پوشش گراف برای کد

محمد تنهایی

Overview

- The most common application of graph criteria is to program source
- **Graph** : Usually the control flow graph (CFG)
- **Node coverage** : Execute every statement
- **Edge coverage** : Execute every branch
- **Loops** : Looping structures such as for loops, while loops, etc.
- **Data flow coverage** : Augment the CFG
  - defs are statements that assign values to variables
  - uses are statements that use variables

Control Flow Graphs

- A CFG models all executions of a method by describing control structures
- **Nodes** : Statements or sequences of statements (basic blocks)
- **Edges** : Transfers of control
- **Basic Block** : A sequence of statements such that if the first statement is executed, all statements will be (no branches)

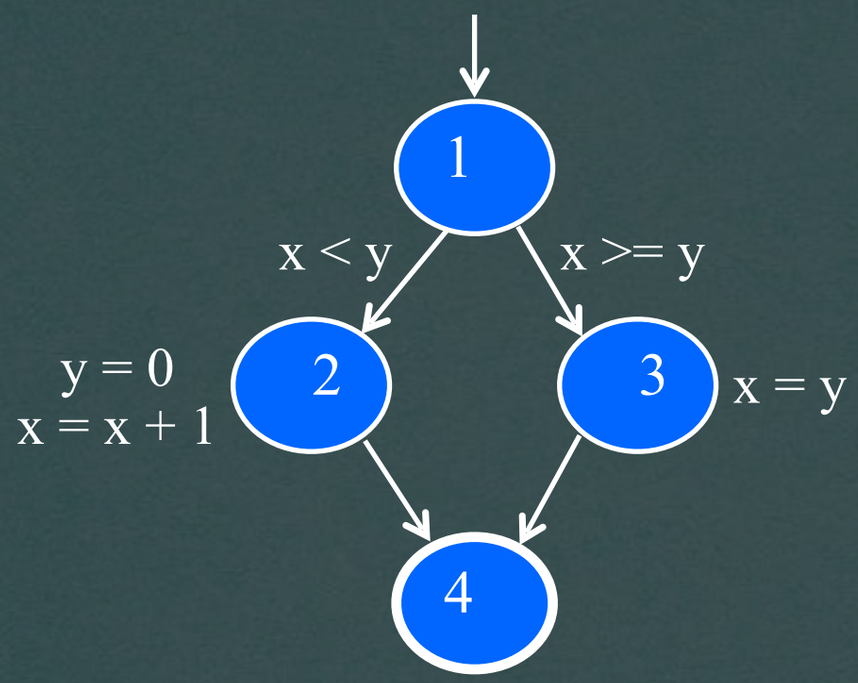
- CFGs are sometimes annotated with extra information
  - branch predicates
  - defs
  - uses
- Rules for translating statements into graphs ...

CFG : The if Statement

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

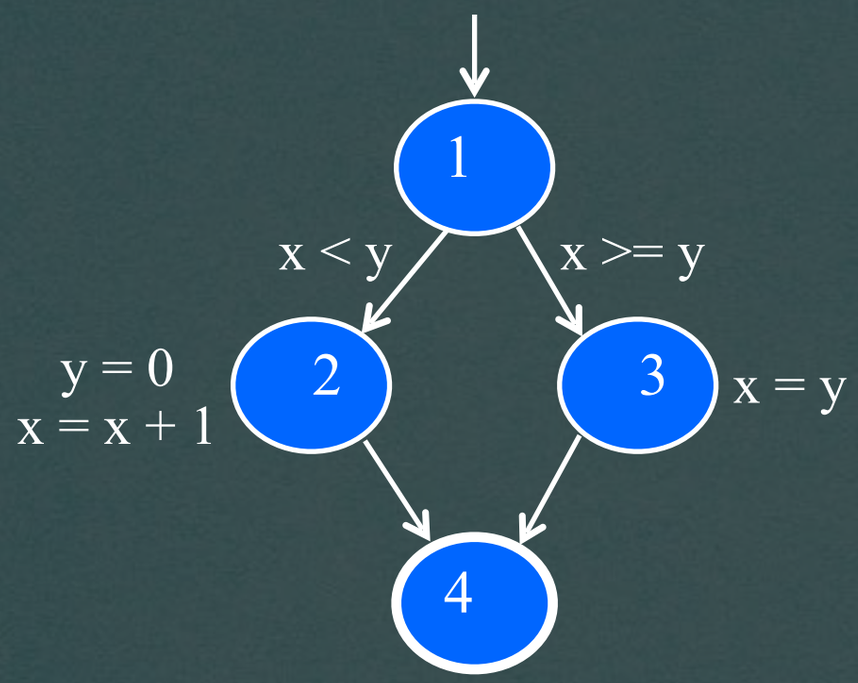
CFG : The if Statement

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



CFG : The if Statement

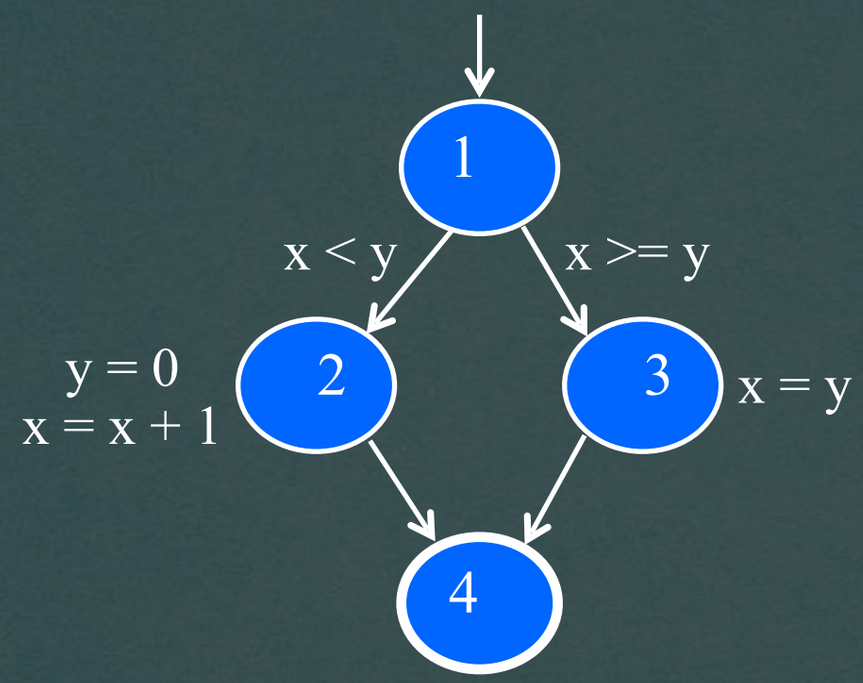
```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```



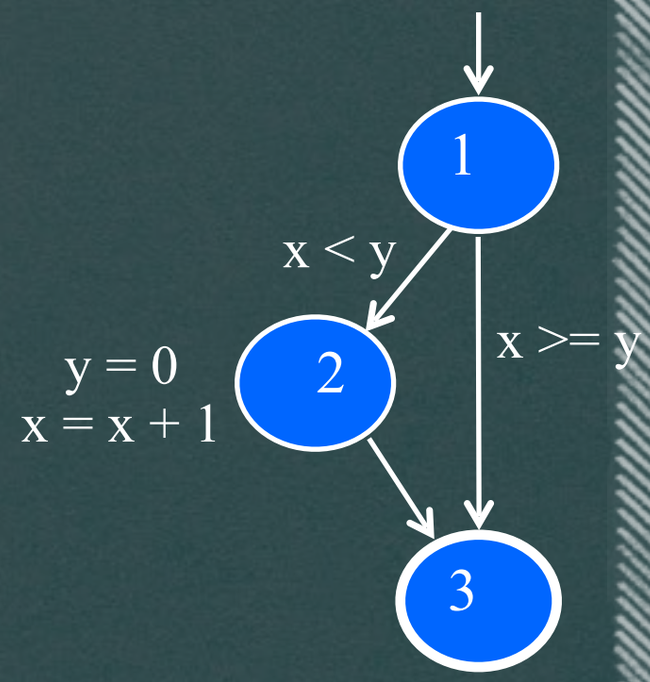

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
```

CFG : The if Statement

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




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

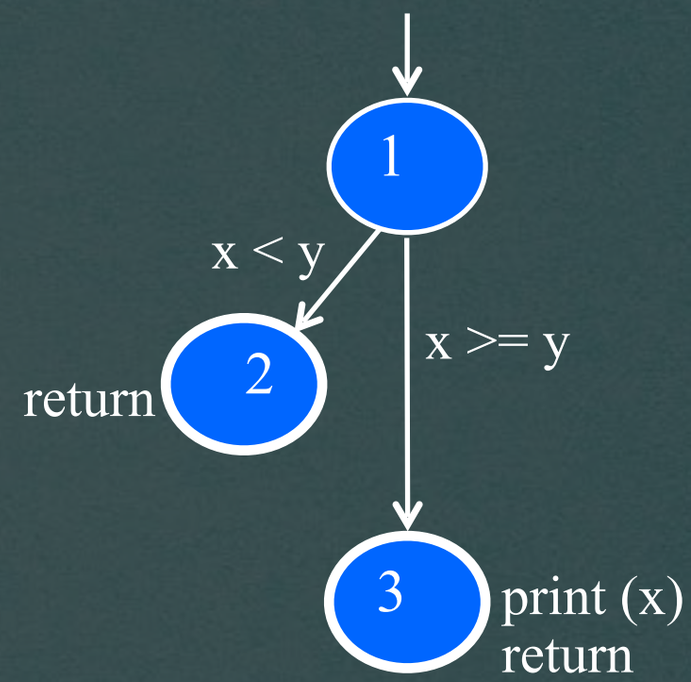


CFG : The if-Return Statement

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

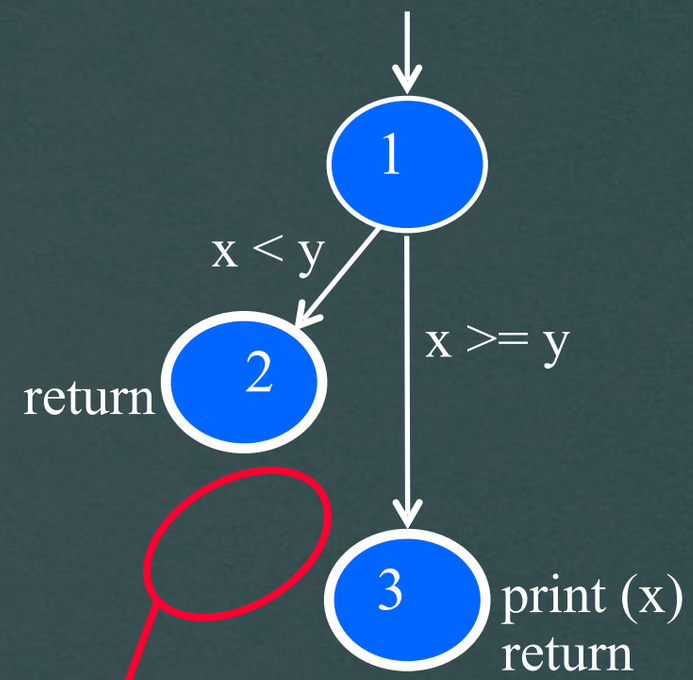
CFG : The if-Return Statement

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



CFG : The if-Return Statement

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



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

Loops

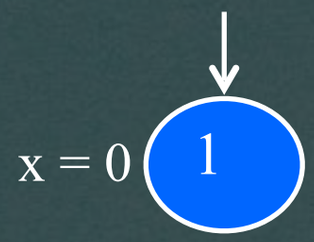
- Loops require "*extra*" nodes to be added
- Nodes that do not represent statements or basic blocks

