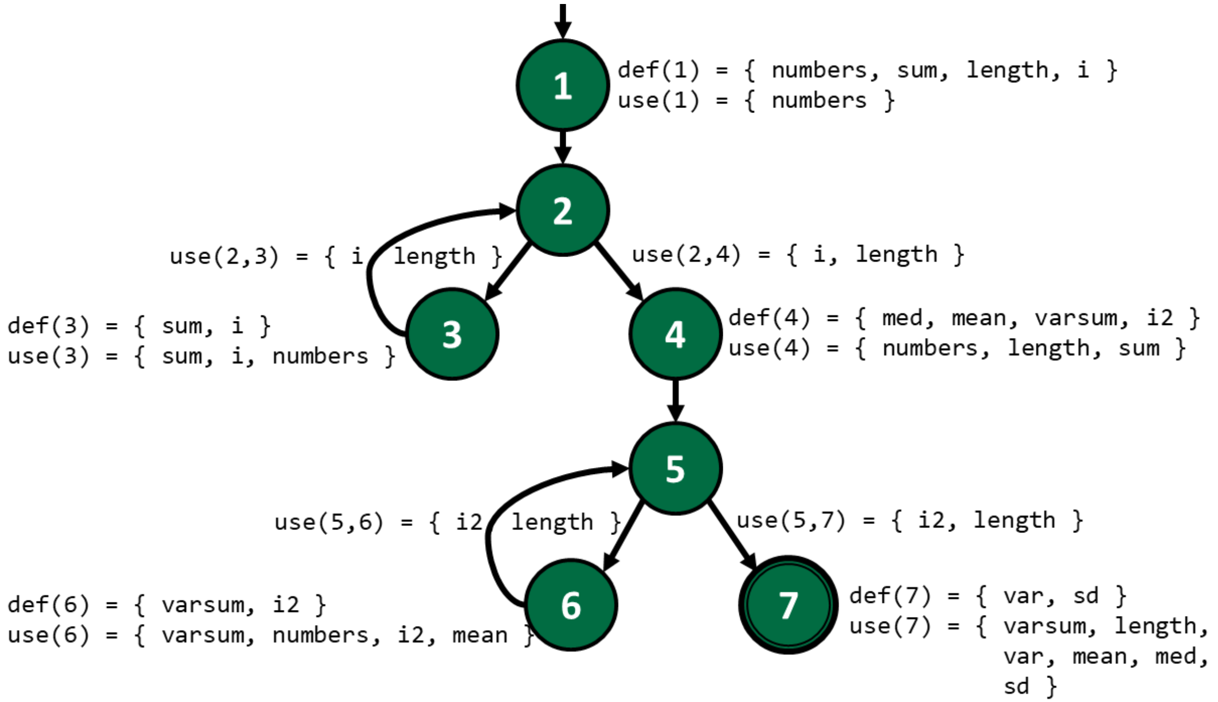
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

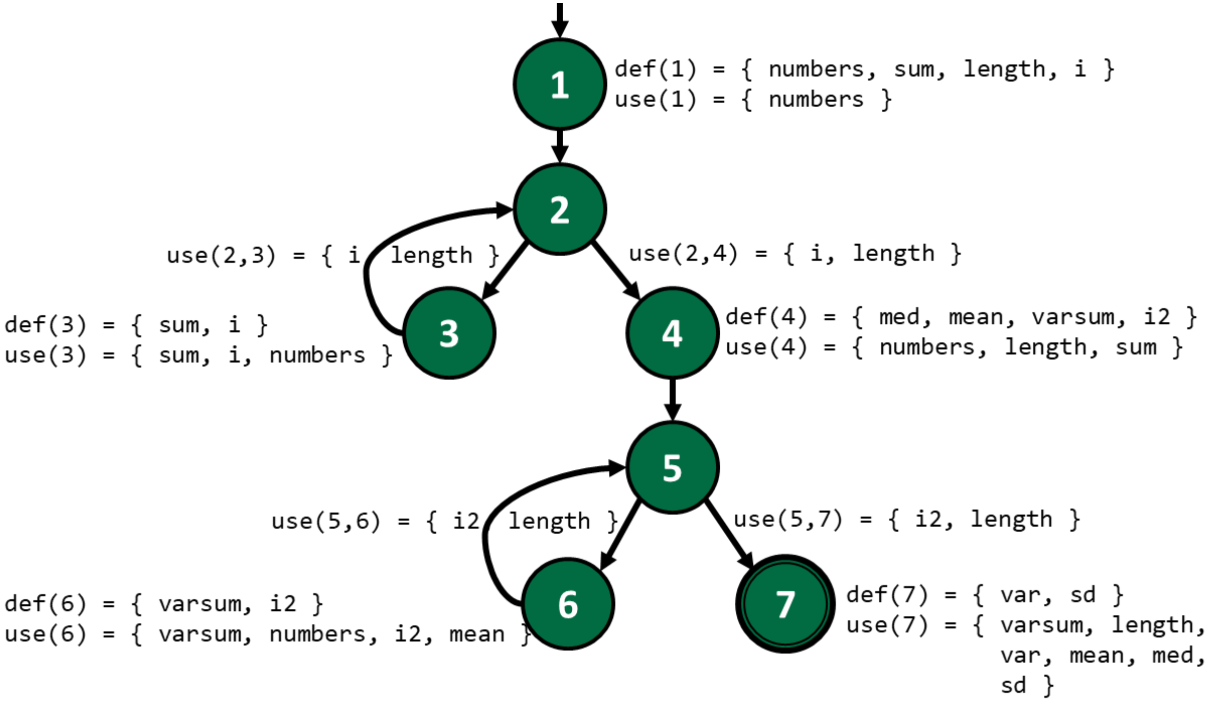
Edge	Uses
(4,5)	

Uses for Edge (4,5)


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(4,5)	--

Uses for Edge (5,6)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

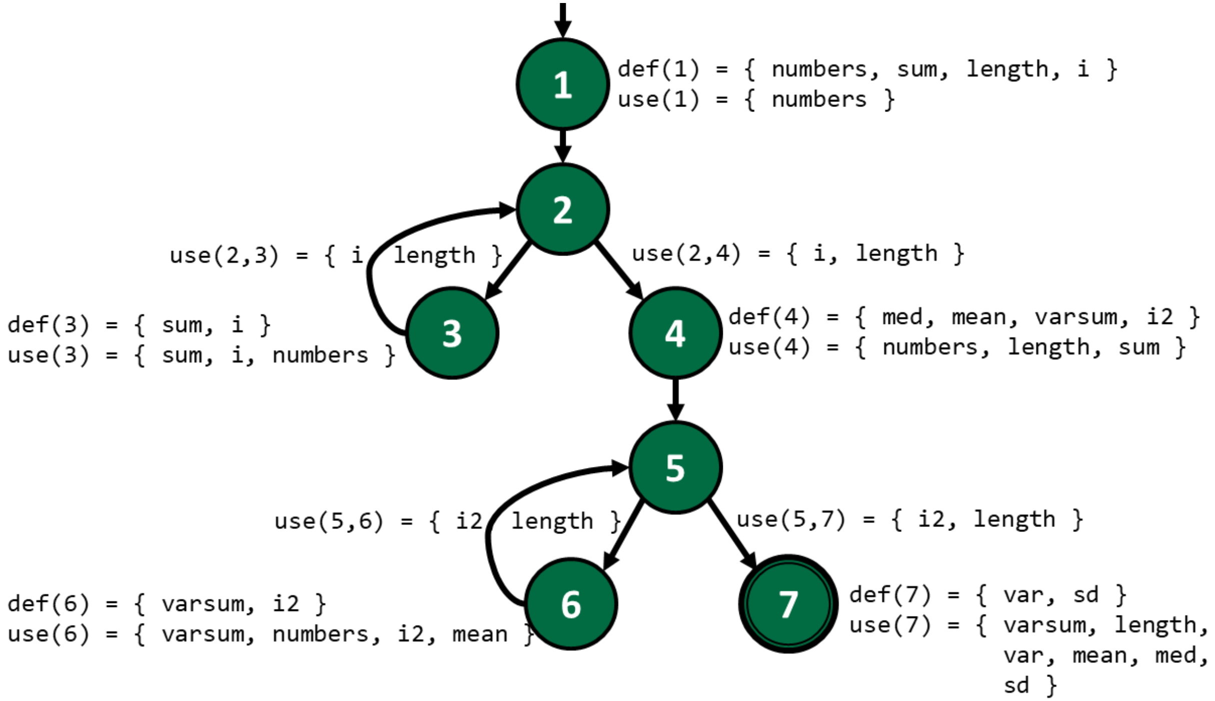
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(5,6)	

Uses for Edge (5,6)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

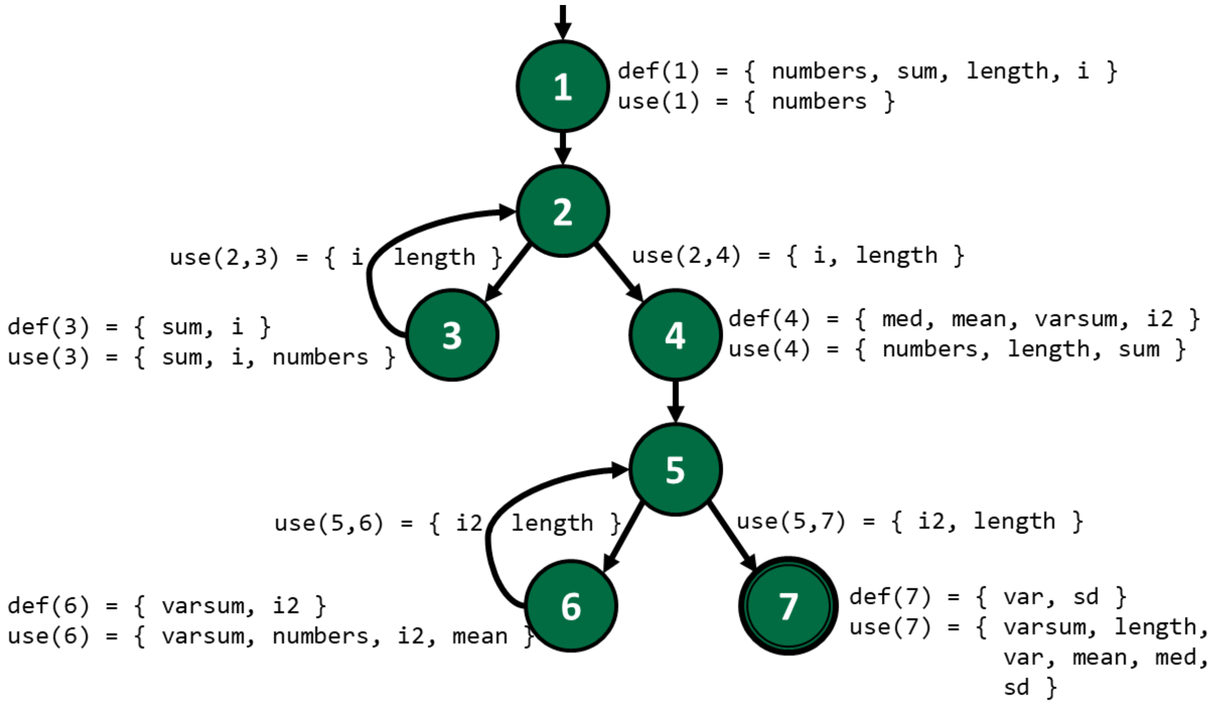
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(5,6)	{ i2, length }

Uses for Edge (5,7)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

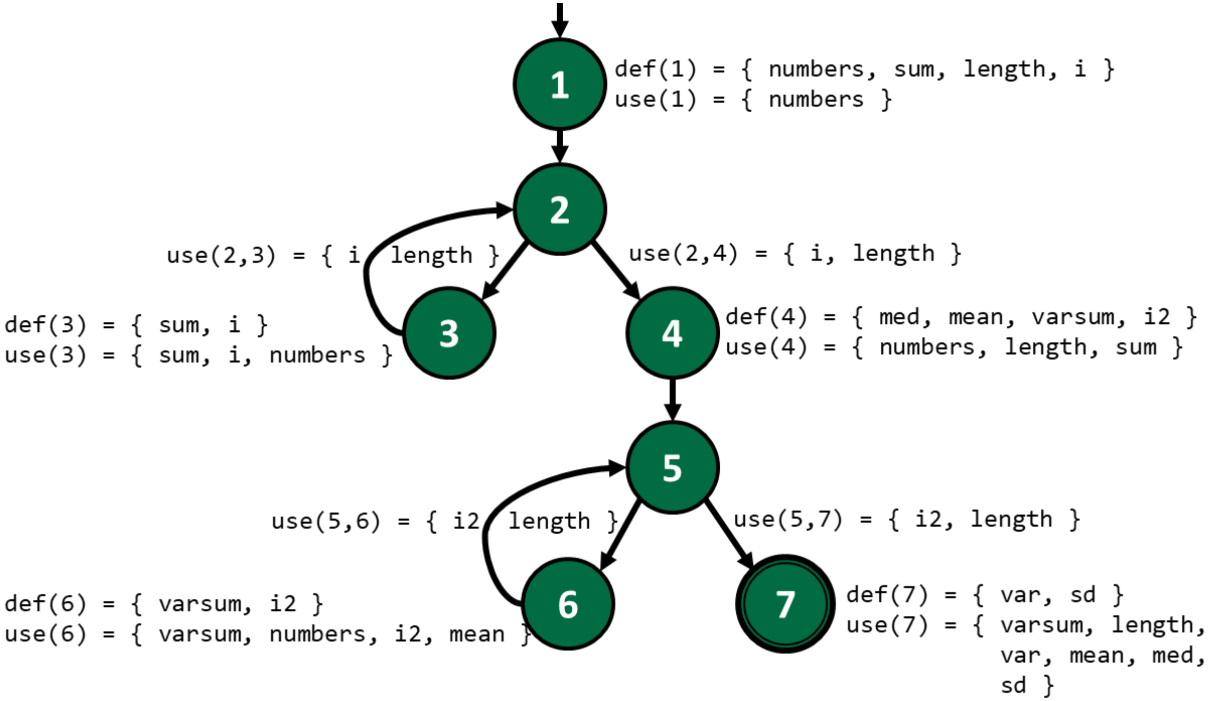
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(5,7)	

Uses for Edge (5,7)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

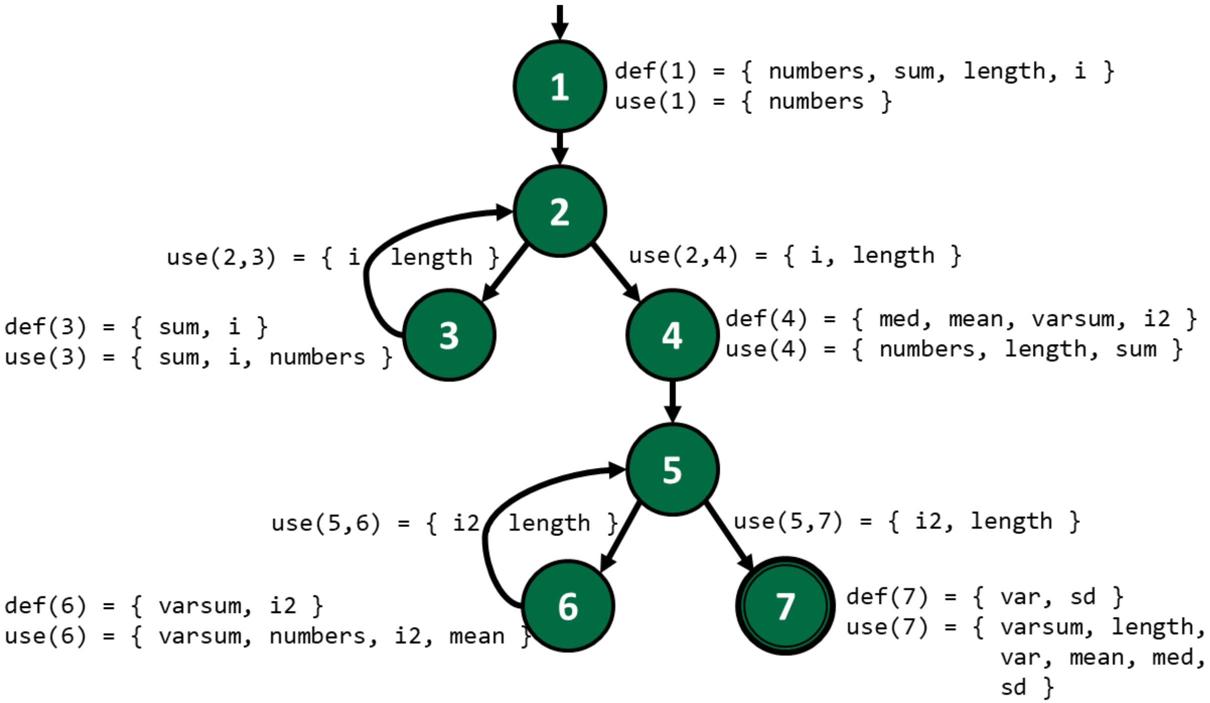
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(5,7)	{ i2, length }

Uses for Edge (6,5)



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

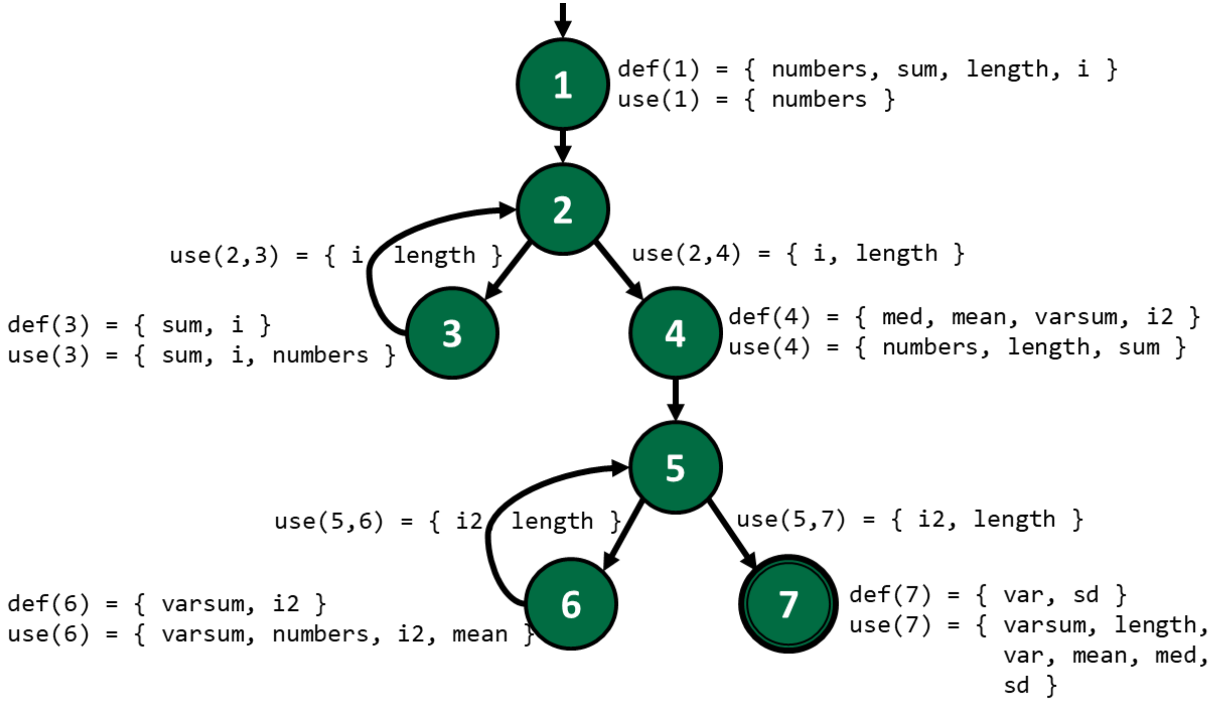
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(6,5)	

Uses for Edge (6,5)


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Edge	Uses
(6,5)	--

Def/Use Tables for computeStats

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

All-Defs Coverage

- The first (and simplest) data flow coverage criterion requires coverage of at least one path from each *def* to at least one *use* of that *def*

DEFINITION

All-Defs Coverage (ADC) – test set *T* satisfies all-defs coverage on graph *G* if and only if *TR* contains at least one DU-path for every def

All-Uses Coverage

- A more complete data flow coverage criterion requires that there is coverage of at least one path from each *def* to every *use* of that *def*

DEFINITION

All-Uses Coverage (AUC) – test set *T* satisfies all-uses coverage on graph *G* if and only if *TR* contains a DU-path for every def to every use

All-DU-Paths Coverage

- An even more complete data flow coverage criterion requires that there is coverage of every path from each *def* to every *use* of that *def*

DEFINITION

All-DU-Paths Coverage (ADUPC) – for each set $S=du(n_i,n_j,v)$, *TR* contains every path *d* in *S*.

DU-Pairs for computeStats

Variable	DU-Pairs
numbers	
length	
med	
var	
sd	
mean	
sum	
varsum	
i	

DU-Pairs for *numbers*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
numbers	

DU-Pairs for *numbers*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
numbers	(1,3), (1,4), (1,6)

DU-Pairs for *length*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
length	

DU-Pairs for *length*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
length	(1,(2,3)), (1,(2,4)), (1,4), (1,(5,6)), (1,(5,7)), (1,7)

DU-Pairs for *med*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
med	

DU-Pairs for *med*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
med	(4,7)

DU-Pairs for *var*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
var	

DU-Pairs for *var*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
var	(7,7)

DU-Pairs for *var*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers,
7	{ var, sd }	

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }

```
var = varsum / (length - 1.0);
...
System.out.println("variance: " + var);
...
```

Def before use in the same node, so (7,7) is not a DU-pair for variable “var”

Variable	
var	~~(7,7)~~

DU-Pairs for *sd*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
sd	

DU-Pairs for *sd*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
sd	(7,7)

DU-Pairs for *sd*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers,
7	{ var, sd }	

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }

```
...
sd   = Math.sqrt(var);
...
System.out.println("std dev:  " + sd);
```

Def before use in the same node, so (7,7) is not a DU-pair for variable "sd"

Variable	
sd	~~(7,7)~~

DU-Pairs for *mean*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
mean	

DU-Pairs for *mean*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
mean	(4,6), (4,7)

DU-Pairs for *sum*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
sum	

DU-Pairs for *sum*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
sum	(1,3), (1,4), (3,3), (3,4)

DU-Pairs for *varsum*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
varsum	

DU-Pairs for *varsum*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
varsum	(4,6), (4,7), (6,6), (6,7)

DU-Pairs for *i*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
i	

DU-Pairs for *i*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
i	(1,(2,3)), (1,(2,4)), (1,3), (3,(2,3)), (3,(2,4)), (3,3)

DU-Pairs for *i*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
i2	

DU-Pairs for *i*

Node	Defs	Uses
1	{ numbers, sum, length, i }	{ numbers }
2	--	--
3	{ sum, i }	{ sum, i, numbers }
4	{ med, mean, varsum, i2 }	{ numbers, length, sum }
5	--	--
6	{ varsum, i2 }	{ varsum, numbers, i2, mean }
7	{ var, sd }	{ varsum, length, var, mean, med, sd }

Edge	Uses
(1,2)	--
(2,3)	{ i, length }
(2,4)	{ i, length }
(3,2)	--
(4,5)	--
(5,6)	{ i2, length }
(5,7)	{ i2, length }
(6,5)	--

Variable	DU-Pairs
i2	(4,(5,6)), (4,(5,7)), (4,6), (6,(5,6)), (6,(5,7)), (6,6)

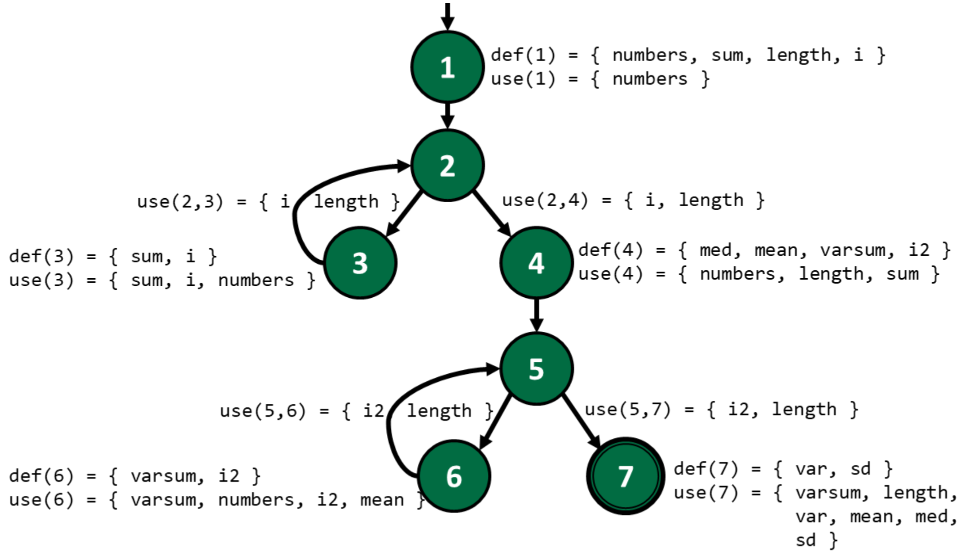
DU-Pairs for computeStats

Variable	DU-Pairs
numbers	(1,3), (1,4), (1,6)
length	(1,(2,3)), (1,(2,4)), (1,4), (1,(5,6)), (1,(5,7)), (1,7)
med	(4,7)
var	~~(7,7)~~
sd	~~(7,7)~~
mean	(4,6), (4,7)
sum	(1,3), (1,4), (3,3), (3,4)
varsum	(4,6), (4,7), (6,6), (6,7)
i	(1,(2,3)), (1,(2,4)), (1,3), (3,(2,3)), (3,(2,4)), (3,3)
i2	(4,(5,6)), (4,(5,7)), (4,6), (6,(5,6)), (6,(5,7)), (6,6)

DU-Paths for computeStats

Variable	DU-Pairs	DU-Paths
numbers	(1,3), (1,4), (1,6)	
length	(1,(2,3)), (1,(2,4)), (1,4), (1,(5,6)), (1,(5,7)), (1,7)	
med	(4,7)	
mean	(4,6), (4,7)	
sum	(1,3), (1,4), (3,3), (3,4)	
varsum	(4,6), (4,7), (6,6), (6,7)	
i	(1,(2,3)), (1,(2,4)), (1,3), (3,(2,3)), (3,(2,4)),(3,3)	
i2	(4,(5,6)), (4,(5,7)), (4,6), (6,(5,6)), (6,(5,7)), (6,6)	

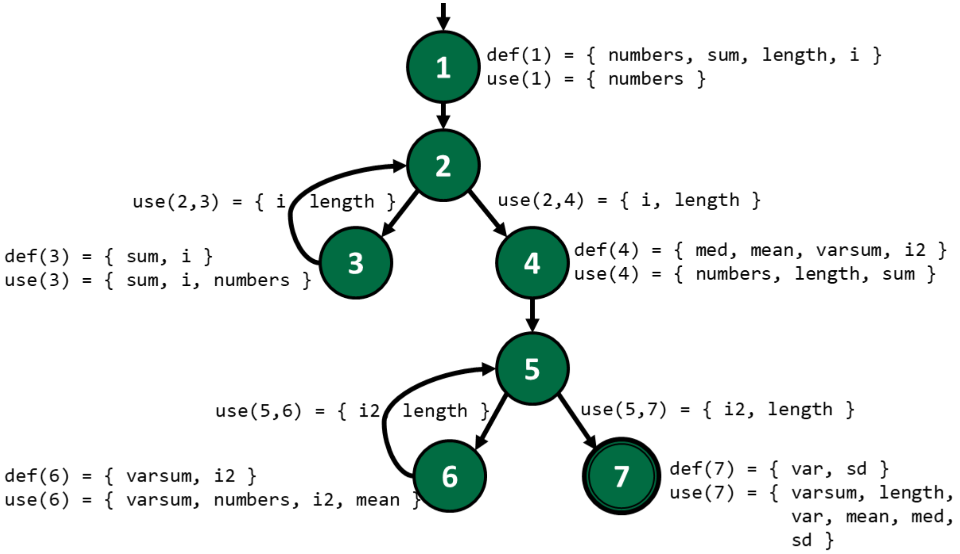
DU-Paths for *numbers*



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	

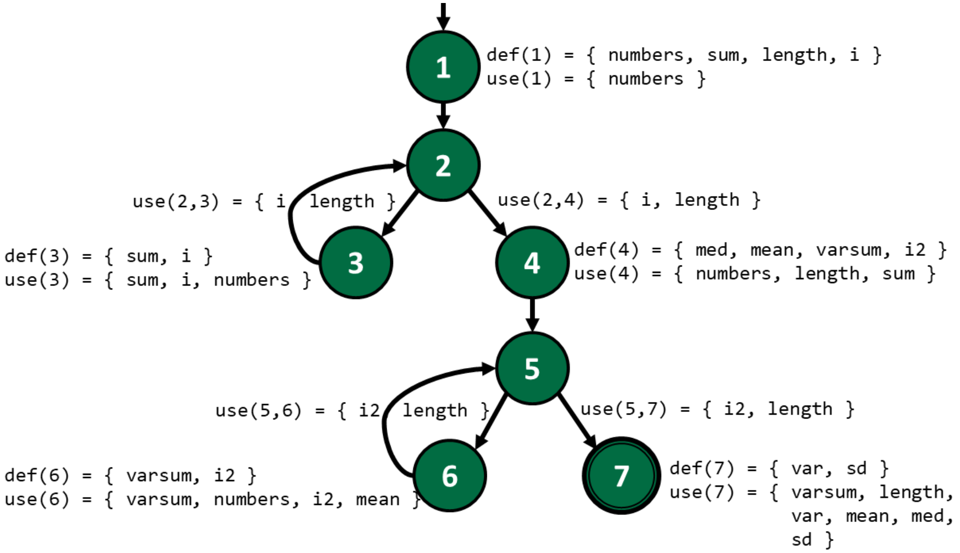
DU-Paths for *numbers*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	[1,2,3]

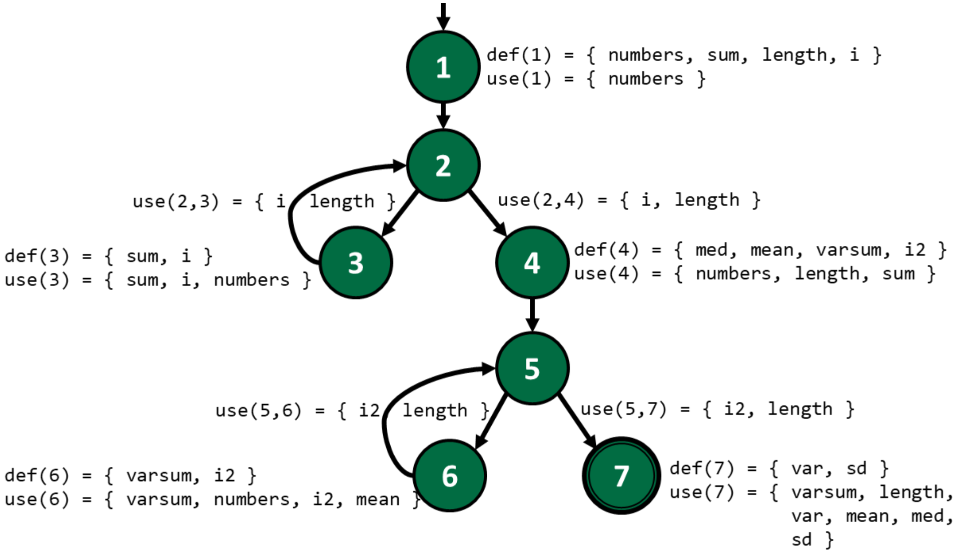
DU-Paths for *numbers*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	[1,2,3] [1,2,4]