CFG : while and for Loops

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

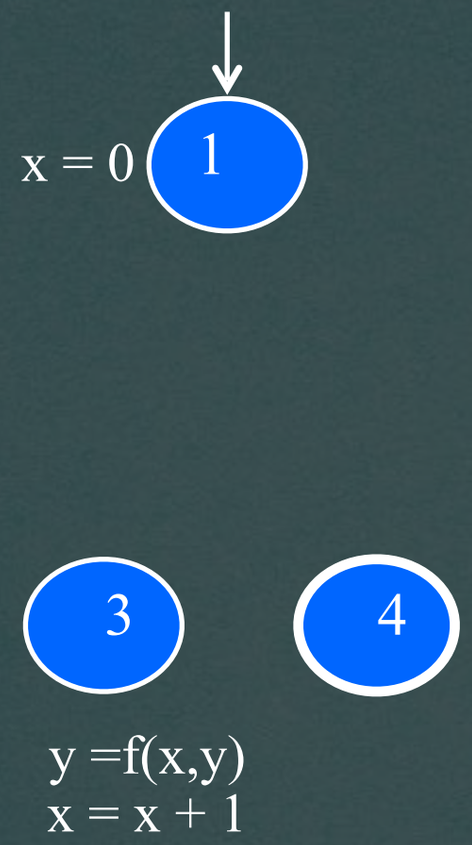
CFG : while and for Loops

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



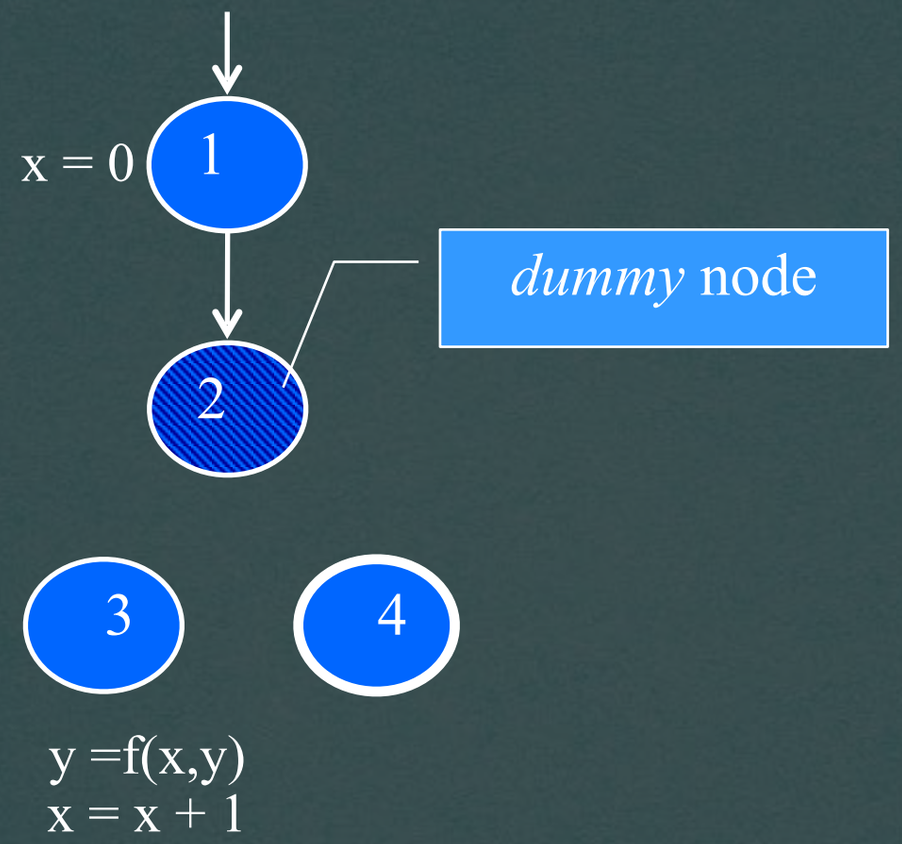
CFG : while and for Loops

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



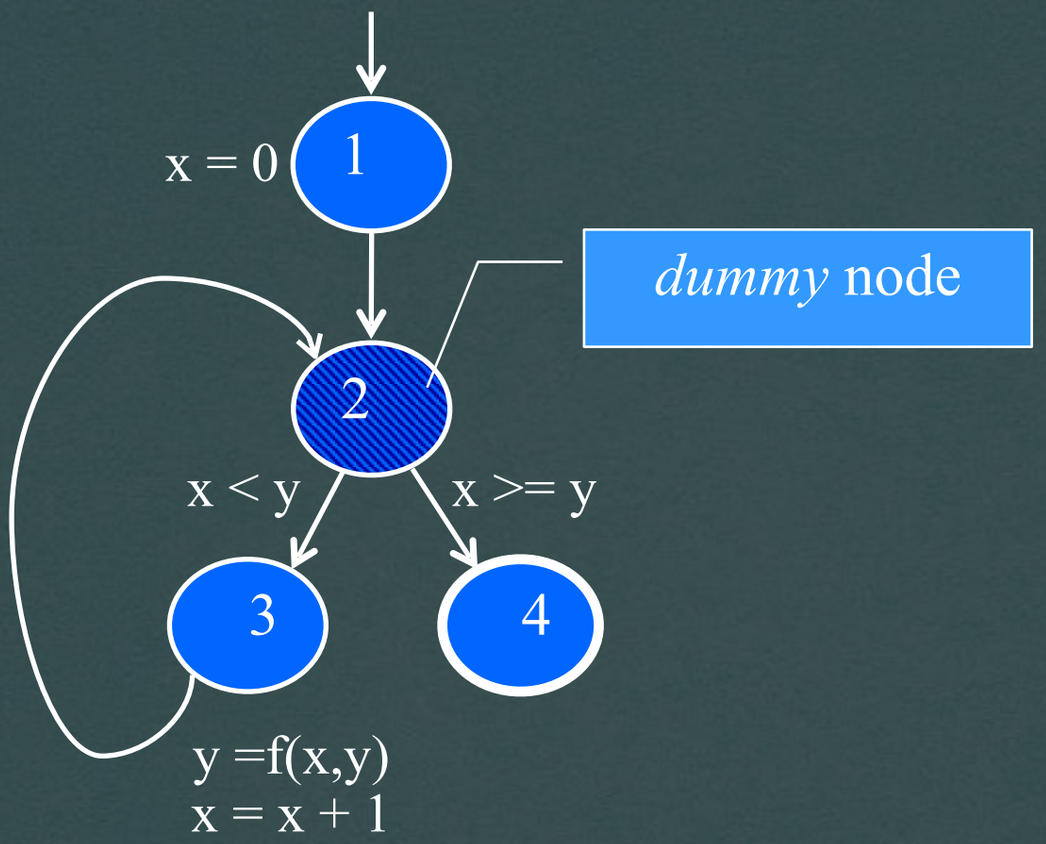
CFG : while and for Loops

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



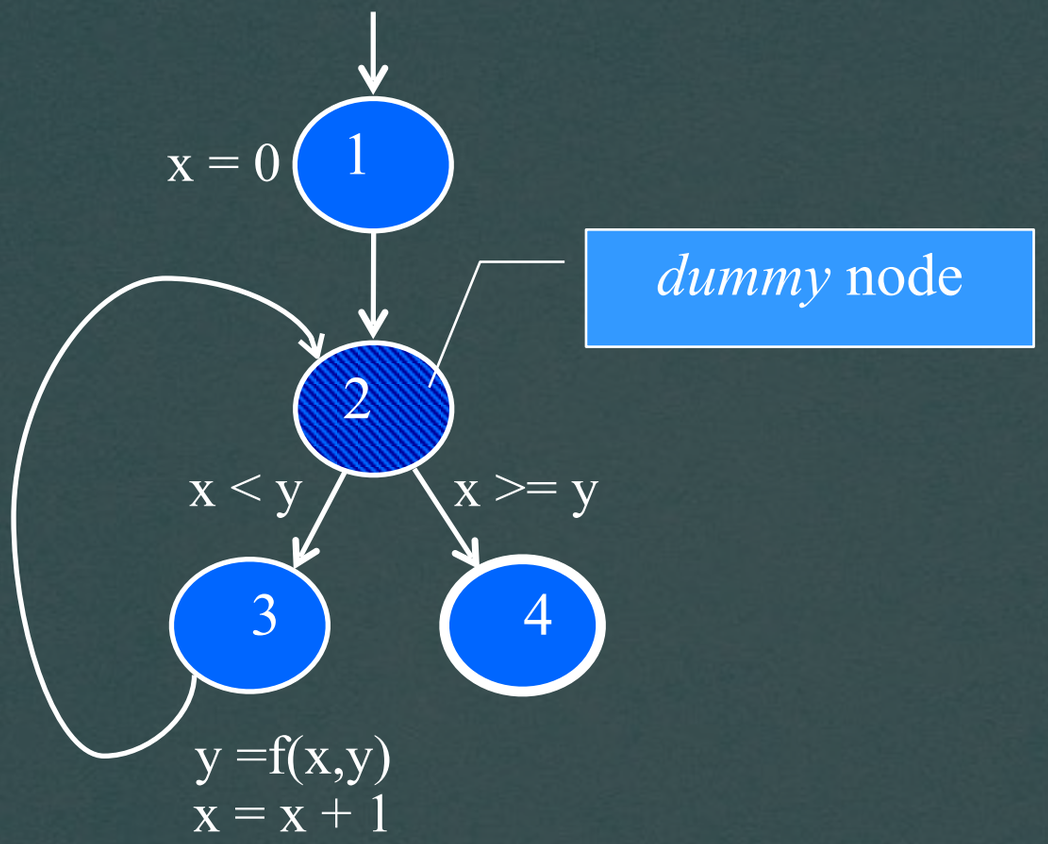
CFG : while and for Loops

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



CFG : while and for Loops

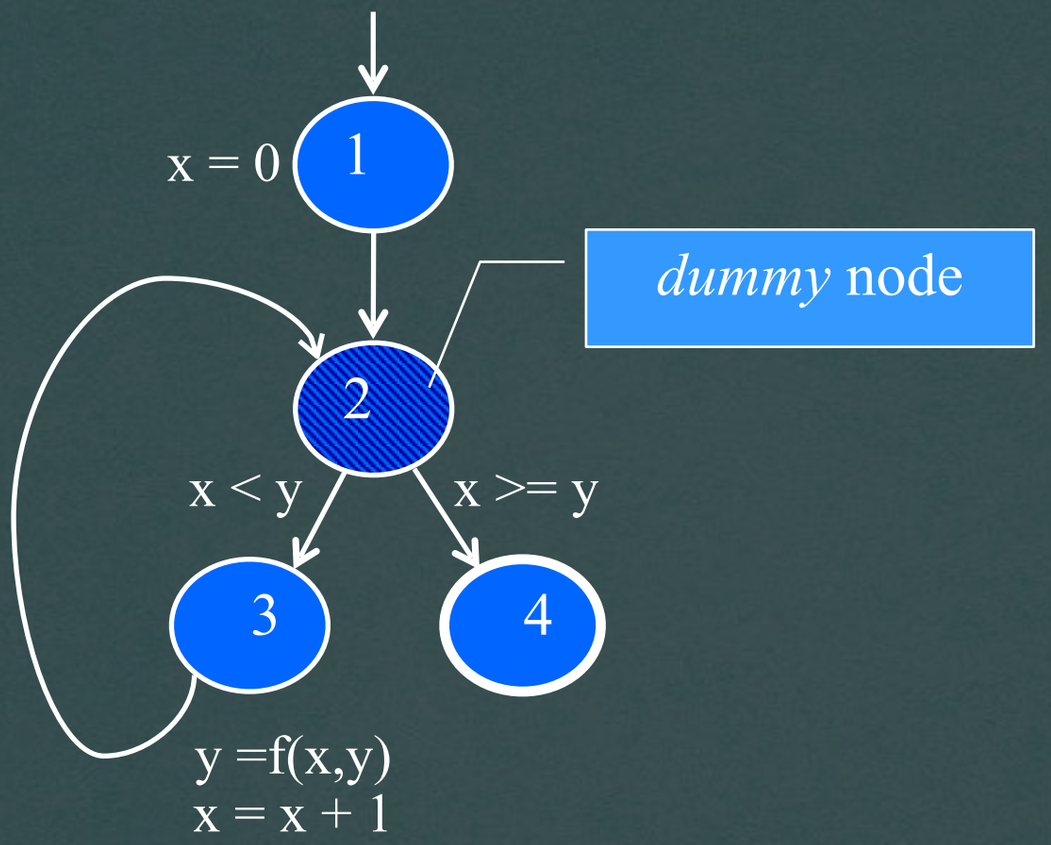
```
x = 0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
```



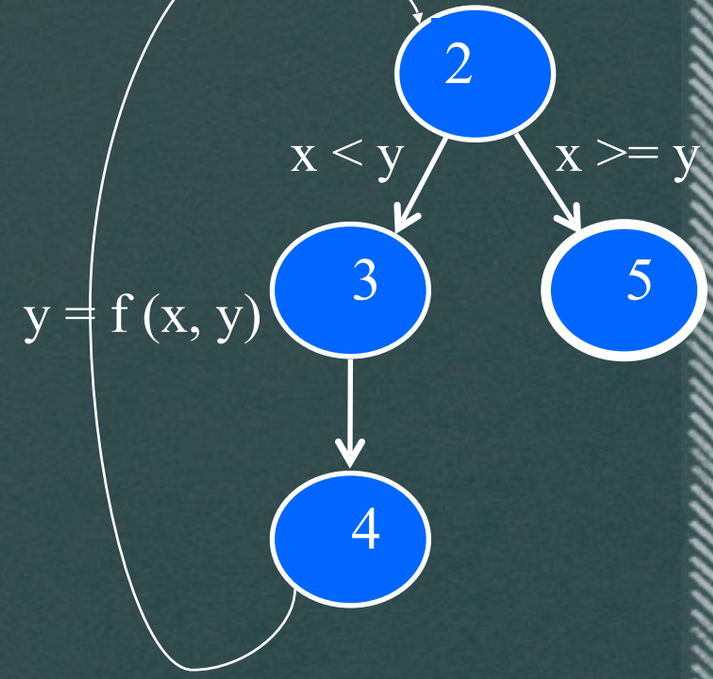

```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

CFG : while and for Loops

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

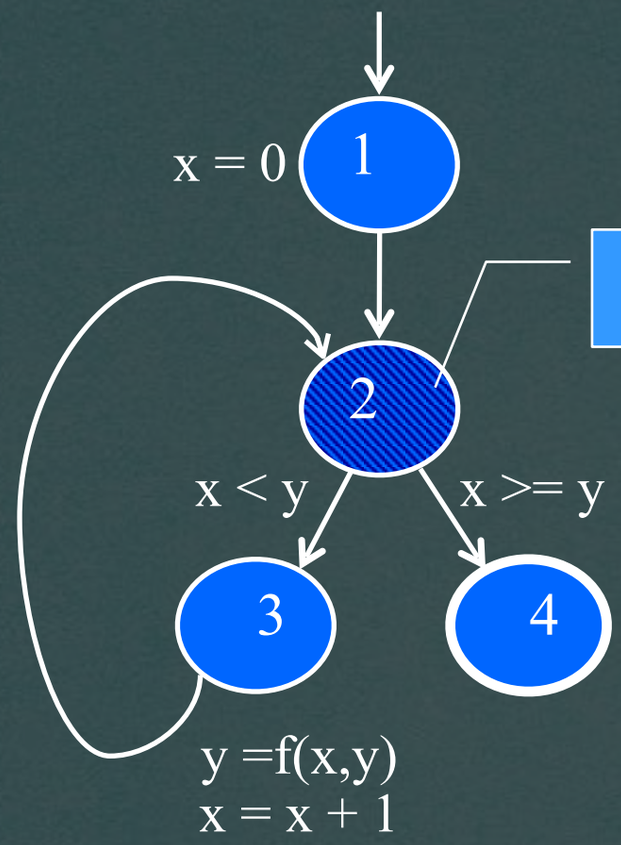