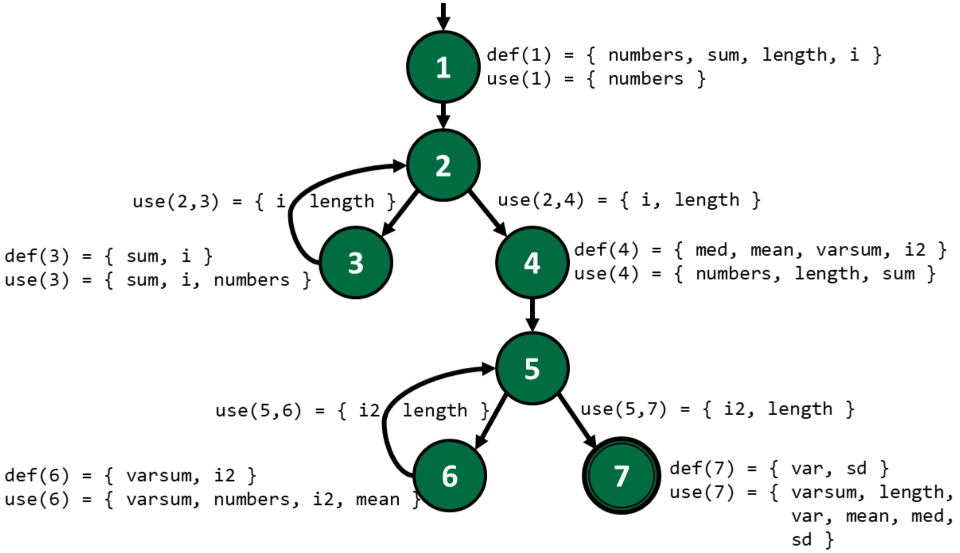
DU-Paths for *numbers*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

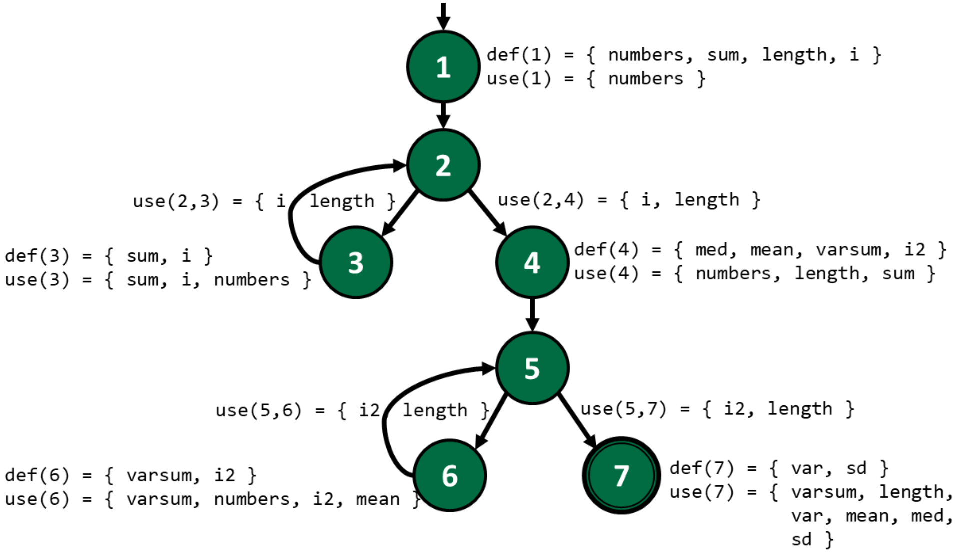
Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	[1,2,3] [1,2,4] [1,2,4,5,6]

DU-Paths for *length*


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	

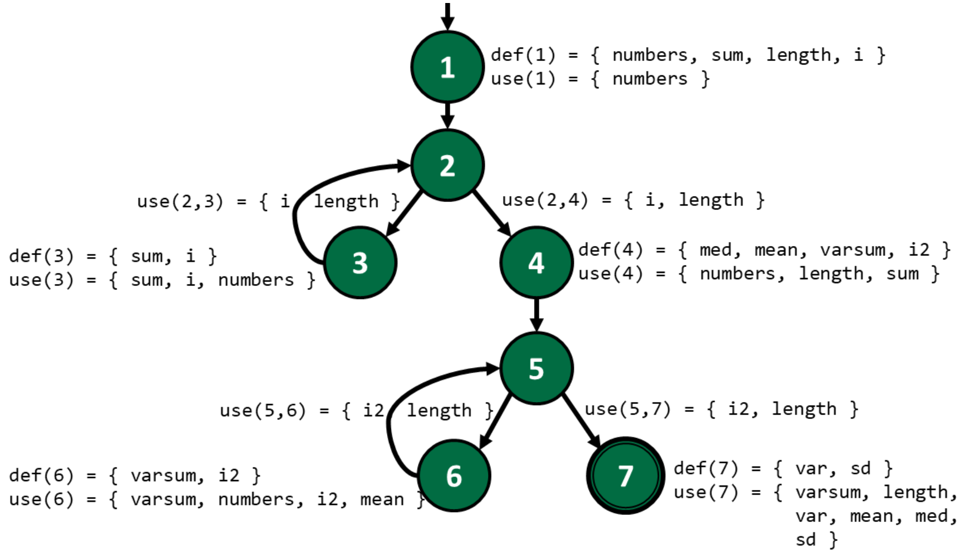
DU-Paths for *length*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3]

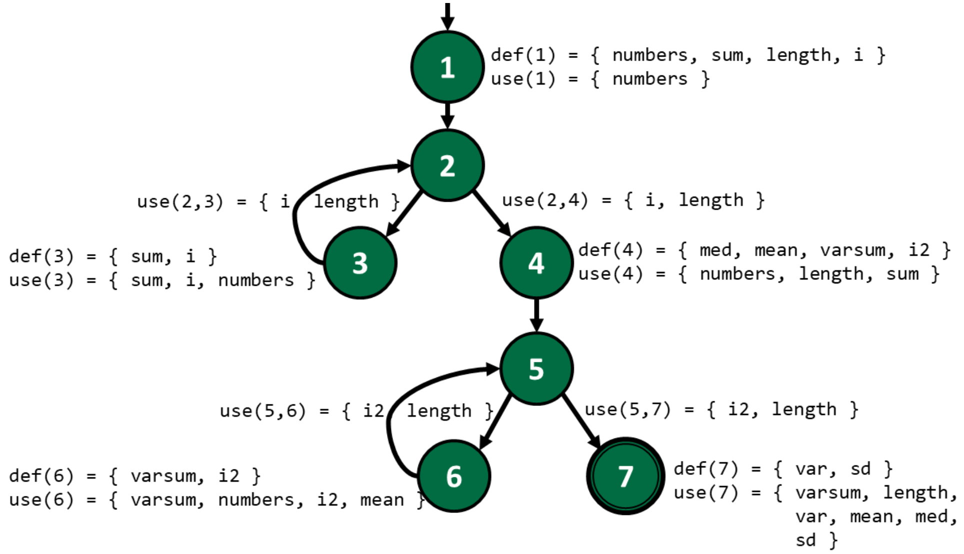
DU-Paths for *length*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3] [1,2,4]

DU-Paths for *length*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

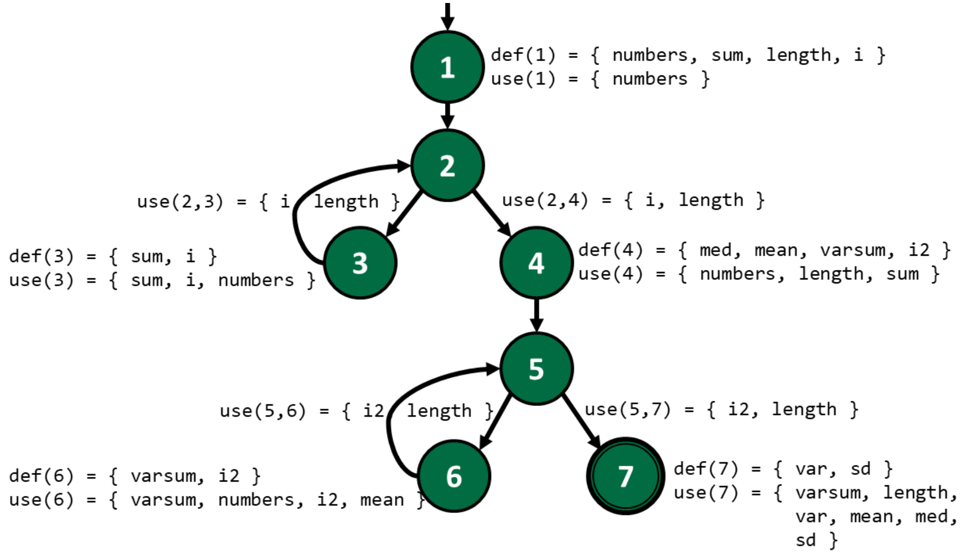
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3] [1,2,4] [1,2,4]

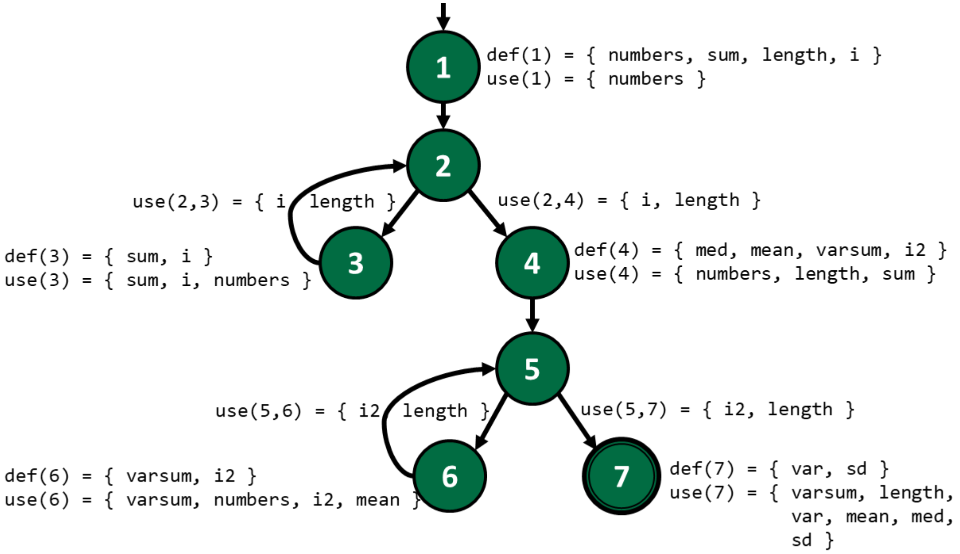
DU-Paths for *length*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }
use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3] [1,2,4] [1,2,4] [1,2,4,5,6]

DU-Paths for *length*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

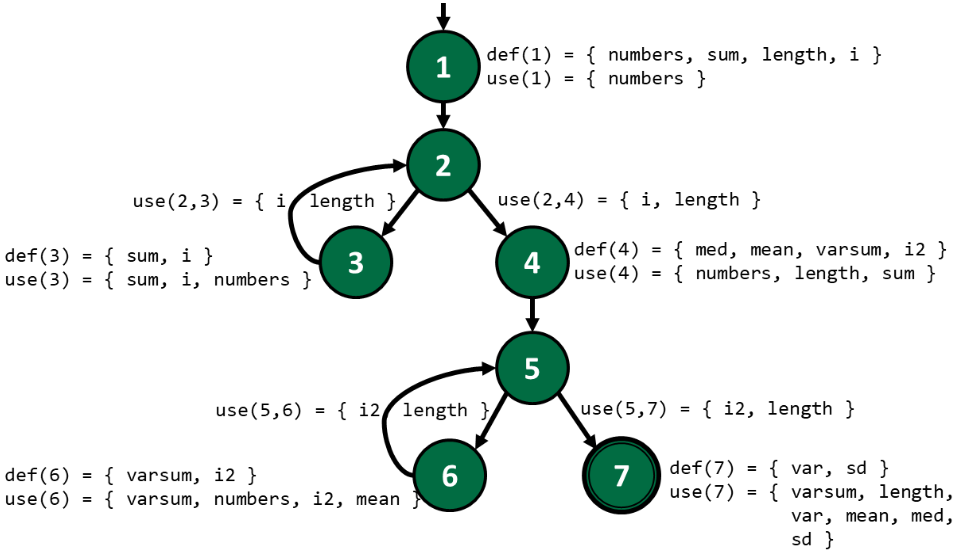
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3] [1,2,4] [1,2,4] [1,2,4,5,6] [1,2,4,5,7]

DU-Paths for *length*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

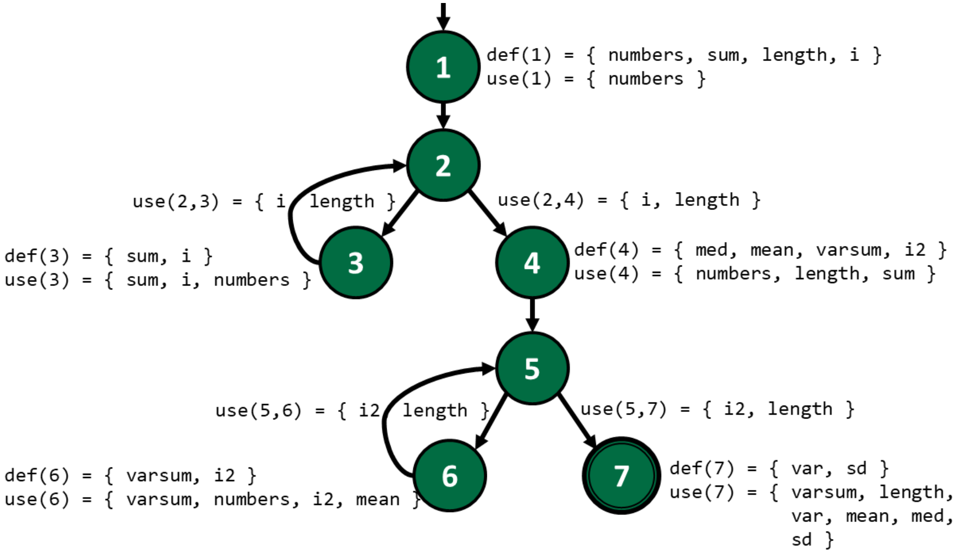
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3] [1,2,4] [1,2,4] [1,2,4,5,6] [1,2,4,5,7] [1,2,4,5,7]