```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

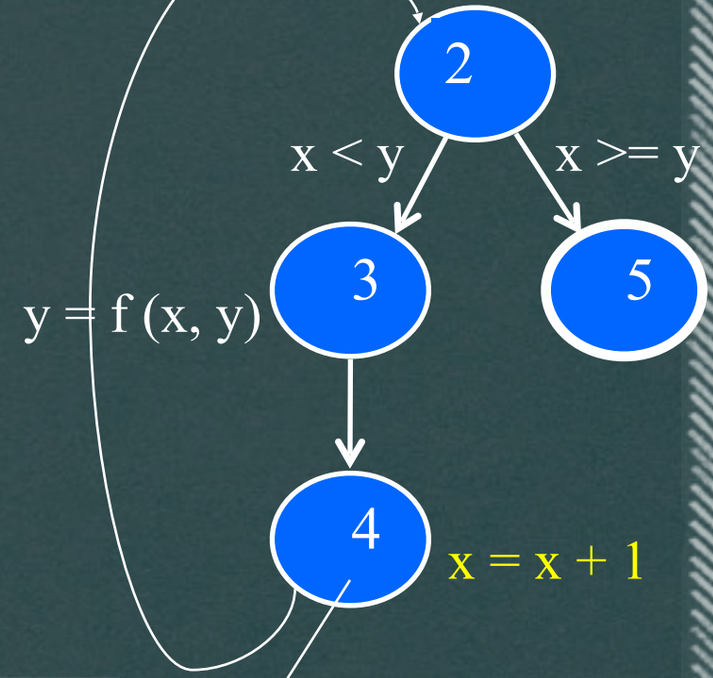


CFG : while and for Loops

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



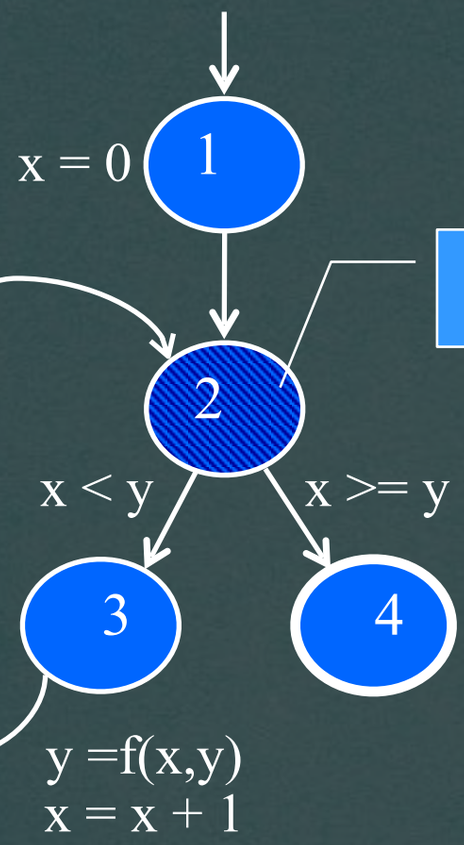

```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```



implicitly increments loop

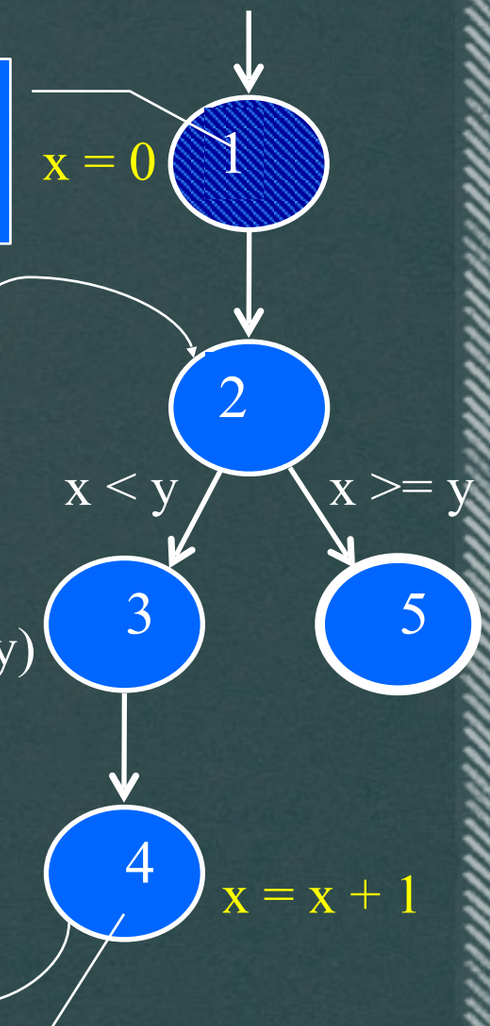
CFG : while and for Loops

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



dummy node

implicitly initializes loop




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

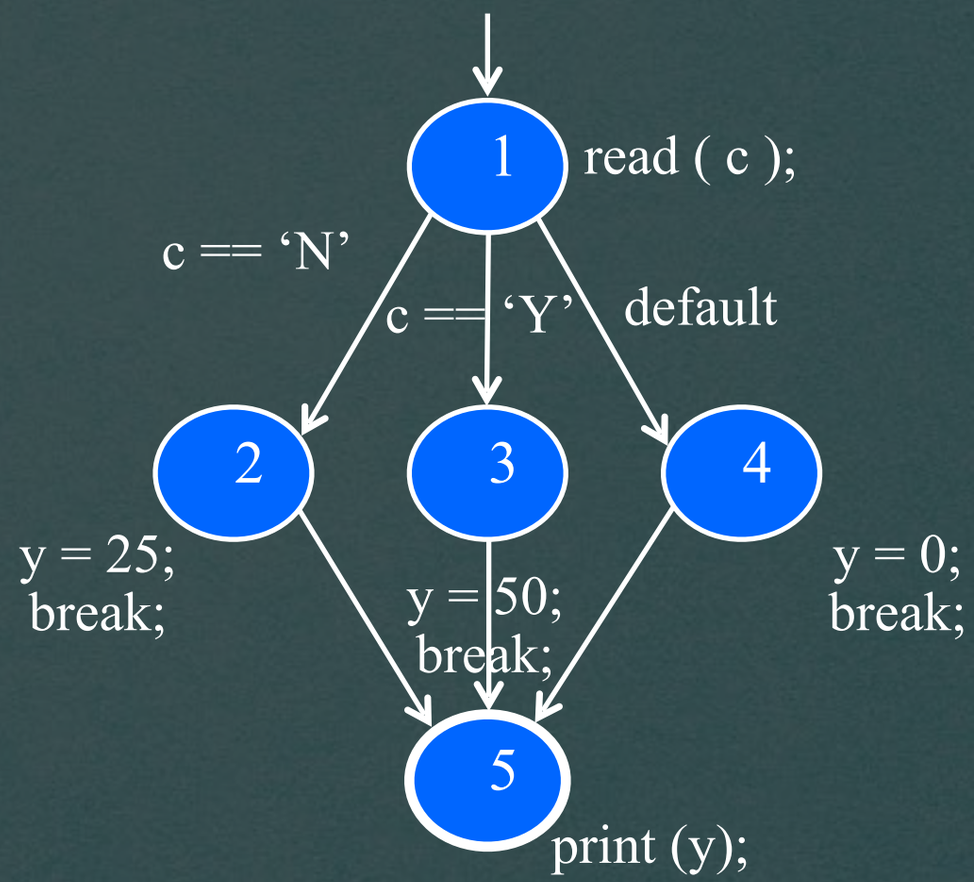
implicitly increments loop

CFG : The case (switch) Structure

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

CFG : The case (switch) Structure

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



Example Control Flow – Stats

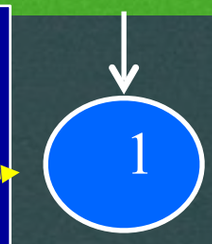
```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    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 ("standard deviation: " + sd);
}
```

Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```

Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```

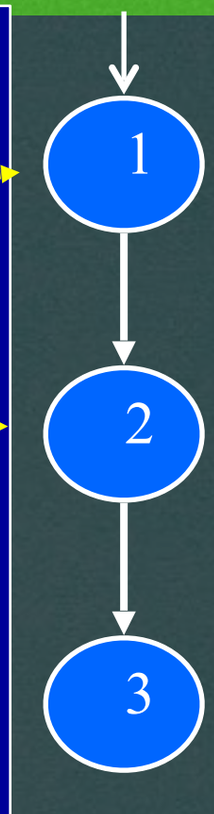


Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```

Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```

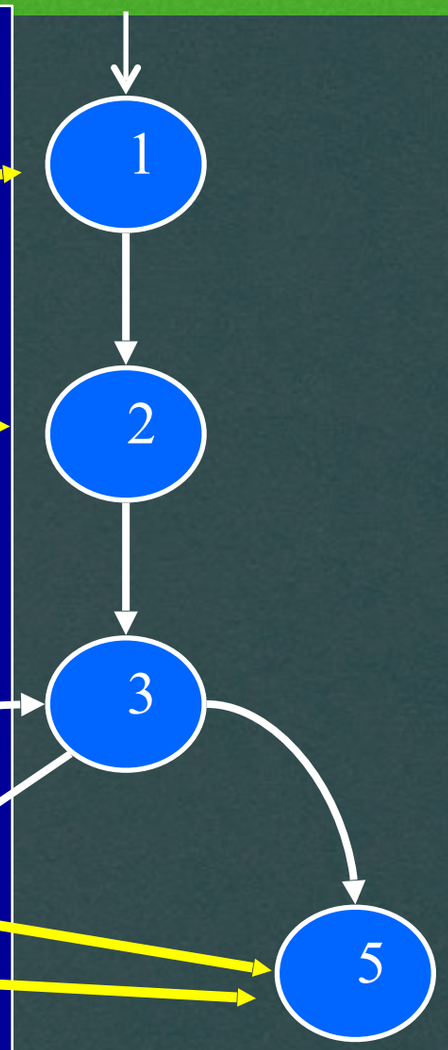


Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```

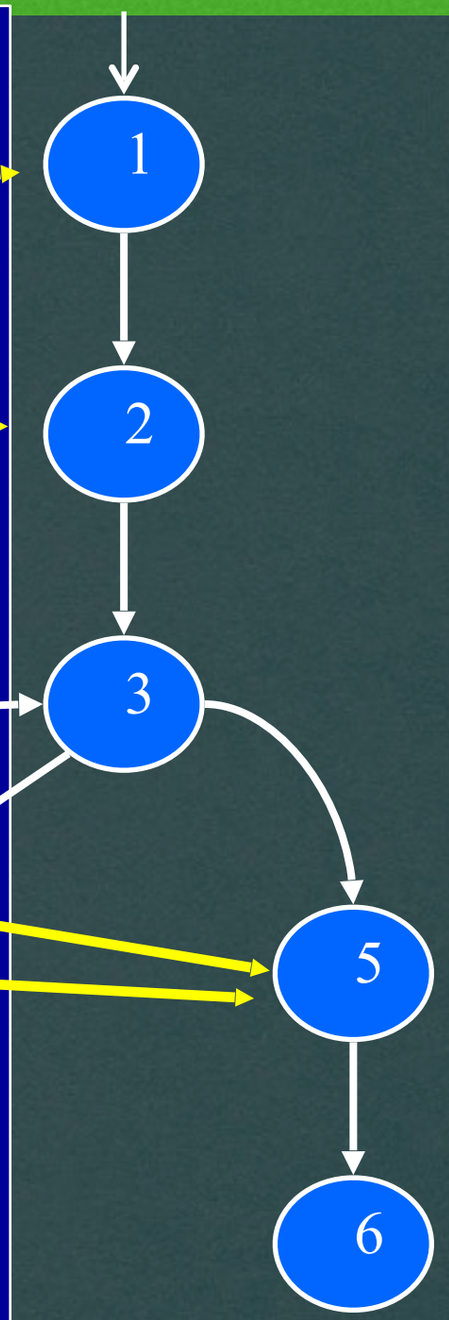
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



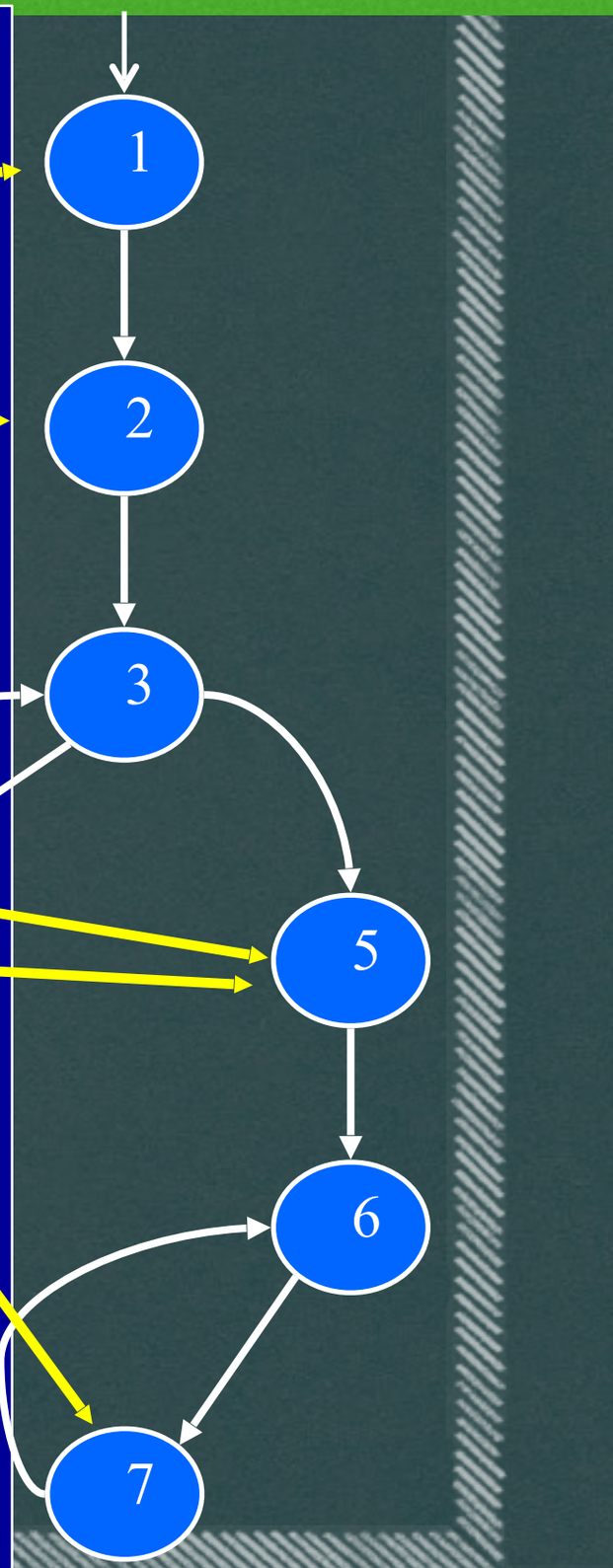
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