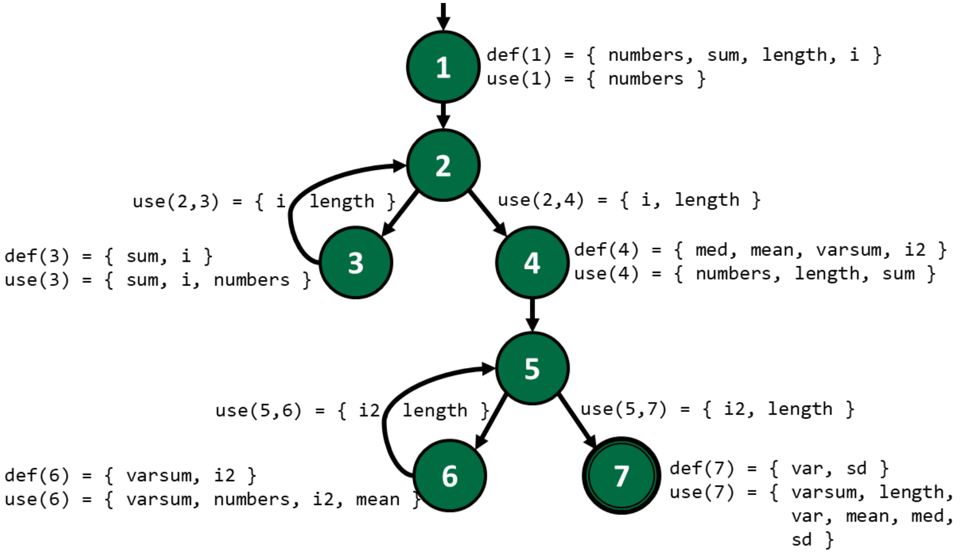
DU-Paths for *med*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
med	(4,7)	

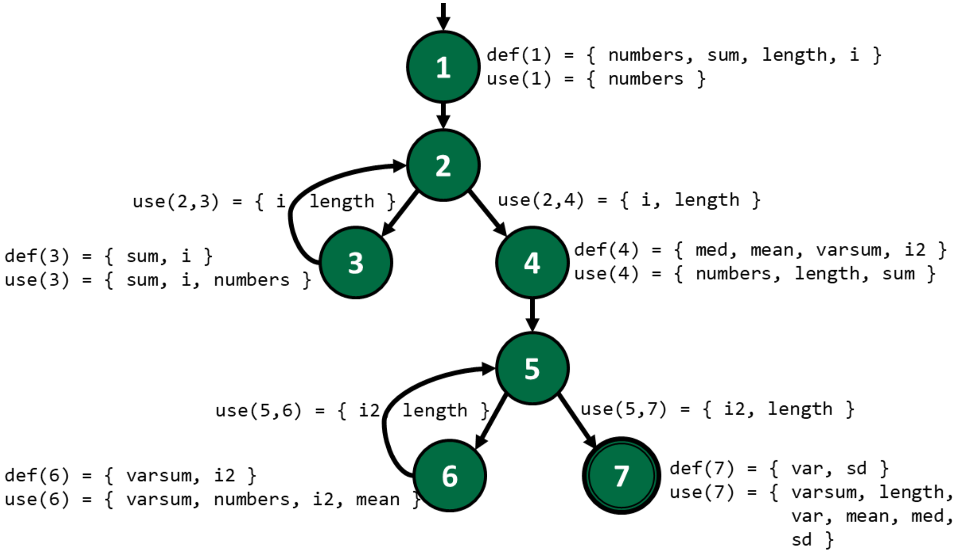
DU-Paths for *med*



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

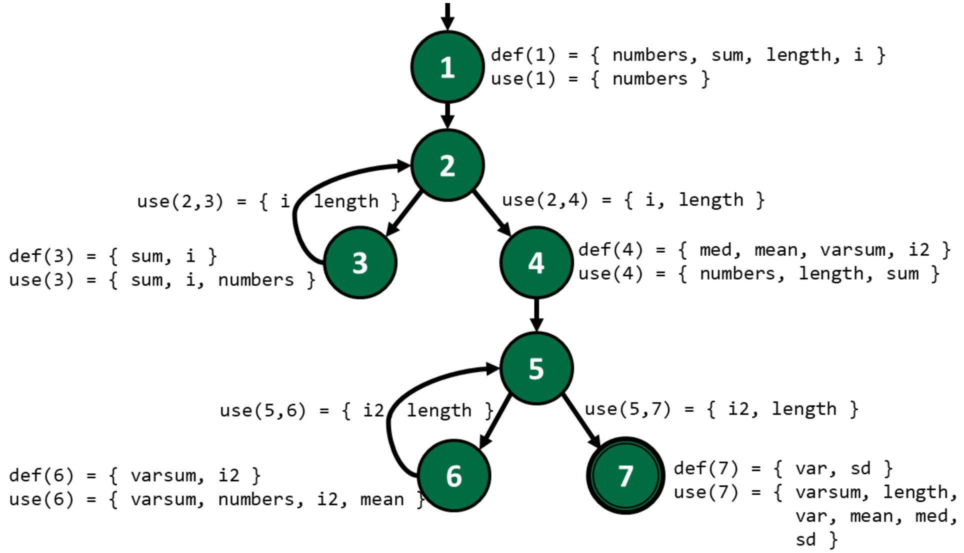
Variable	DU-Pairs	DU-Paths
med	(4,7)	[4,5,7]

DU-Paths for *mean*


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
mean	(4,6) (4,7)	

DU-Paths for *mean*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

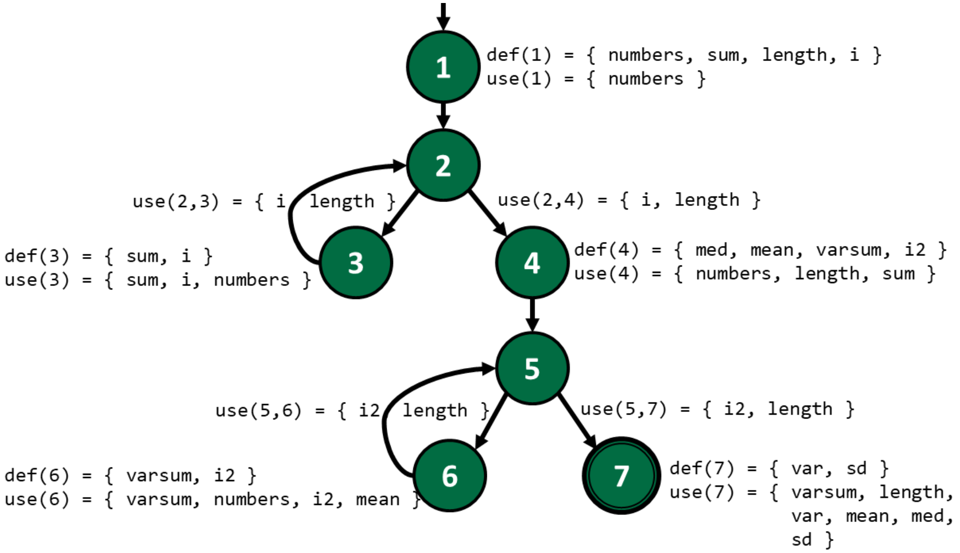
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
mean	(4,6) (4,7)	[4,5,6]

DU-Paths for *mean*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

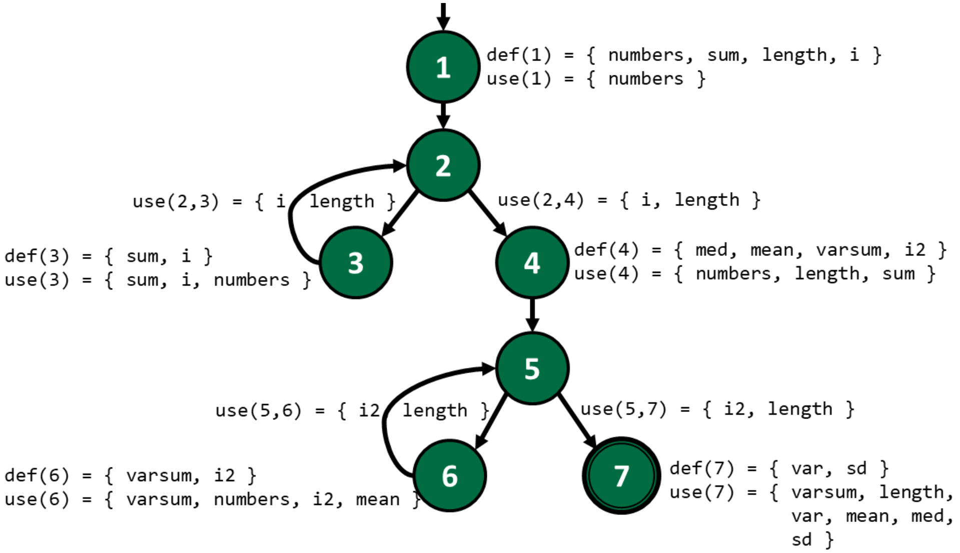
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

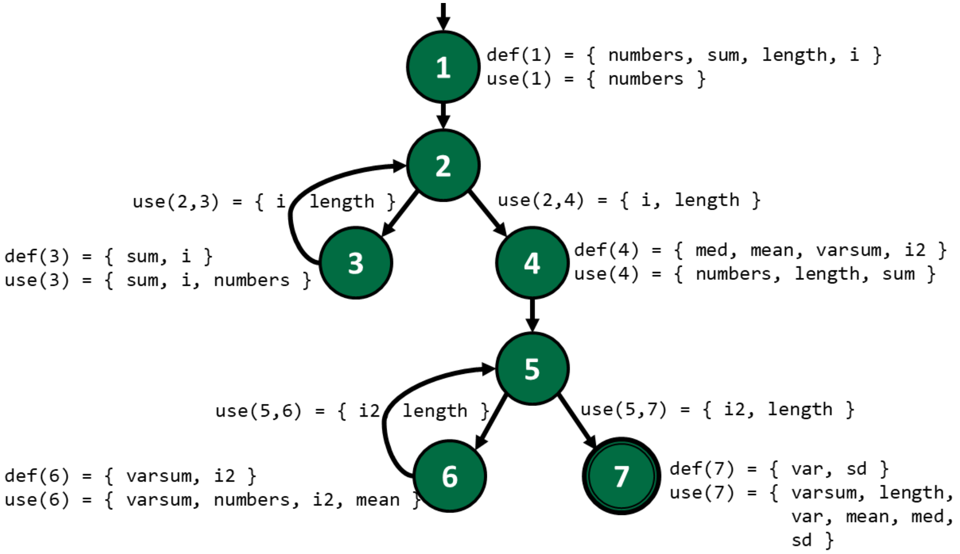
Variable	DU-Pairs	DU-Paths
mean	(4,6) (4,7)	[4,5,6] [4,5,7]

DU-Paths for *sum*


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	

DU-Paths for *sum*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

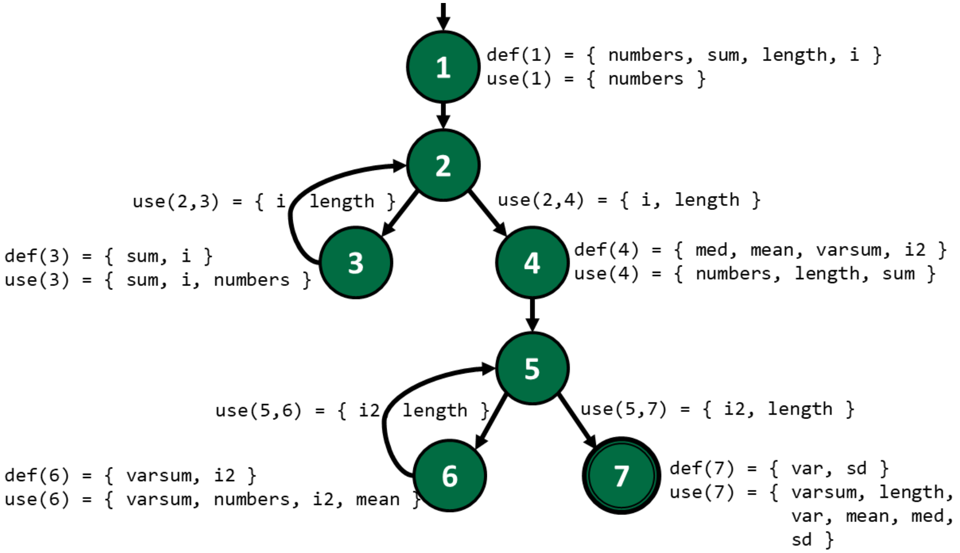
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
var, mean, med,
sd }

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	[1,2,3]

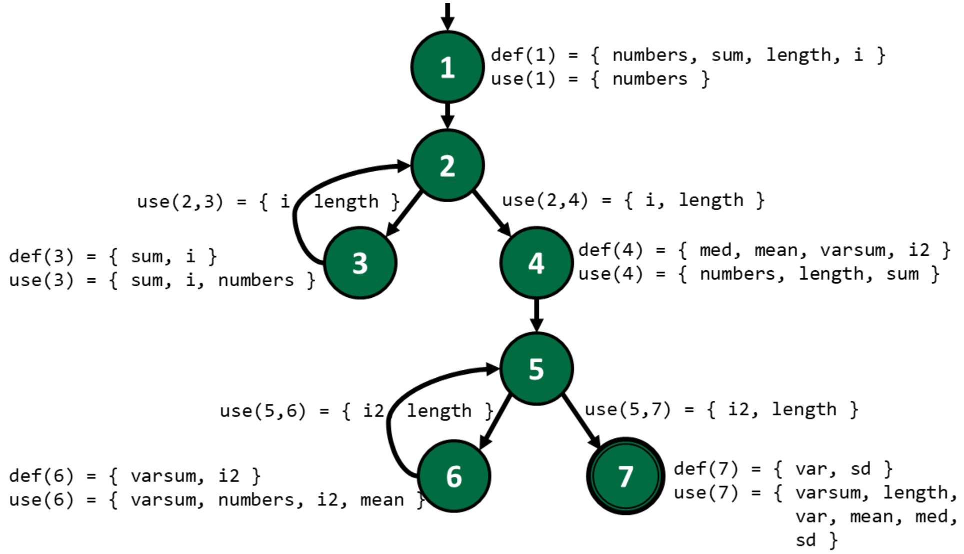
DU-Paths for *sum*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	[1,2,3] [1,2,4]

DU-Paths for *sum*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

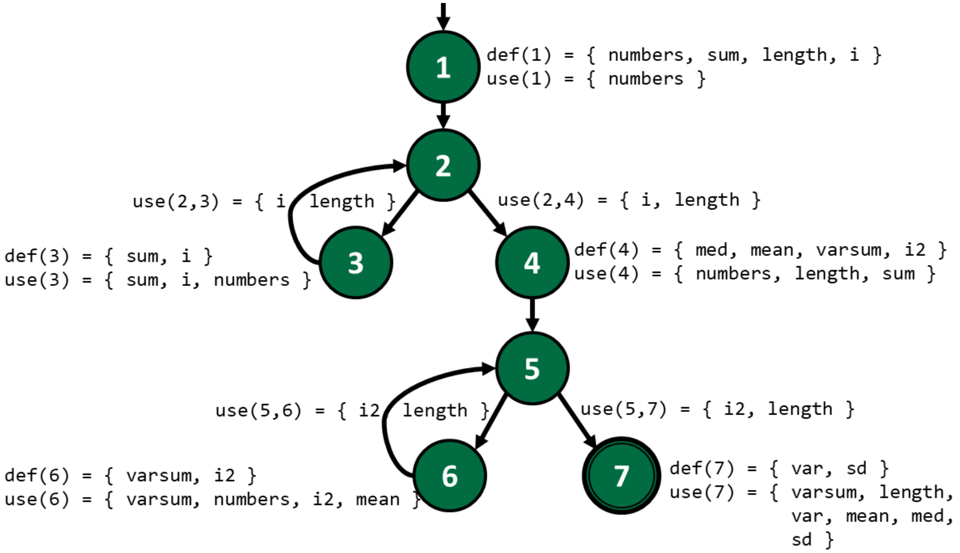
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	[1,2,3] [1,2,4] [3,2,3]

DU-Paths for *sum*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

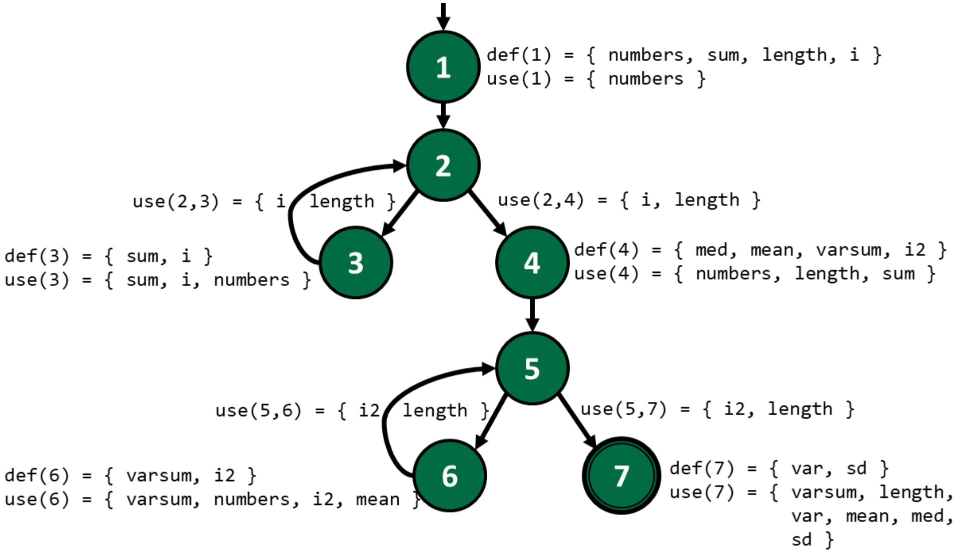
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	[1,2,3] [1,2,4] [3,2,3] [3,2,4]

DU-Paths for *varsum*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

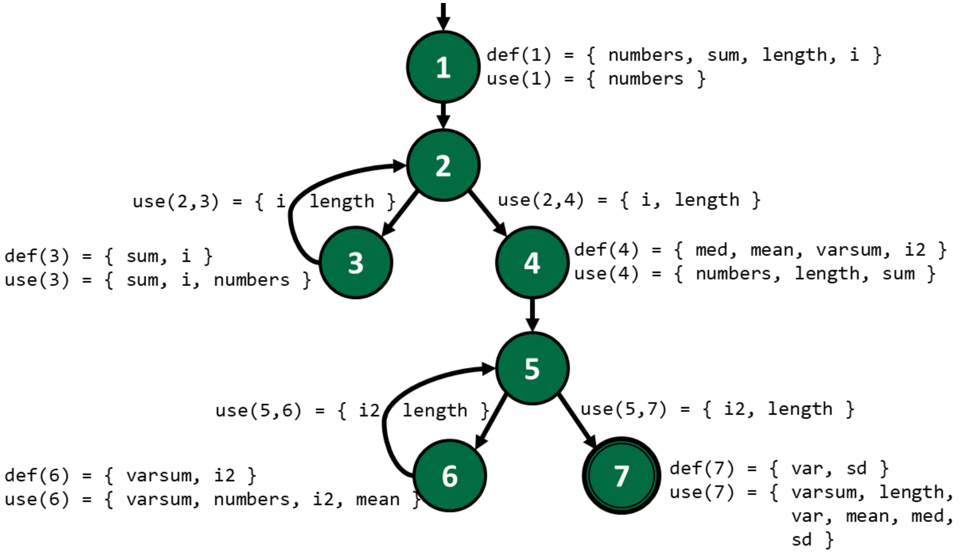
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
varsum	(4,6) (4,7) (6,6) (6,7)	

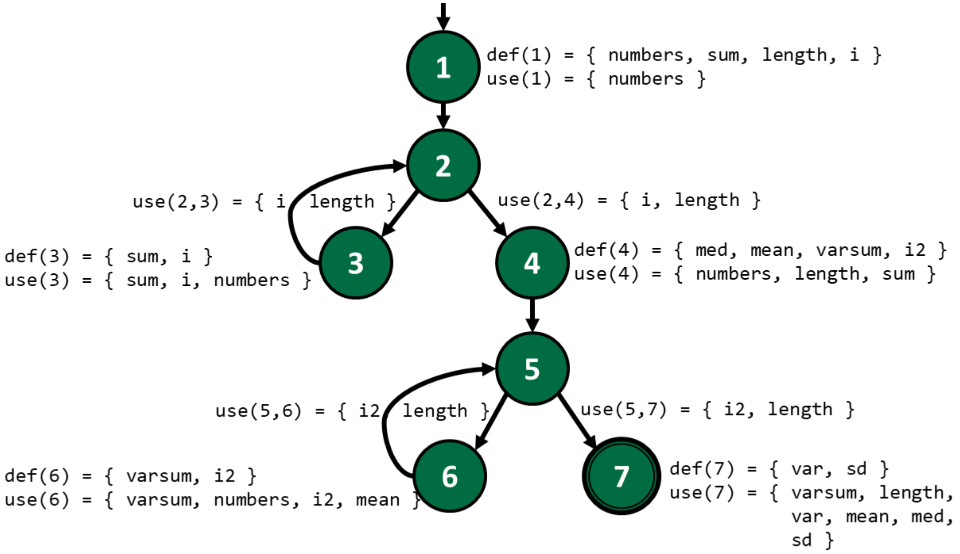
DU-Paths for *varsum*



1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
varsum	(4,6) (4,7) (6,6) (6,7)	[4,5,6]

DU-Paths for *varsum*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

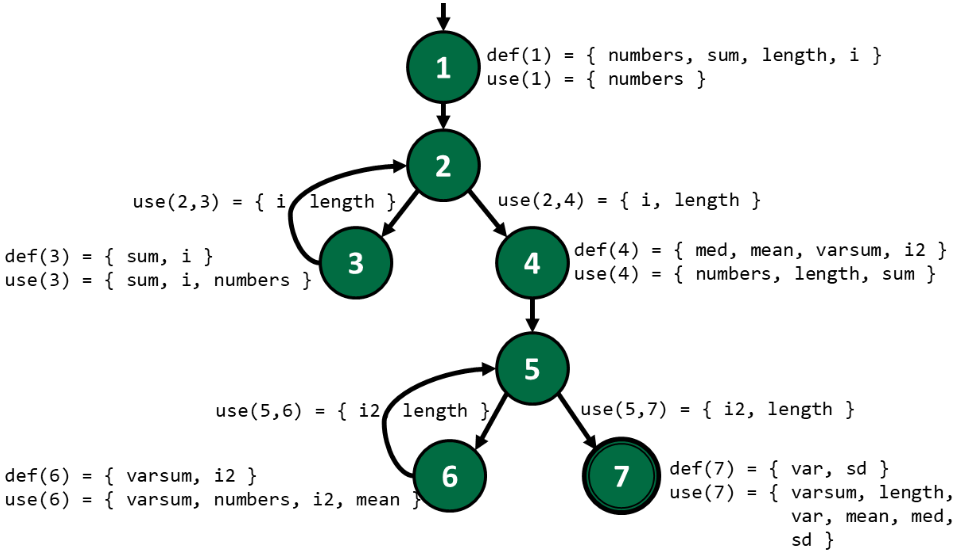
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
var, mean, med,
sd }

Variable	DU-Pairs	DU-Paths
varsum	(4,6) (4,7) (6,6) (6,7)	[4,5,6] [4,5,7]

DU-Paths for *varsum*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

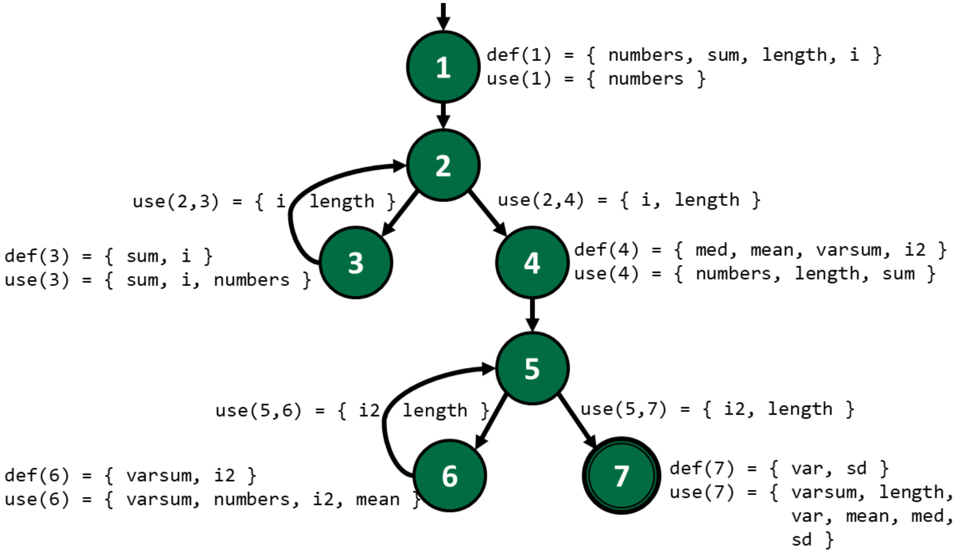
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
var, mean, med,
sd }

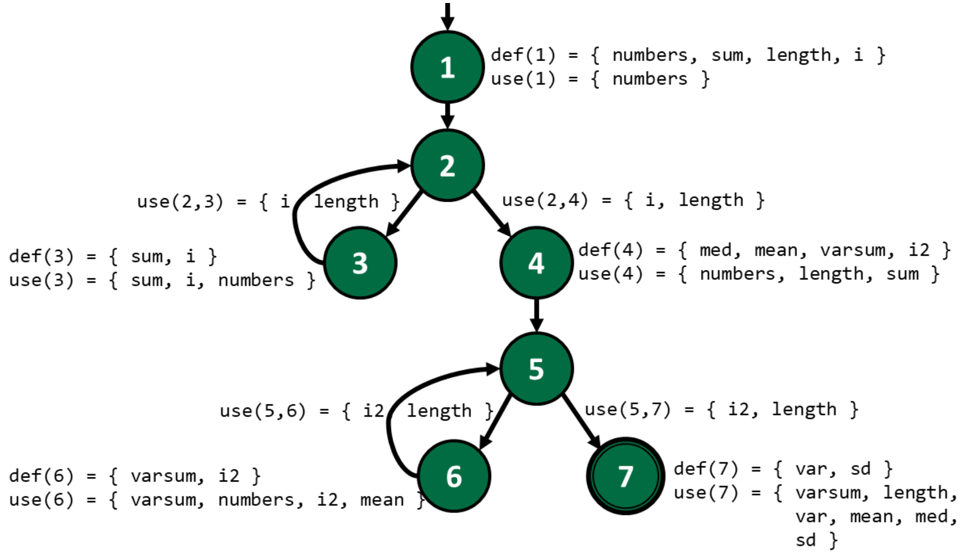
Variable	DU-Pairs	DU-Paths
varsum	(4,6) (4,7) (6,6) (6,7)	[4,5,6] [4,5,7] [6,5,6]

DU-Paths for *varsum*


1
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
2
use(2,3) = { i, length }
use(2,4) = { i, length }
3
def(3) = { sum, i }
use(3) = { sum, i, numbers }
4
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
5
use(5,6) = { i2, length }
use(5,7) = { i2, length }
6
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
7
def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
varsum	(4,6)	[4,5,6]
	(4,7)	[4,5,7]
	(6,6)	[6,5,6]
	(6,7)	[6,5,7]

DU-Paths for *i*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

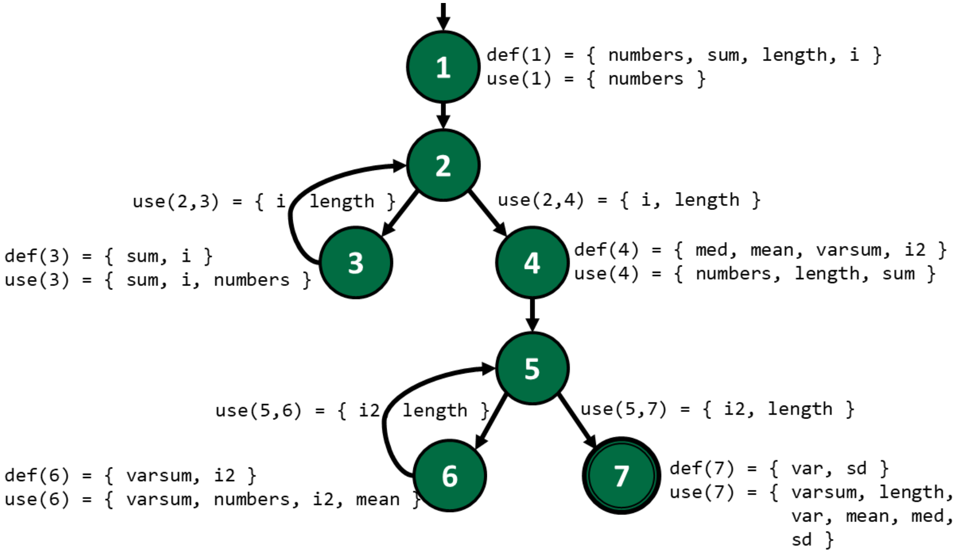
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	

DU-Paths for *i*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

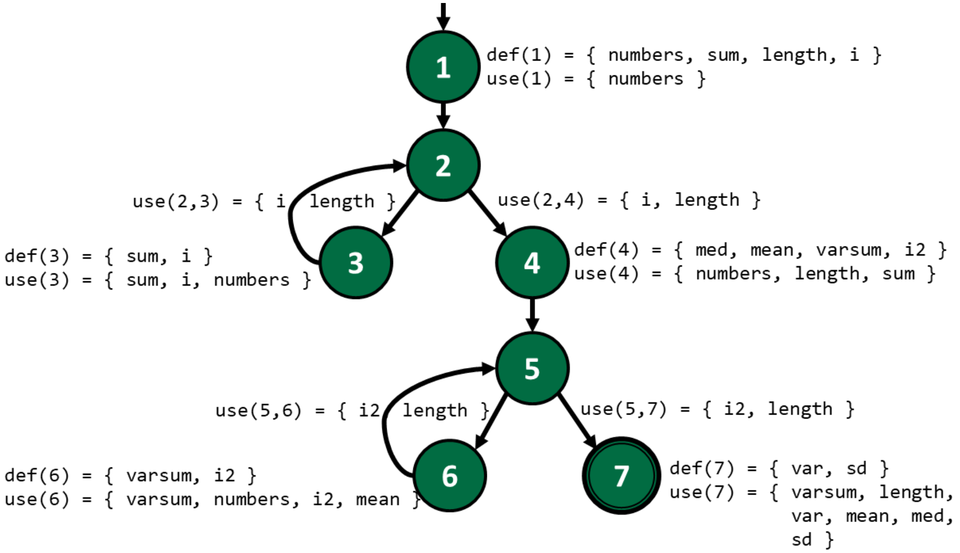
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	[1,2,3]