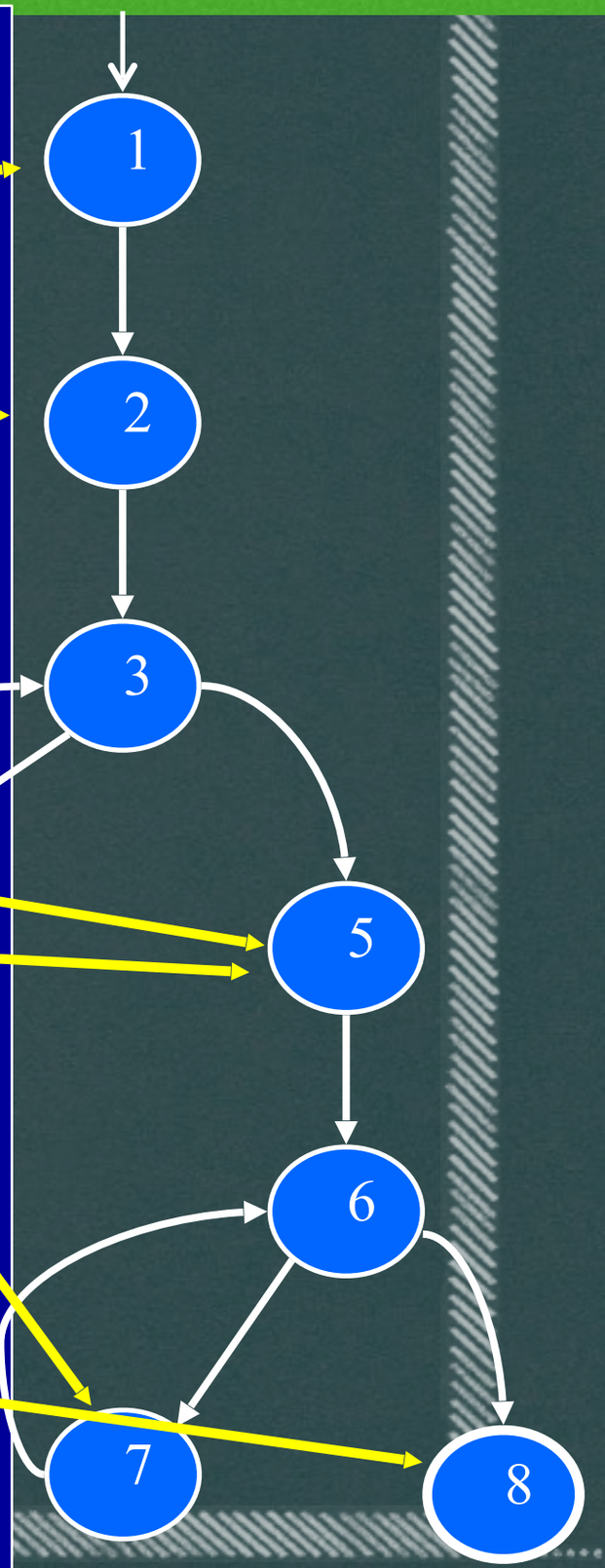
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



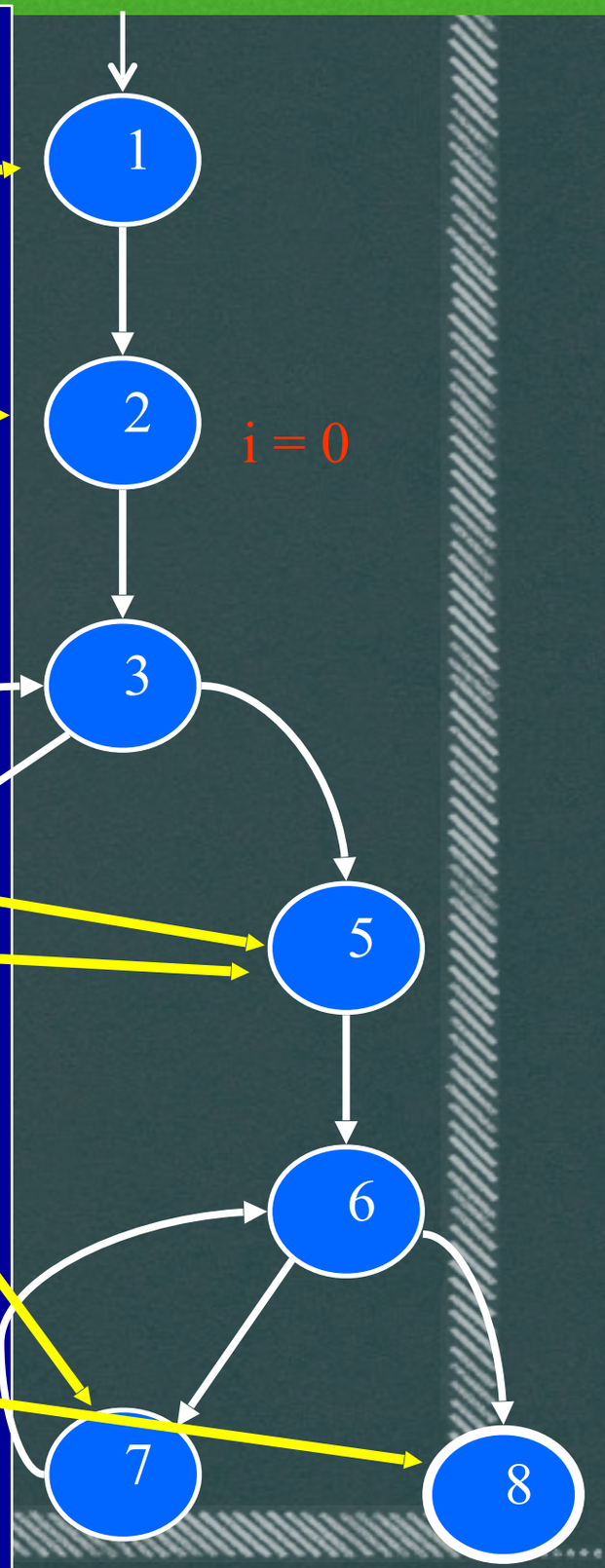
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



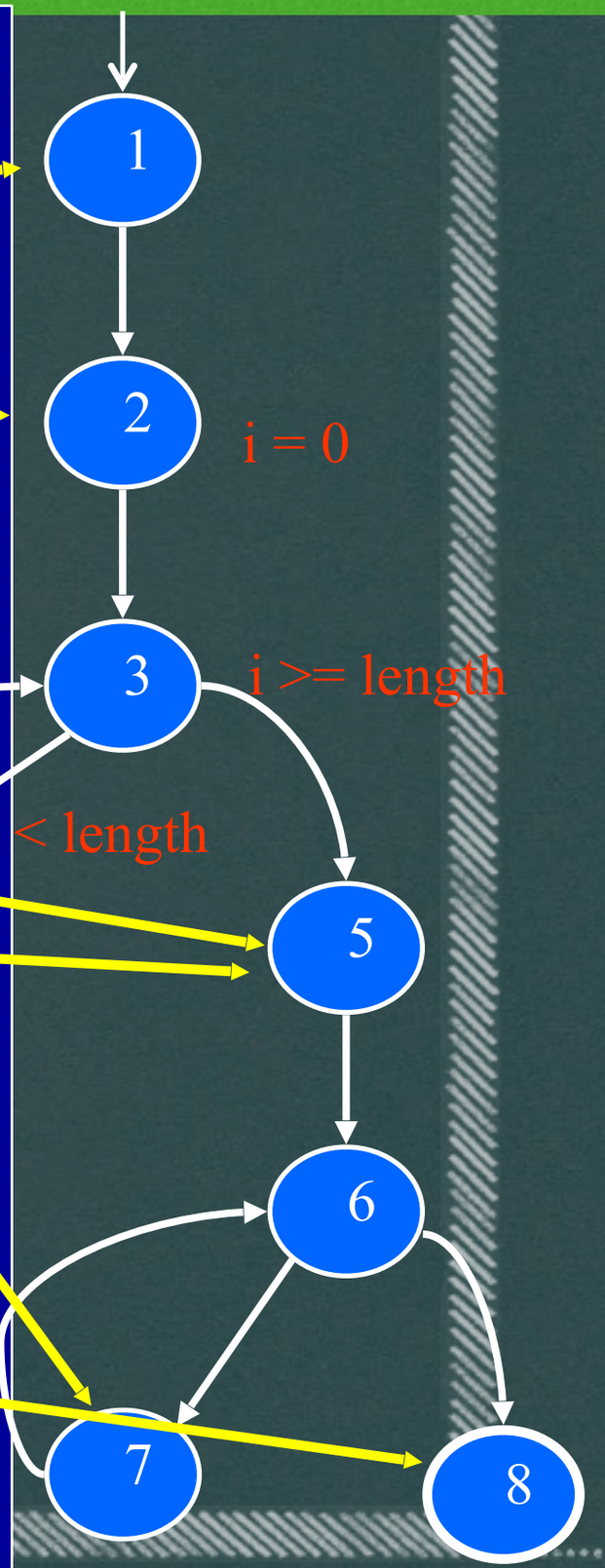
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



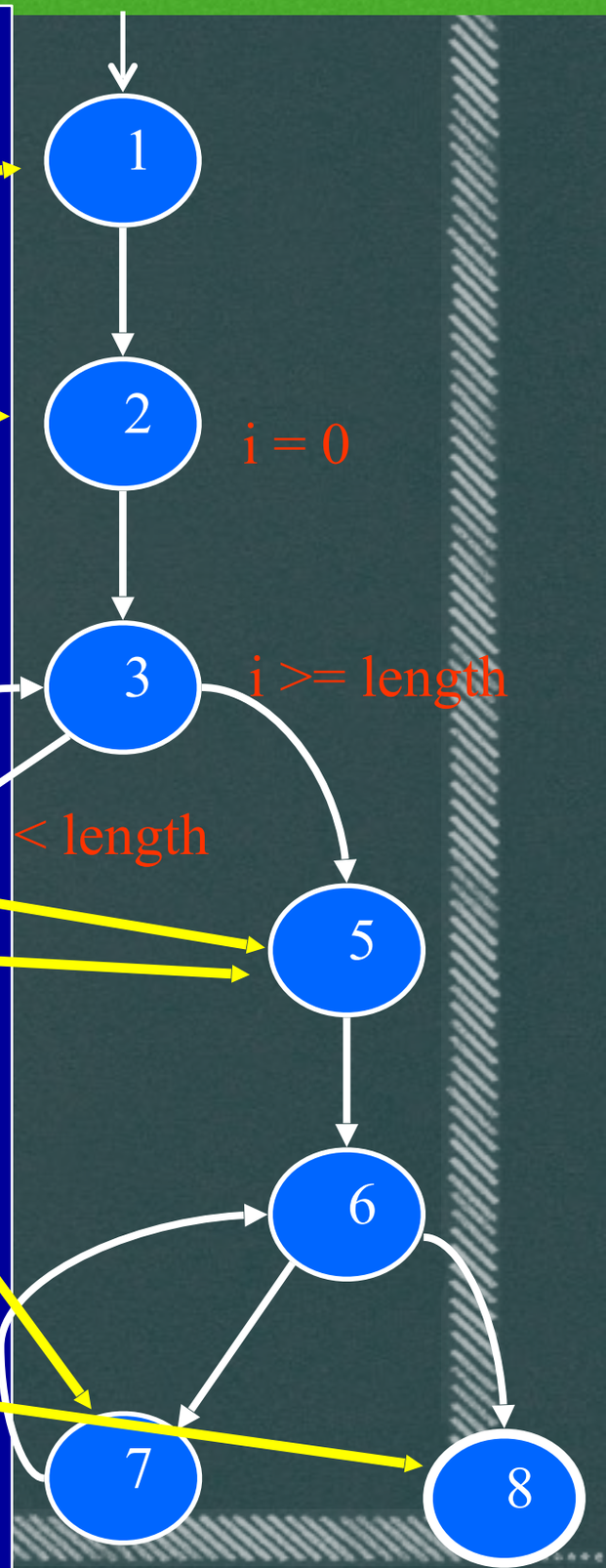
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



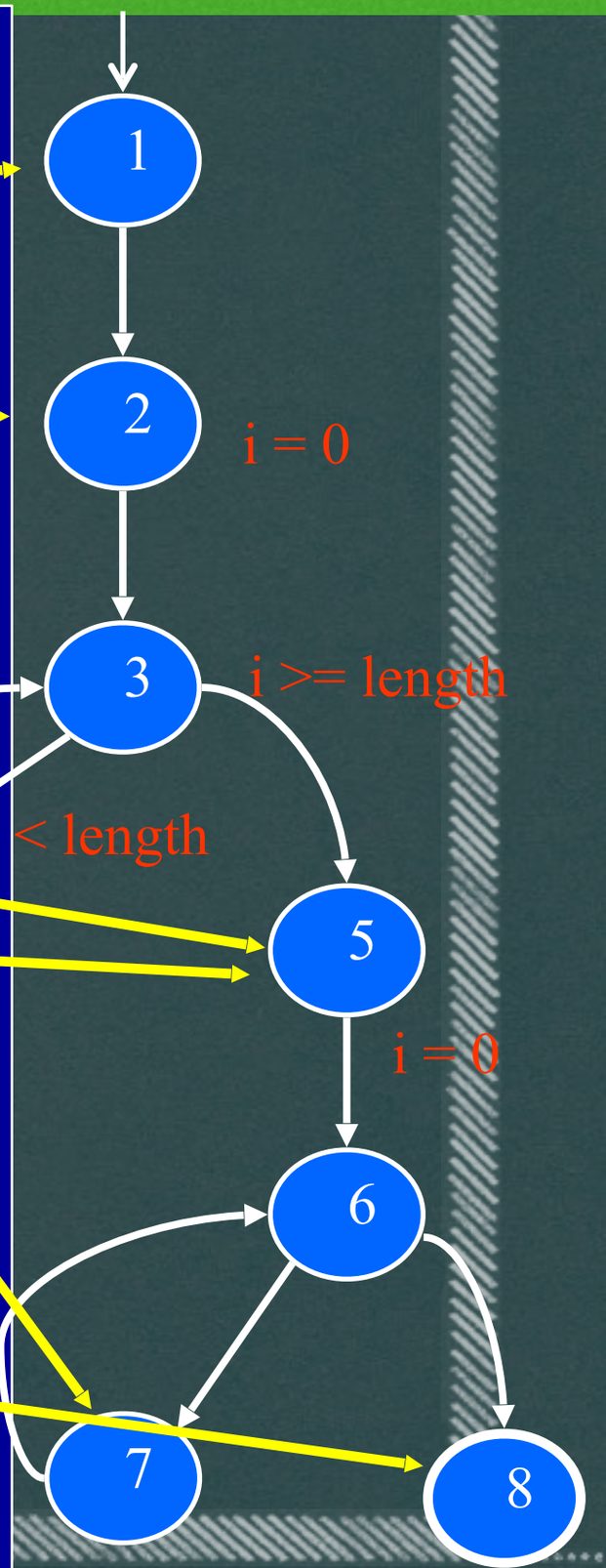
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



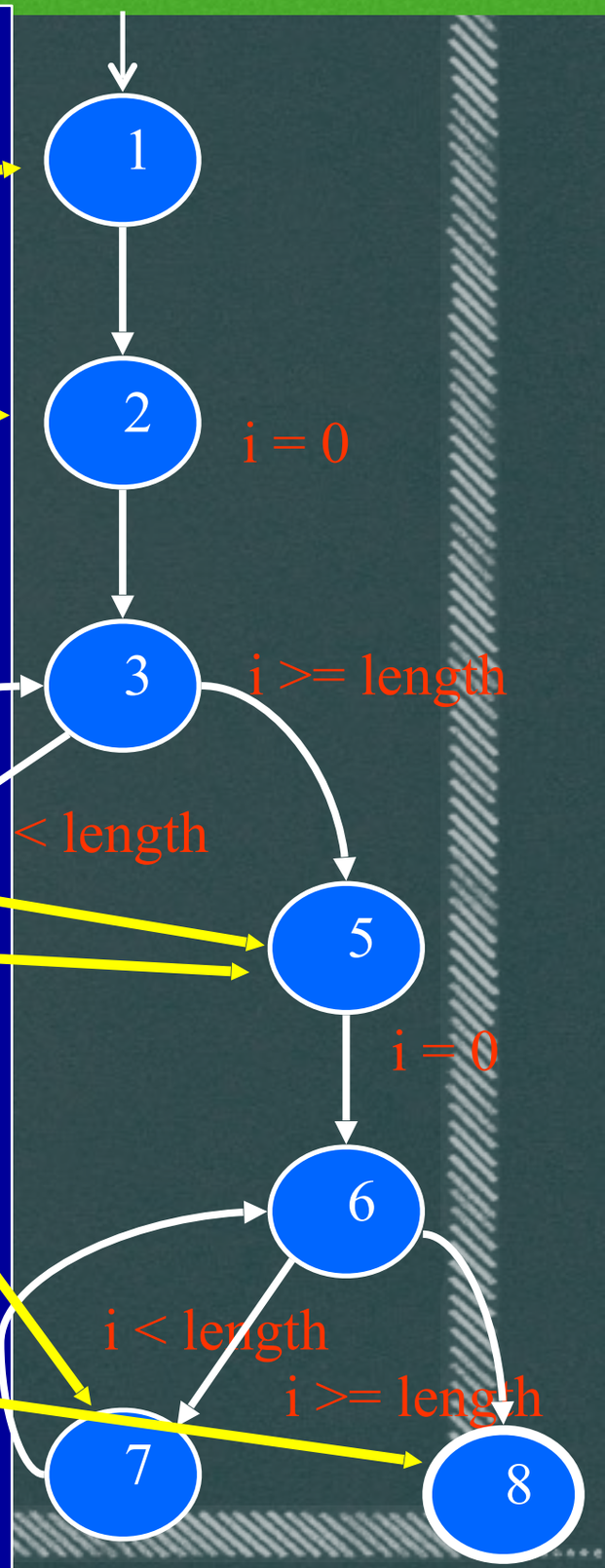
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```



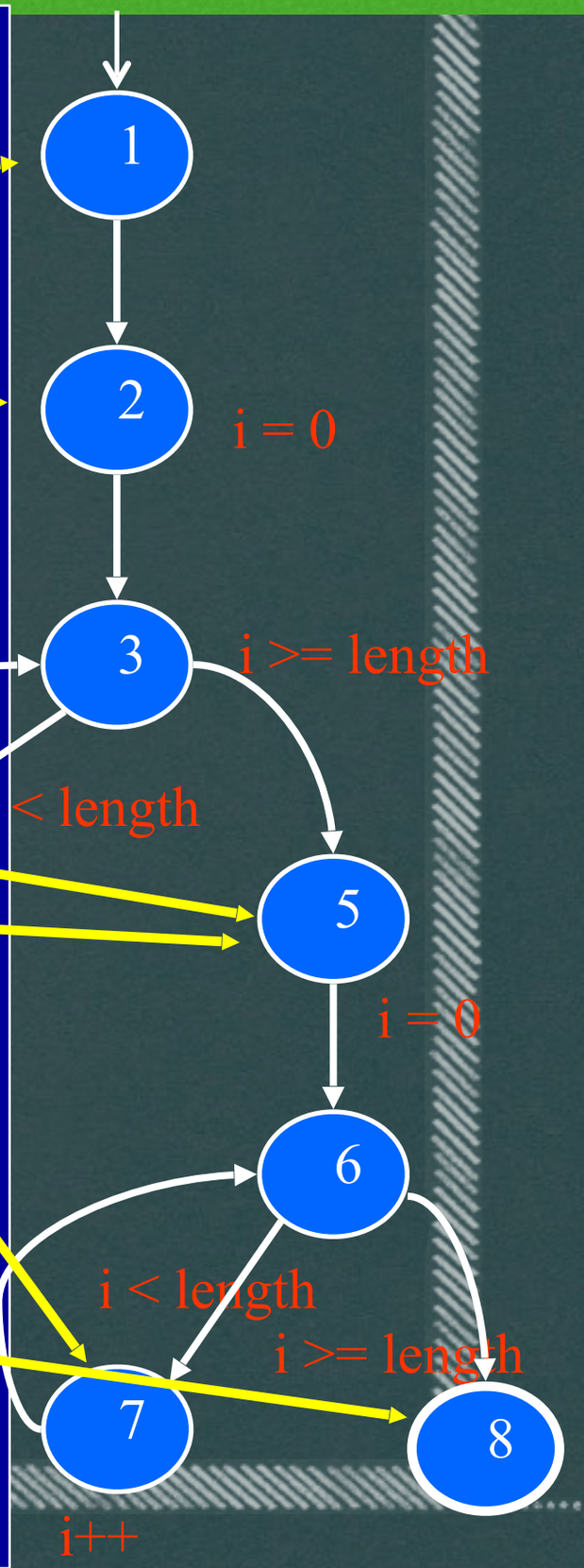
Control Flow Graph for Stats

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, 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 ("standard deviation: " + sd);
}
```