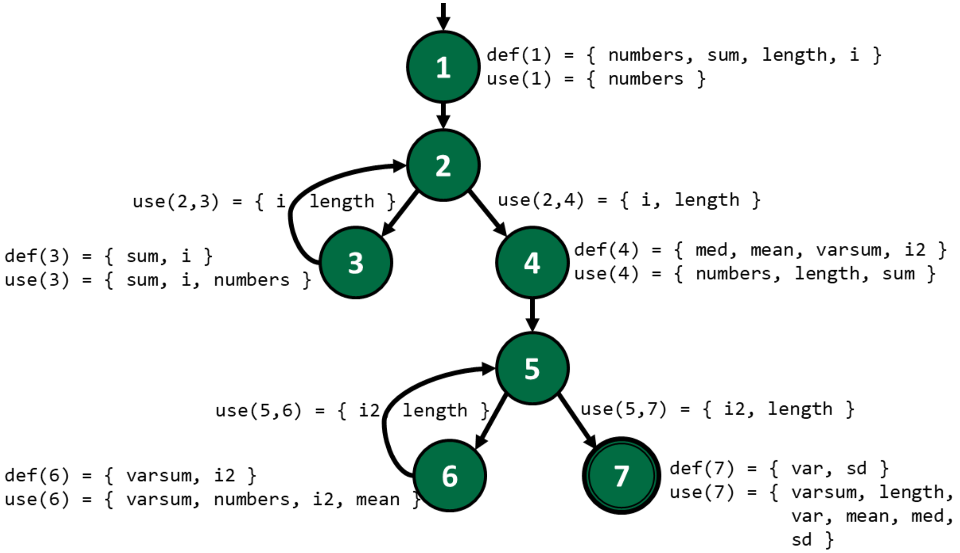
DU-Paths for *i*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	[1,2,3] [1,2,4]

DU-Paths for *i*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

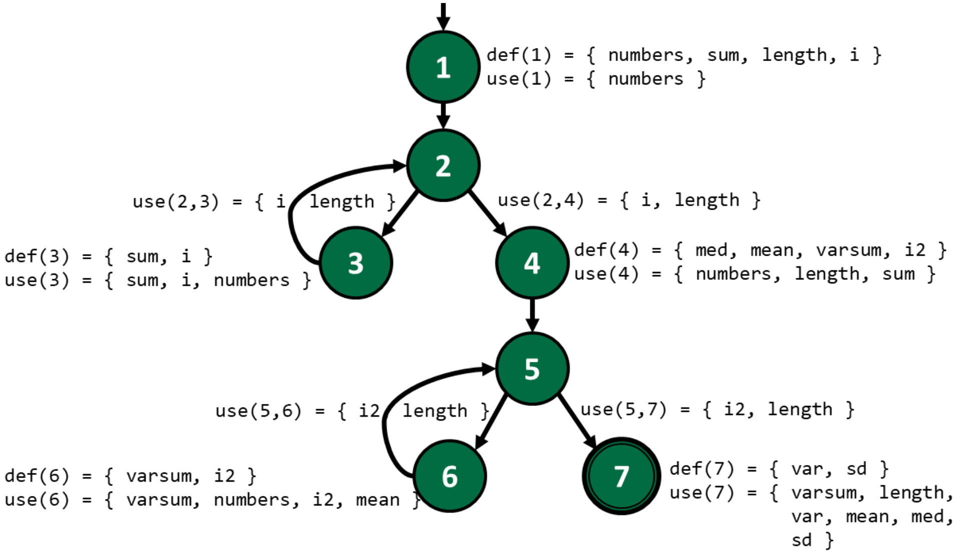
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length,
var, mean, med,
sd }

Variable	DU-Pairs	DU-Paths
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	[1,2,3] [1,2,4] [1,2,3]

DU-Paths for *i*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

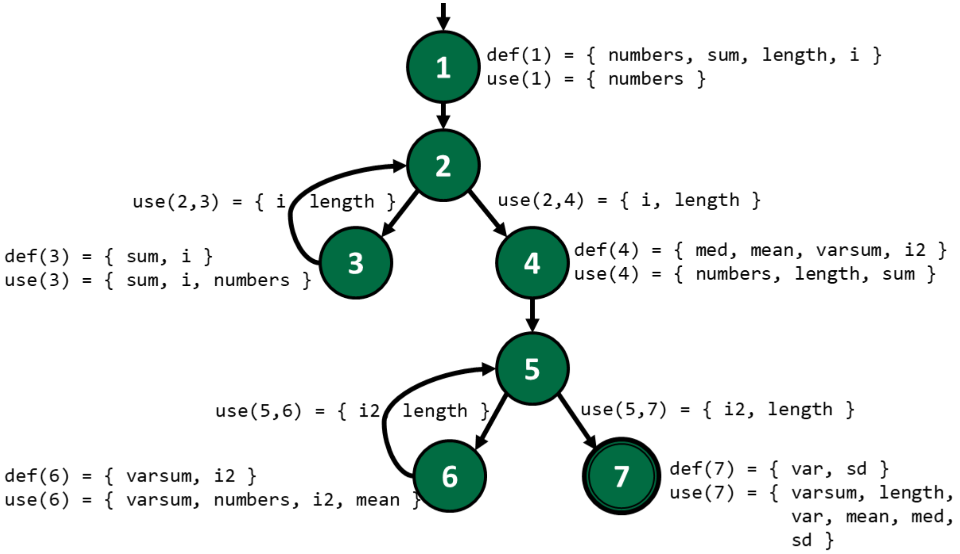
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	[1,2,3] [1,2,4] [1,2,3] [3,2,3]

DU-Paths for *i*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

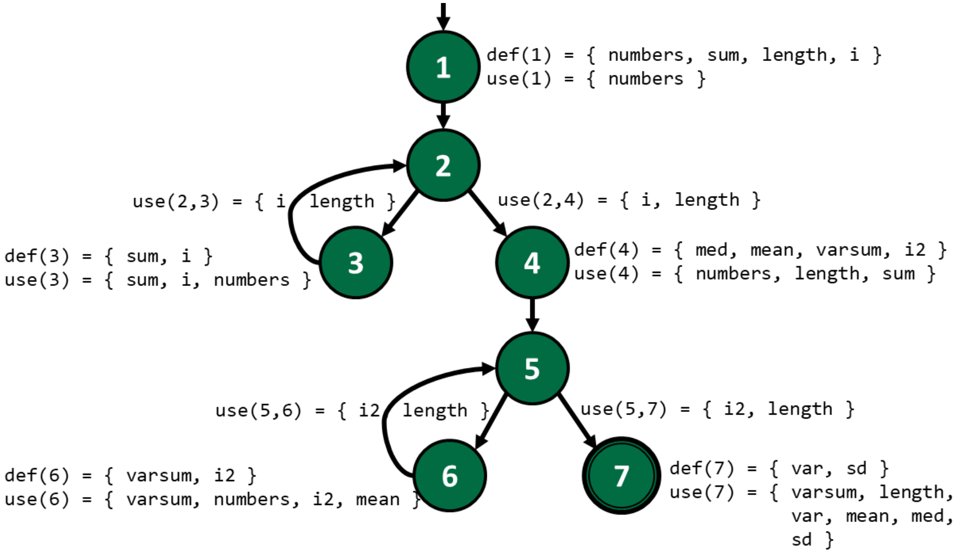
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i	(1,(2,3))	[1,2,3]
	(1,(2,4))	[1,2,4]
	(1,3)	[1,2,3]
	(3,(2,3))	[3,2,3]
	(3,(2,4))	[3,2,4]
	(3,3)	

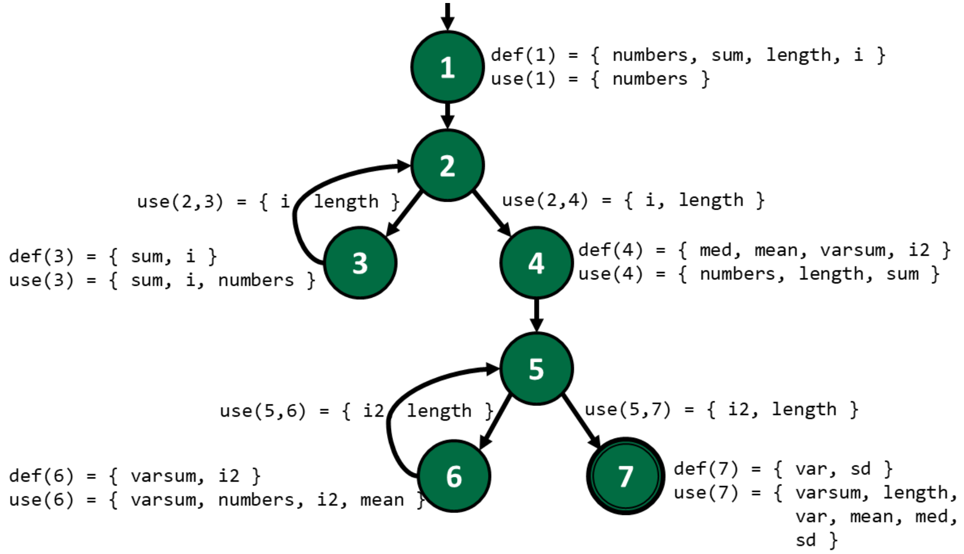
DU-Paths for *i*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	[1,2,3] [1,2,4] [1,2,3] [3,2,3] [3,2,4] [3,2,3]

DU-Paths for *i2*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

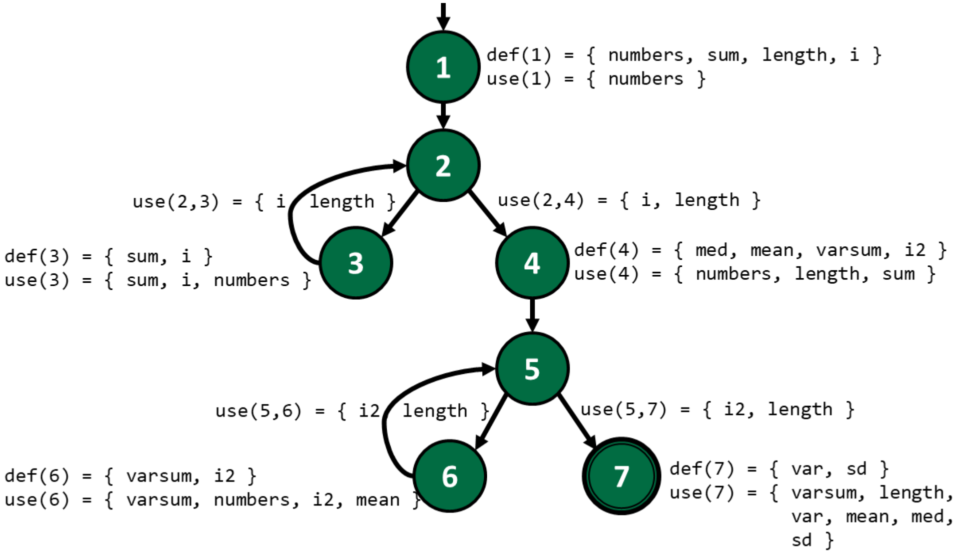
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	

DU-Paths for *i2*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

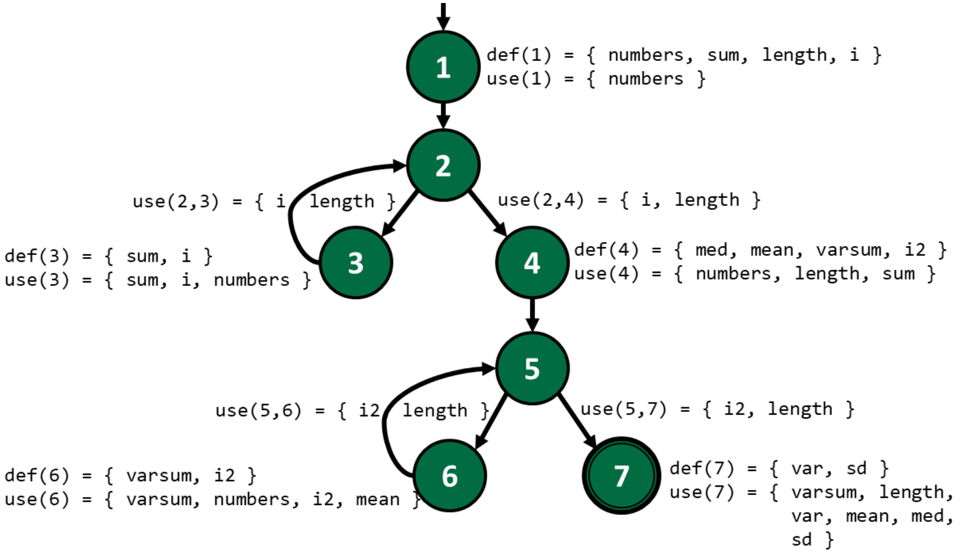
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	[4,5,6]

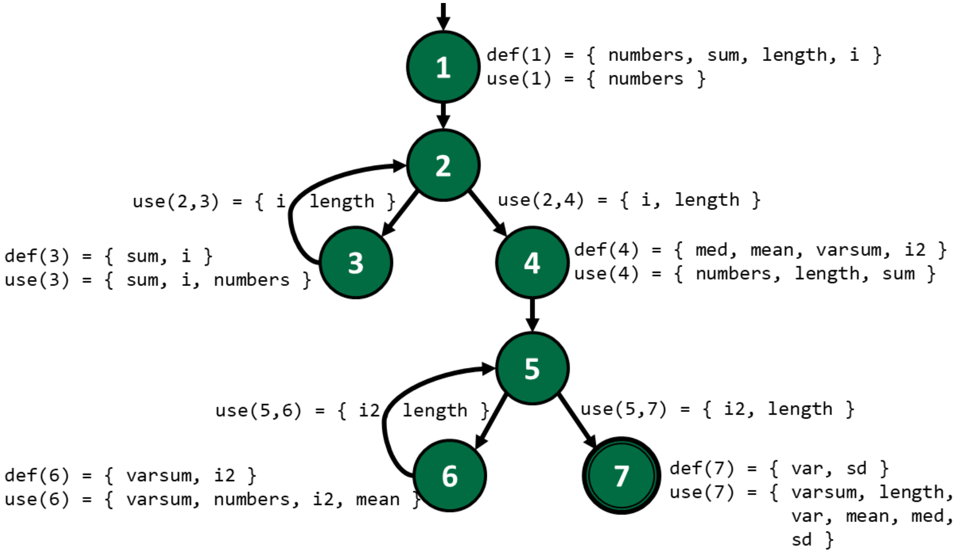
DU-Paths for *i2*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	[4,5,6] [4,5,7]

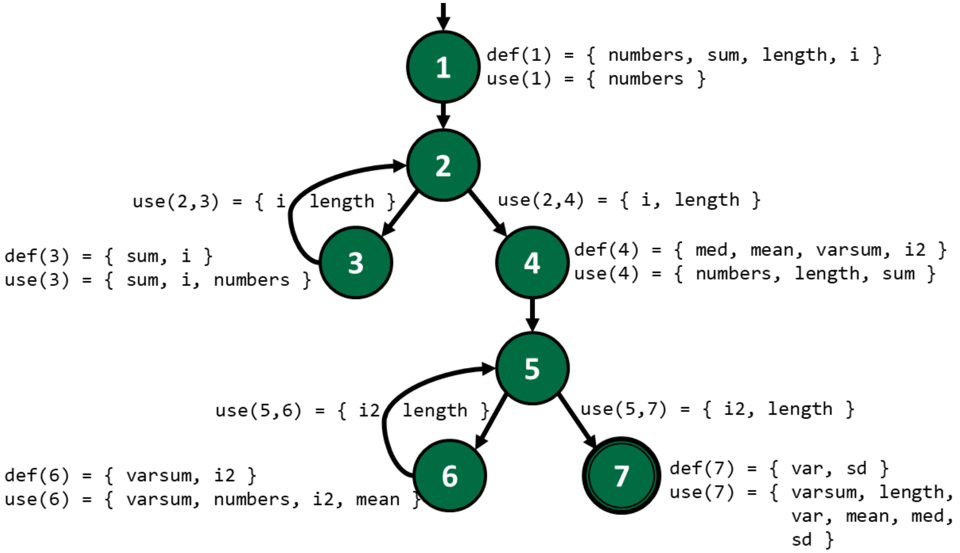
DU-Paths for *i2*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6))	[4,5,6]
	(4,(5,7))	[4,5,7]
	(4,6)	[4,5,6]
	(6,(5,6))	
	(6,(5,7))	
	(6,6)	

DU-Paths for *i2*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

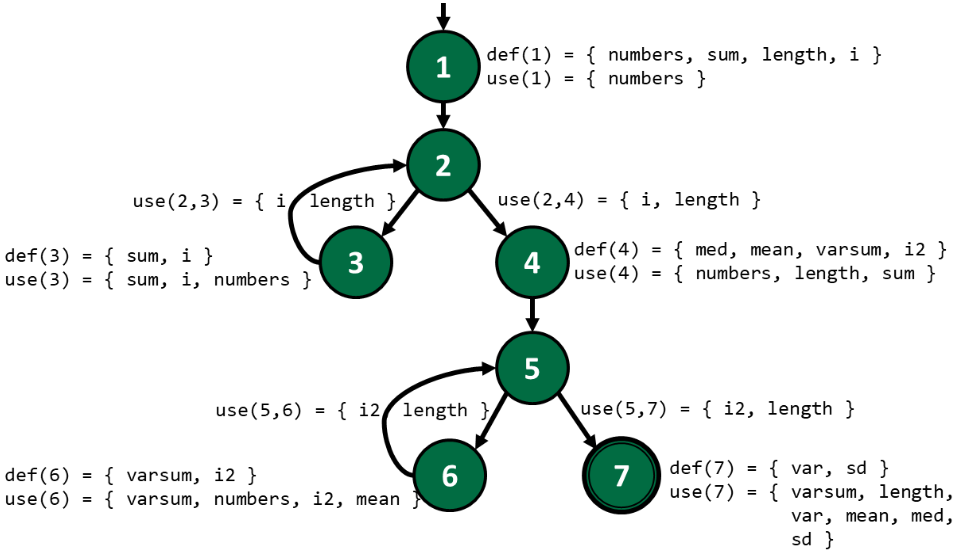
use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6))	[4,5,6]
	(4,(5,7))	[4,5,7]
	(4,6)	[4,5,6]
	(6,(5,6))	[6,5,6]
	(6,(5,7))	
	(6,6)	

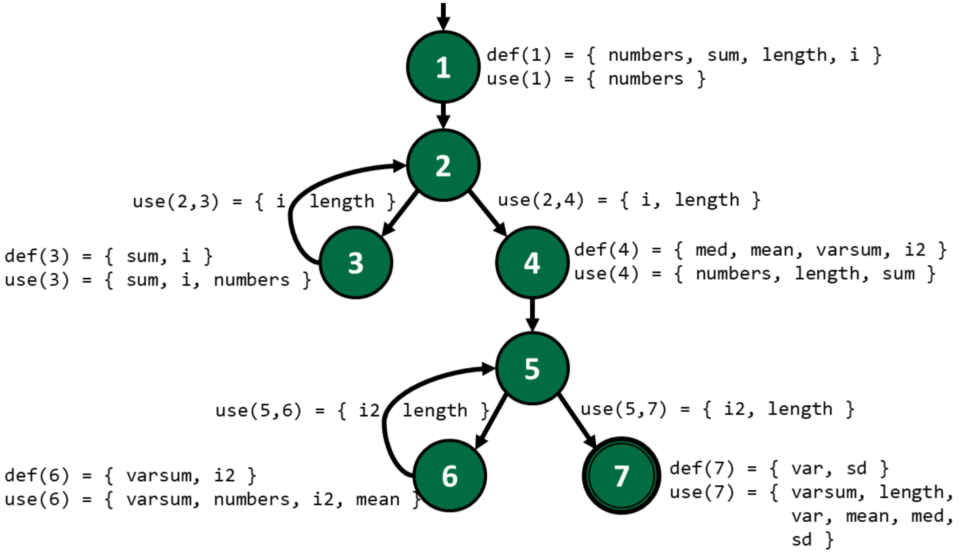
DU-Paths for *i2*



```
def(1) = { numbers, sum, length, i }
use(1) = { numbers }
use(2,3) = { i, length }
use(2,4) = { i, length }
def(3) = { sum, i }
use(3) = { sum, i, numbers }
def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }
use(5,6) = { i2, length }
use(5,7) = { i2, length }
def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }
def(7) = { var, sd }
use(7) = { varsum, length,
           var, mean, med,
           sd }
```

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6))	[4,5,6]
	(4,(5,7))	[4,5,7]
	(4,6)	[4,5,6]
	(6,(5,6))	[6,5,6]
	(6,(5,7))	[6,5,7]
	(6,6)	

DU-Paths for *i2*



def(1) = { numbers, sum, length, i }
use(1) = { numbers }

use(2,3) = { i, length }

use(2,4) = { i, length }

def(3) = { sum, i }
use(3) = { sum, i, numbers }

def(4) = { med, mean, varsum, i2 }
use(4) = { numbers, length, sum }

use(5,6) = { i2, length }

use(5,7) = { i2, length }

def(6) = { varsum, i2 }
use(6) = { varsum, numbers, i2, mean }

def(7) = { var, sd }
use(7) = { varsum, length, var, mean, med, sd }

Variable	DU-Pairs	DU-Paths
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	[4,5,6] [4,5,7] [4,5,6] [6,5,6] [6,5,7] [6,5,6]

DU-Paths for computeStats

Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	[1,2,3] [1,2,4] [1,2,4,5,6]
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	[1,2,3] [1,2,4] [1,2,4] [1,2,4,5,6] [1,2,4,5,7] [1,2,4,5,7]
med	(4,7)	[4,5,7]
var	~~(7,7)~~	--
sd	~~(7,7)~~	--
mean	(4,6) (4,7)	[4,5,6] [4,5,7]

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	[1,2,3] [1,2,4] [3,2,3] [3,2,4]
varsum	(4,6) (4,7) (6,6) (6,7)	[4,5,6] [4,5,7] [6,5,6] [6,5,7]
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	[1,2,3] [1,2,4] [1,2,3] [3,2,3] [3,2,4] [3,2,3]
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	[4,5,6] [4,5,7] [4,5,6] [6,5,6] [6,5,7] [6,5,6]

Unique DU-Paths

- 32 DU-Paths, but only 10 are unique
  - [1,2,3] 🟩
  - [1,2,4] 🟨
  - [1,2,4,5,6] 🟩
  - [1,2,4,5,7] 🟨
  - [4,5,7] 🟨
  - [4,5,6] 🟩
  - [3,2,3] 🟥
  - [3,2,4] 🟩
  - [6,5,6] 🟥
  - [6,5,7] 🟩

🟨 3 don’t execute a loop

🟩 5 execute a loop at least once

🟥 2 execute a loop at least twice

All-Defs Coverage

Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	**[1,2,3]** [1,2,4] [1,2,4,5,6]
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	**[1,2,3]** [1,2,4] [1,2,4] [1,2,4,5,6] [1,2,4,5,7] [1,2,4,5,7]
med	(4,7)	**[4,5,7]**
var	~~(7,7)~~	--
sd	~~(7,7)~~	--
mean	(4,6) (4,7)	[4,5,6] **[4,5,7]**

Tip: choose DU-paths to maximize coverage (e.g. maximize reuse)

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	**[1,2,3]** [1,2,4] **[3,2,3]** [3,2,4]
varsum	(4,6) (4,7) (6,6) (6,7)	[4,5,6] **[4,5,7]** **[6,5,6]** [6,5,7]
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	**[1,2,3]** [1,2,4] [1,2,3] **[3,2,3]** [3,2,4] [3,2,3]
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	[4,5,6] **[4,5,7]** [4,5,6] **[6,5,6]** [6,5,7] [6,5,6]

For All-Defs coverage, we must cover at least one DU-path from each def of each variable

All-Uses Coverage

Variable	DU-Pairs	DU-Paths
numbers	(1,3) (1,4) (1,6)	**[1,2,3]** **[1,2,4]** **[1,2,4,5,6]**
length	(1,(2,3)) (1,(2,4)) (1,4) (1,(5,6)) (1,(5,7)) (1,7)	**[1,2,3]** **[1,2,4]** **[1,2,4]** **[1,2,4,5,6]** **[1,2,4,5,7]** **[1,2,4,5,7]**
med	(4,7)	**[4,5,7]**
var	~~(7,7)~~	--
sd	~~(7,7)~~	--
mean	(4,6) (4,7)	**[4,5,6]** **[4,5,7]**

Variable	DU-Pairs	DU-Paths
sum	(1,3) (1,4) (3,3) (3,4)	**[1,2,3]** **[1,2,4]** **[3,2,3]** **[3,2,4]**
varsum	(4,6) (4,7) (6,6) (6,7)	**[4,5,6]** **[4,5,7]** **[6,5,6]** **[6,5,7]**
i	(1,(2,3)) (1,(2,4)) (1,3) (3,(2,3)) (3,(2,4)) (3,3)	**[1,2,3]** **[1,2,4]** **[1,2,3]** **[3,2,3]** **[3,2,4]** **[3,2,3]**
i2	(4,(5,6)) (4,(5,7)) (4,6) (6,(5,6)) (6,(5,7)) (6,6)	**[4,5,6]** **[4,5,7]** **[4,5,6]** **[6,5,6]** **[6,5,7]** **[6,5,6]**

For All-Uses coverage, we must cover at least one DU-path from each def to each use (same as all-DU-paths in this case because there are no multiple paths from any def to any use in this graph)

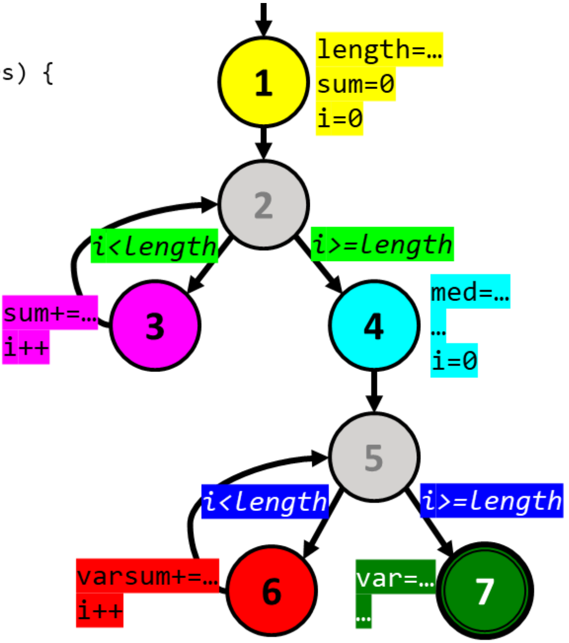
Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]		
[1,2,4]		
[1,2,4,5,6]		
[1,2,4,5,7]		
[4,5,7]		
[4,5,6]		
[3,2,3]		
[3,2,4]		
[6,5,6]		
[6,5,7]		

Test Paths and Test Inputs

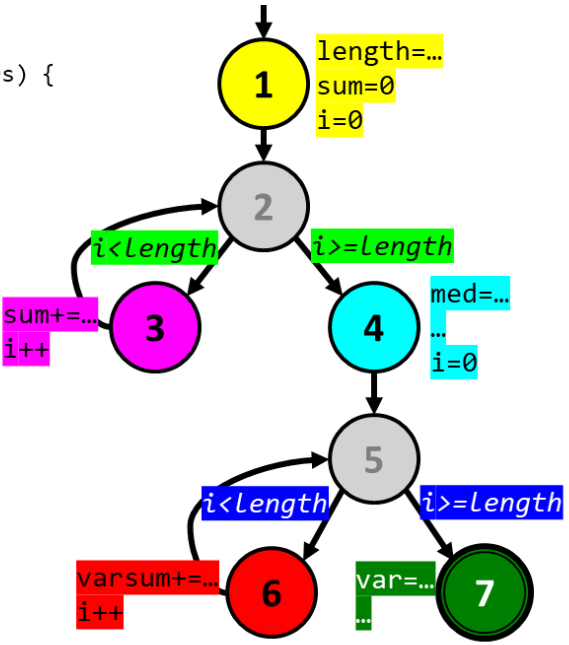
```
public static void computeStats (int[] numbers) {
   int length = numbers.length;
   double med, var, sd;
   double mean, sum, varsum;

   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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[1,2,3]		

Test Paths and Test Inputs

```
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);
}
```



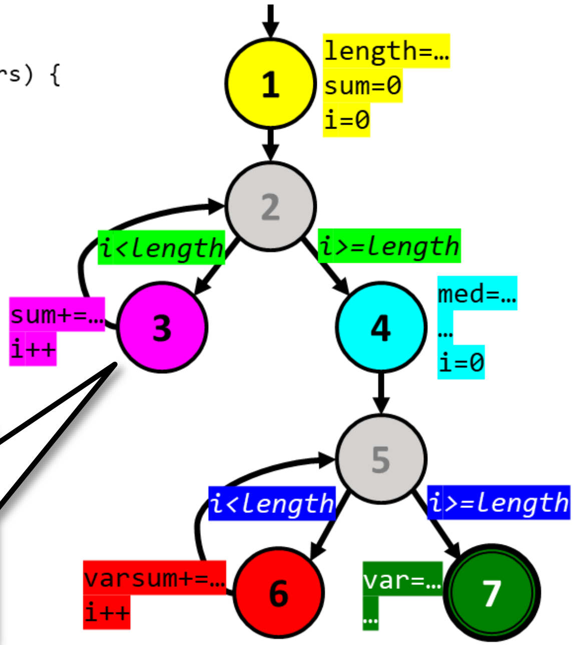
DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,7]	

Test Paths and Test Inputs

```
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;
```

```
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);
}
```

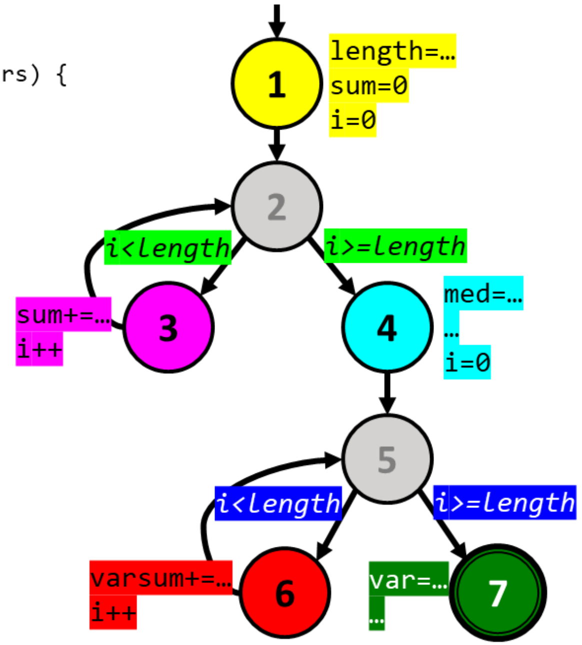
Loops are coupled by same inputs, so we can't skip the first loop and execute the second!



DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,7]	INFEASIBLE

Test Paths and Test Inputs

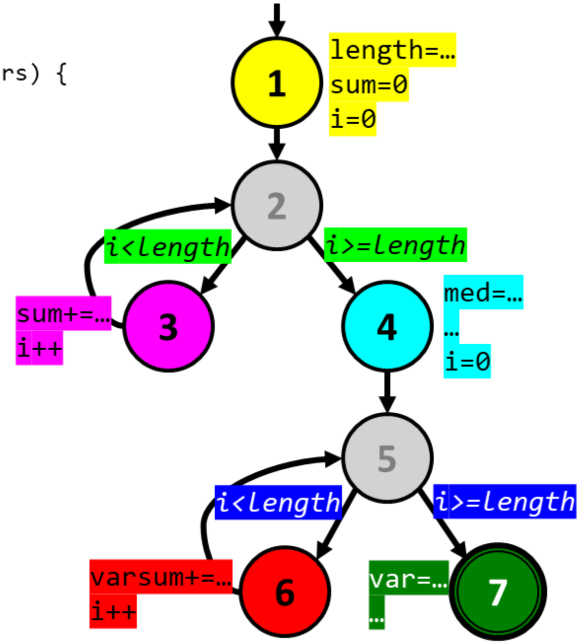
```
public static void computeStats (int[] numbers) {
   int length = numbers.length;
   double med, var, sd;
   double mean, sum, varsum;