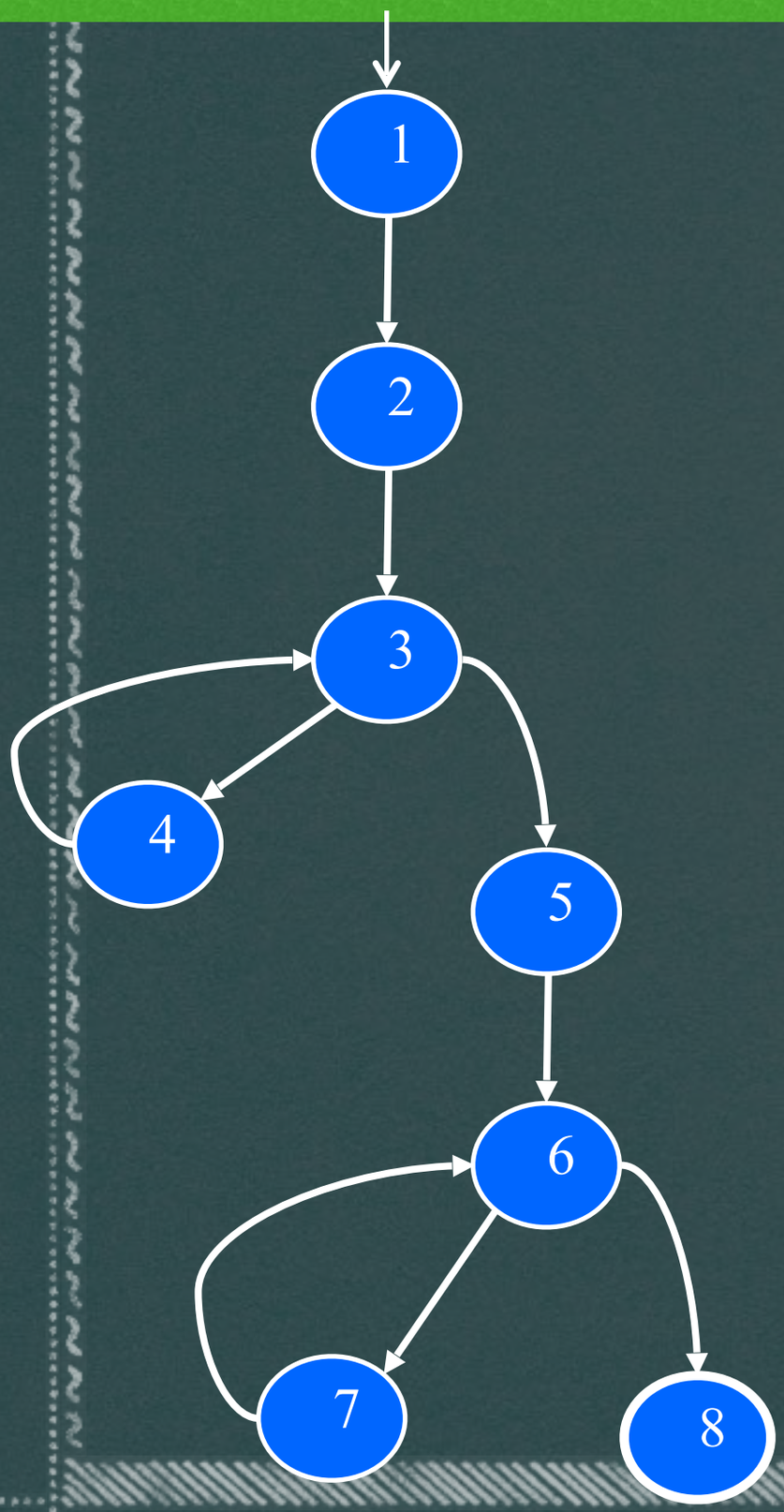


Control Flow Graph for Stats

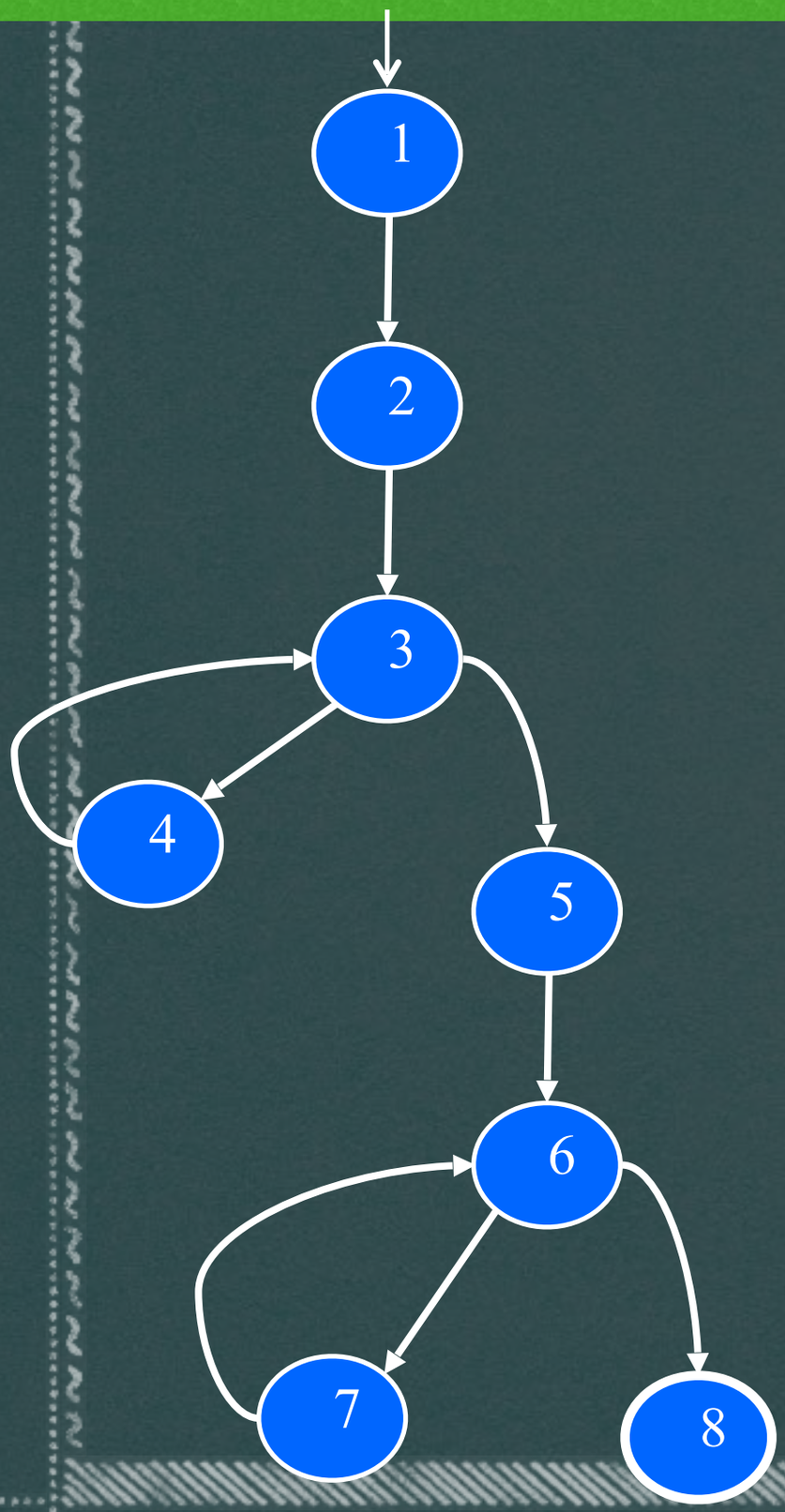
```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    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 ("standard deviation: " + sd);
}
```



Control Flow TRs and Test Paths – EC

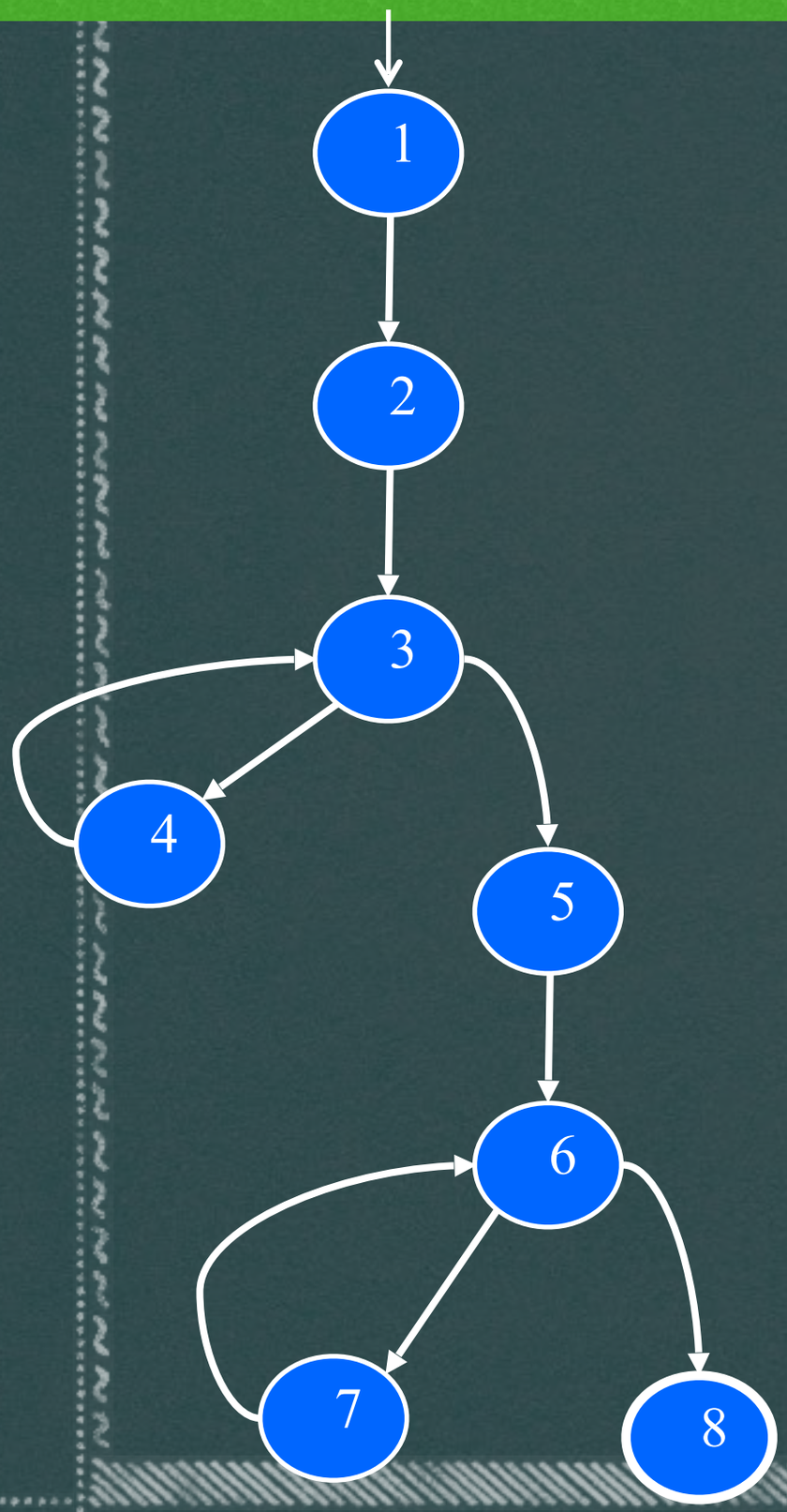


Control Flow TRs and Test Paths – EC

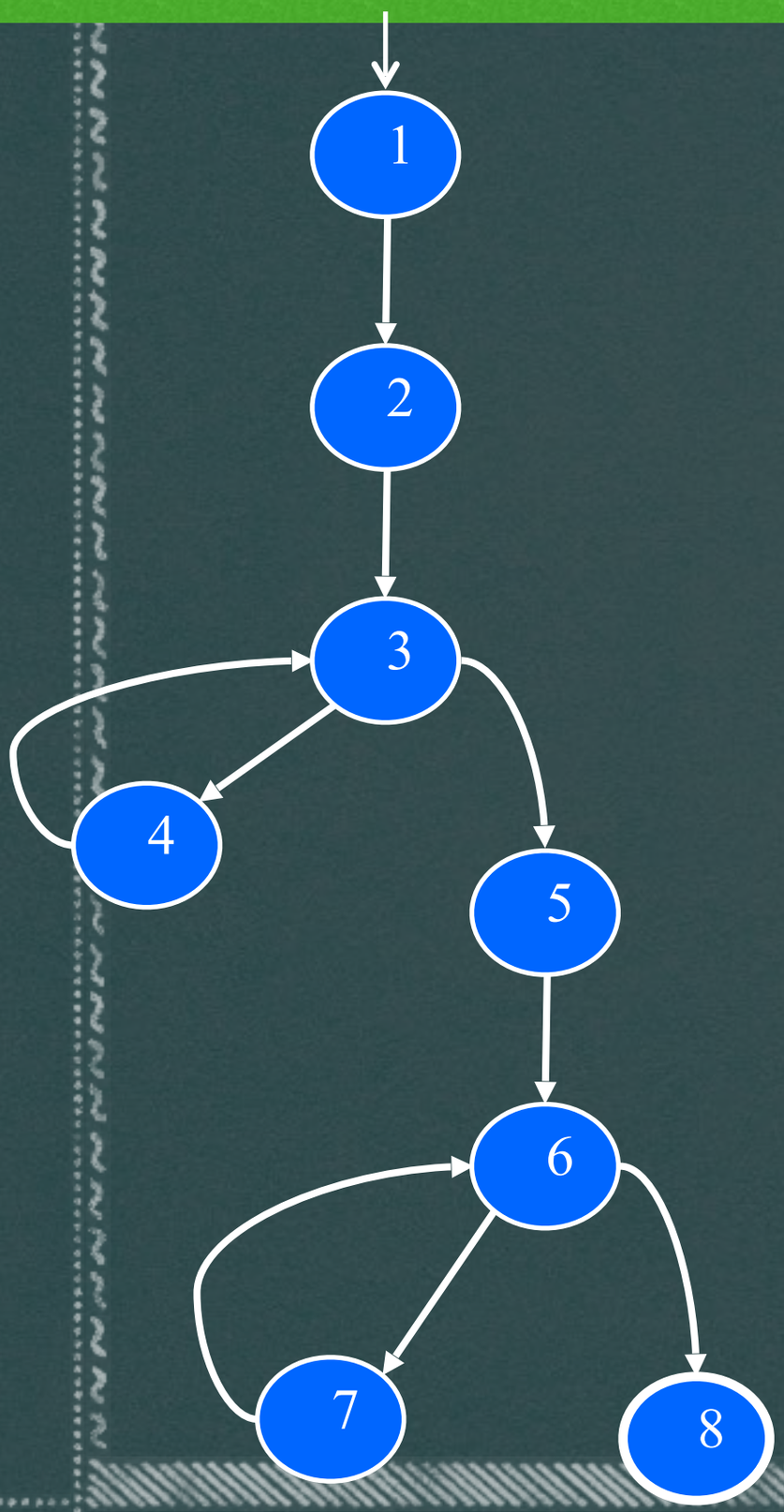


Edge Coverage	
TR	Test Path
A. [1, 2]	[1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
B. [2, 3]	
C. [3, 4]	
D. [3, 5]	
E. [4, 3]	
F. [5, 6]	
G. [6, 7]	
H. [6, 8]	
I. [7, 6]	