   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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	

Test Paths and Test Inputs

```
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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }

Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]		
[1,2,4,5,6]		
[1,2,4,5,7]		
[4,5,7]		
[4,5,6]		
[3,2,3]		
[3,2,4]		
[6,5,6]		
[6,5,7]		

Test Paths and Test Inputs

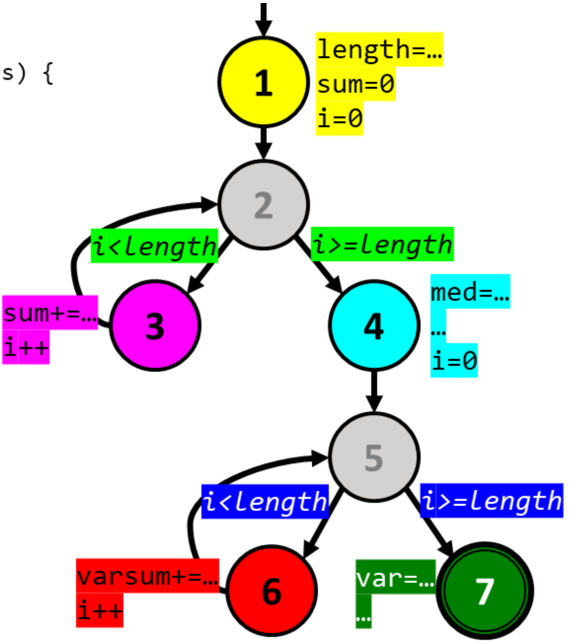
- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]		
[1,2,4,5,6]		
[1,2,4,5,7]		
[4,5,7]		
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]		
[3,2,4]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[6,5,6]		
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

This test path satisfies other DU-paths too!

Test Paths and Test Inputs

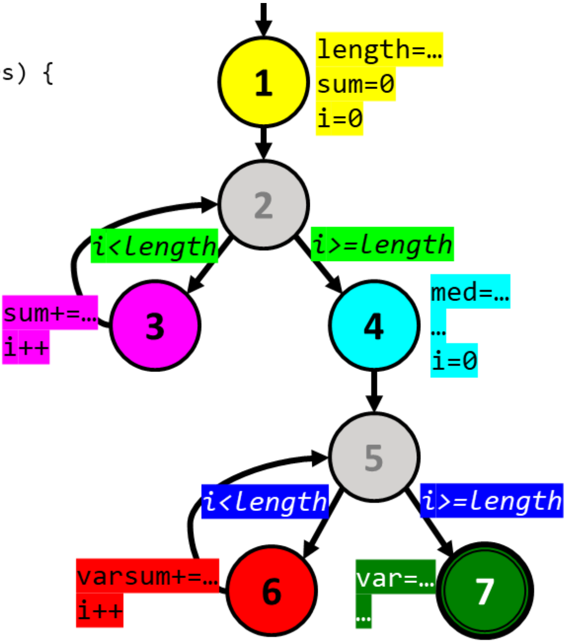
```
public static void computeStats (int[] numbers) {
   int length = numbers.length;
   double med, var, sd;
   double mean, sum, varsum;

   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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[1,2,4]		

Test Paths and Test Inputs

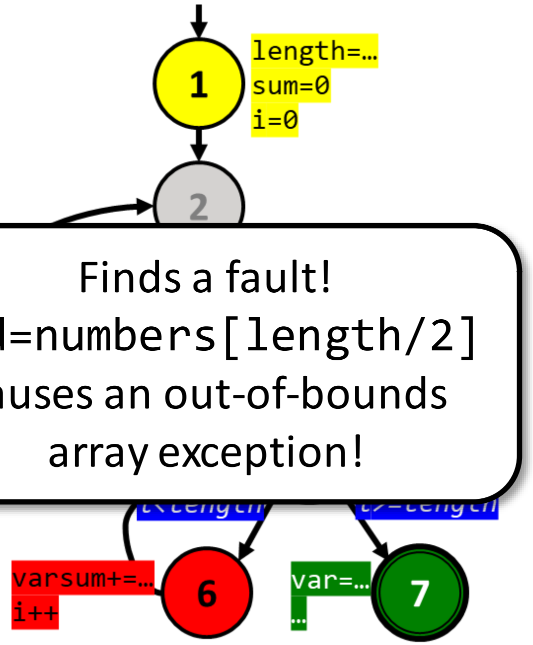
```
public static void computeStats (int[] numbers) {
   int length = numbers.length;
   double med, var, sd;
   double mean, sum, varsum;

   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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[1,2,4]	[1,2,4,5,7]	

Test Paths and Test Inputs

```
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);
}
```



Finds a fault!
med=numbers[length/2]
causes an out-of-bounds array exception!

DU-Path	Test Path	Test Input numbers={?}
[1,2,4]	[1,2,4,5,7]	{ }

Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]	[1,2,4,5,7]	{ }
[1,2,4,5,6]		
[1,2,4,5,7]		
[4,5,7]		
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]		
[3,2,4]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[6,5,6]		
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

Test Paths and Test Inputs

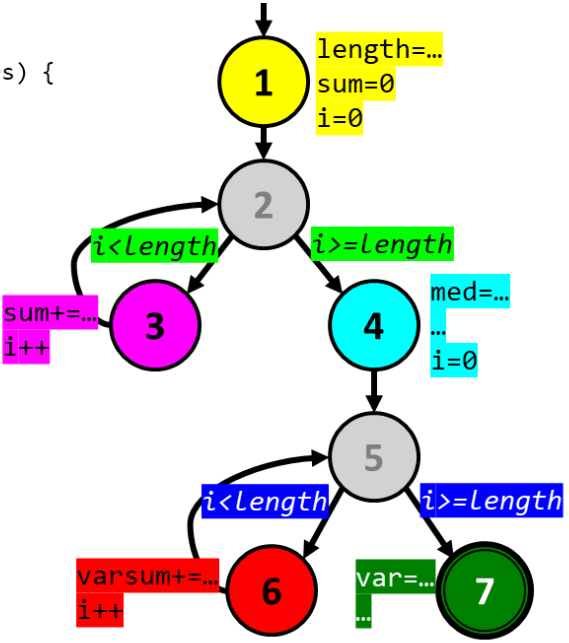
- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]	[1,2,4,5,7]	{ }
[1,2,4,5,6]		
[1,2,4,5,7]	[1,2,4,5,7]	{ }
[4,5,7]	[1,2,4,5,7]	{ }
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]		
[3,2,4]		{ 1 }
[6,5,6]		
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

This test path satisfies other DU-paths too!

Test Paths and Test Inputs

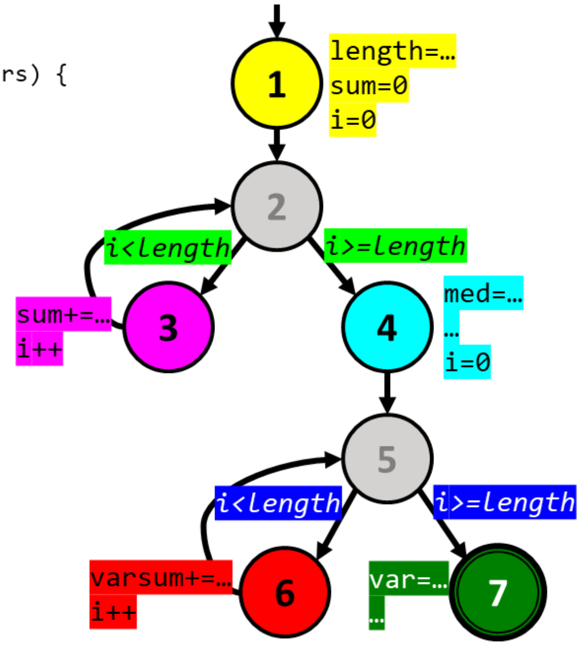
```
public static void computeStats (int[] numbers) {
   int length = numbers.length;
   double med, var, sd;
   double mean, sum, varsum;

   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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[1,2,4,5,6]		

Test Paths and Test Inputs

```
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);
}
```

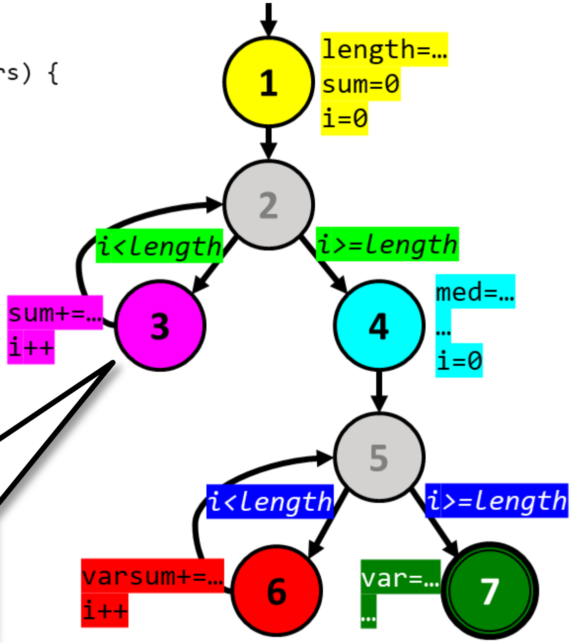


DU-Path	Test Path	Test Input numbers={?}
[1,2,4,5,6]	[1,2,4,5,6,5,7]	

Test Paths and Test Inputs

```
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;
```

```
  System.out.println("variance: " + var);
  System.out.println("std dev:  " + sd);
}
```



Loops are coupled by same inputs, so we can't skip the first loop and execute the second!

DU-Path	Test Path	Test Input numbers={?}
[1,2,4,5,6]	[1,2,4,5,6,5,7]	INFEASIBLE!

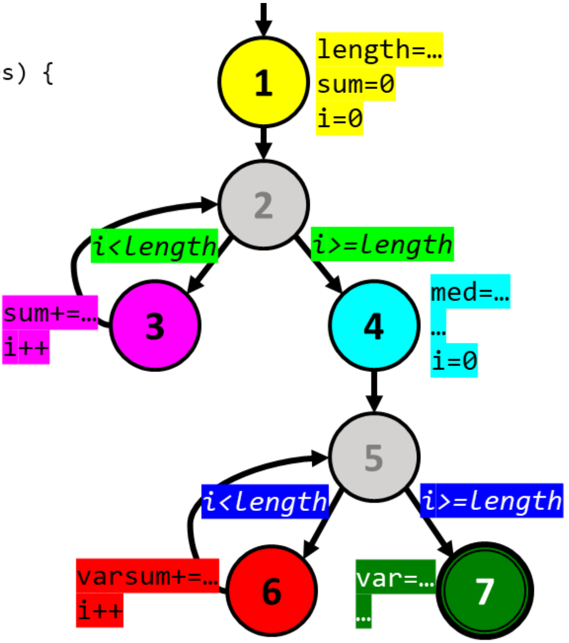
Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]	[1,2,4,5,7]	{ }
[1,2,4,5,6]	INFEASIBLE	
[1,2,4,5,7]	[1,2,4,5,7]	{ }
[4,5,7]	[1,2,4,5,7]	{ }
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]		
[3,2,4]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[6,5,6]		
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

Test Paths and Test Inputs

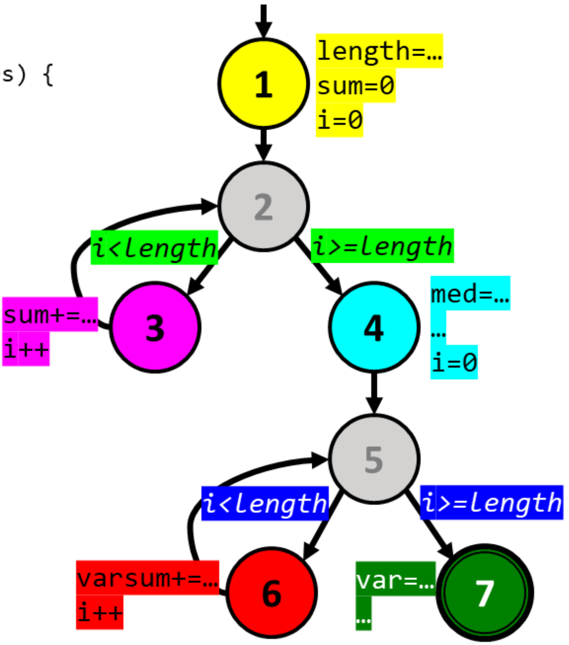
```
public static void computeStats (int[] numbers) {
 int length = numbers.length;
 double med, var, sd;
 double mean, sum, varsum;

 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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[3,2,3]		

Test Paths and Test Inputs

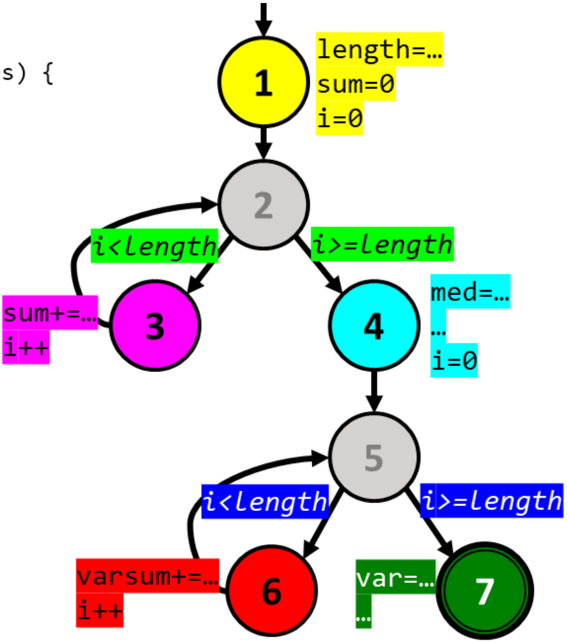
```
public static void computeStats (int[] numbers) {
  int length = numbers.length;
  double med, var, sd;
  double mean, sum, varsum;

  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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[3,2,3]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	

Test Paths and Test Inputs

```
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);
}
```



DU-Path	Test Path	Test Input numbers={?}
[3,2,3]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	{ 2, 3 }

Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]	[1,2,4,5,7]	{ }
[1,2,4,5,6]	INFEASIBLE	
[1,2,4,5,7]	[1,2,4,5,7]	{ }
[4,5,7]	[1,2,4,5,7]	{ }
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	{ 2, 3 }
[3,2,4]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[6,5,6]		
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]	[1,2,4,5,7]	{ }
[1,2,4,5,6]	INFEASIBLE	
[1,2		{ }
[4		{ }
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	{ 2, 3 }
[3,2,4]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[6,5,6]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	{ 2, 3 }
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

This test path satisfies other DU-paths too!

Test Paths and Test Inputs

- Find a test path and a test input for each DU-path to satisfy All-Uses coverage:

DU-Path	Test Path	Test Input numbers={?}
[1,2,3]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[1,2,4]	[1,2,4,5,7]	{ }
[1,2,4,5,6]	INFEASIBLE	
[1,2,4,5,7]	[1,2,4,5,7]	{ }
[4,5,7]	[1,2,4,5,7]	{ }
[4,5,6]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[3,2,3]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	{ 2, 3 }
[3,2,4]	[1,2,3,2,4,5,6,5,7]	{ 1 }
[6,5,6]	[1,2,3,2,3,2,4,5,6,5,6,5,7]	{ 2, 3 }
[6,5,7]	[1,2,3,2,4,5,6,5,7]	{ 1 }

All-Uses is satisfied by 3 tests