Control Flow TRs and Test Paths – EPC

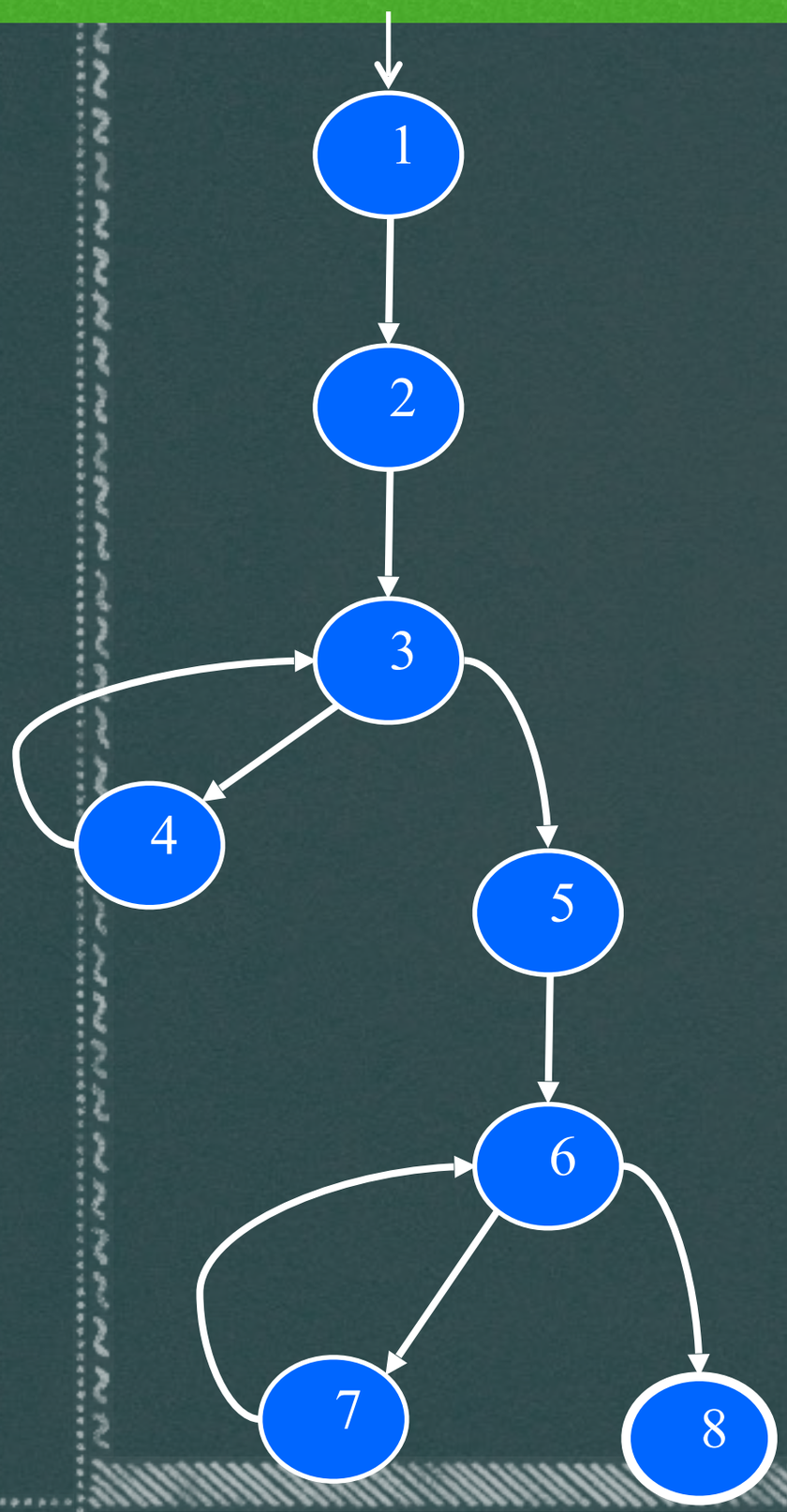


Control Flow TRs and Test Paths – EPC



Edge-Pair Coverage	
TR	**Test Paths**
A. [1, 2, 3]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
B. [2, 3, 4]	ii. [1, 2, 3, 5, 6, 8]
C. [2, 3, 5]	iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
D. [3, 4, 3]	
E. [3, 5, 6]	
F. [4, 3, 5]	
G. [5, 6, 7]	
H. [5, 6, 8]	
I. [6, 7, 6]	
J. [7, 6, 8]	
K. [4, 3, 4]	
L. [7, 6, 7]	

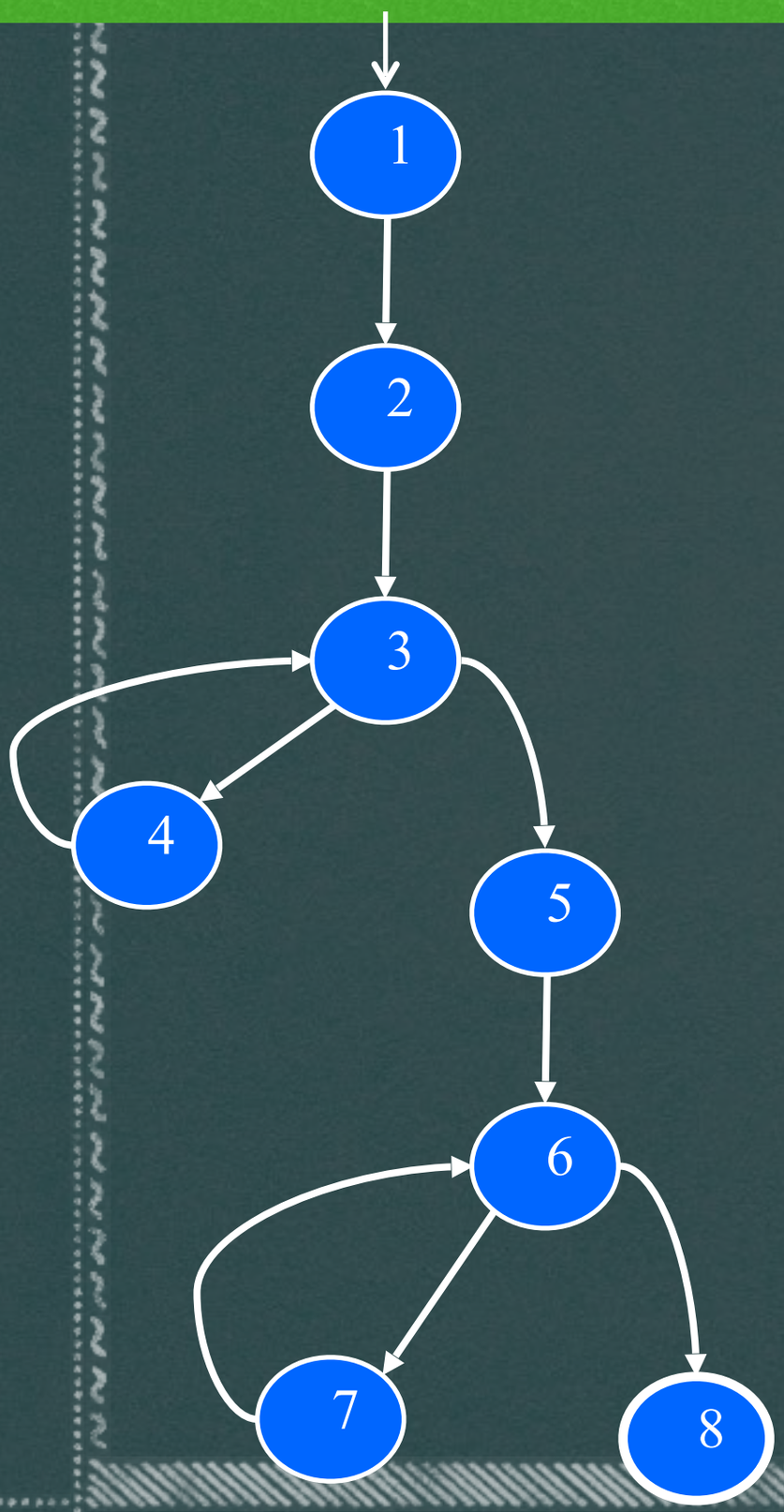
Control Flow TRs and Test Paths – EPC



Edge-Pair Coverage	
TR	Test Paths
A. [1, 2, 3] B. [2, 3, 4] C. [2, 3, 5] D. [3, 4, 3] E. [3, 5, 6] F. [4, 3, 5] G. [5, 6, 7] H. [5, 6, 8] I. [6, 7, 6] J. [7, 6, 8] K. [4, 3, 4] L. [7, 6, 7]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 5, 6, 8] iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]

TP	TRs toured	*sidetrips*

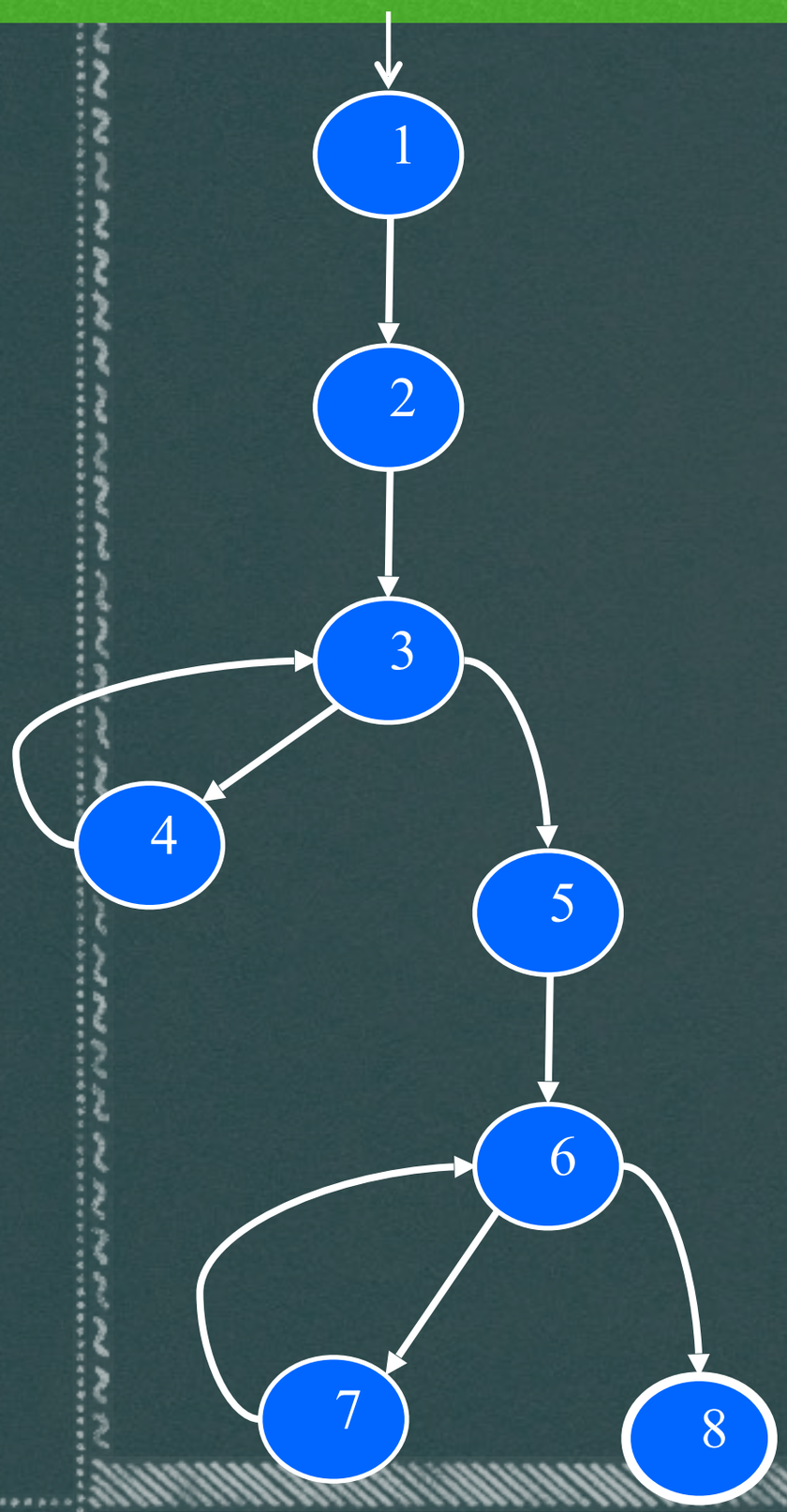
Control Flow TRs and Test Paths – EPC



Edge-Pair Coverage	
TR	Test Paths
A. [1, 2, 3] B. [2, 3, 4] C. [2, 3, 5] D. [3, 4, 3] E. [3, 5, 6] F. [4, 3, 5] G. [5, 6, 7] H. [5, 6, 8] I. [6, 7, 6] J. [7, 6, 8] K. [4, 3, 4] L. [7, 6, 7]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 5, 6, 8] iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]

TP	TRs toured	*sidetrips*
i	A, B, D, E, F, G, I, J	C, H

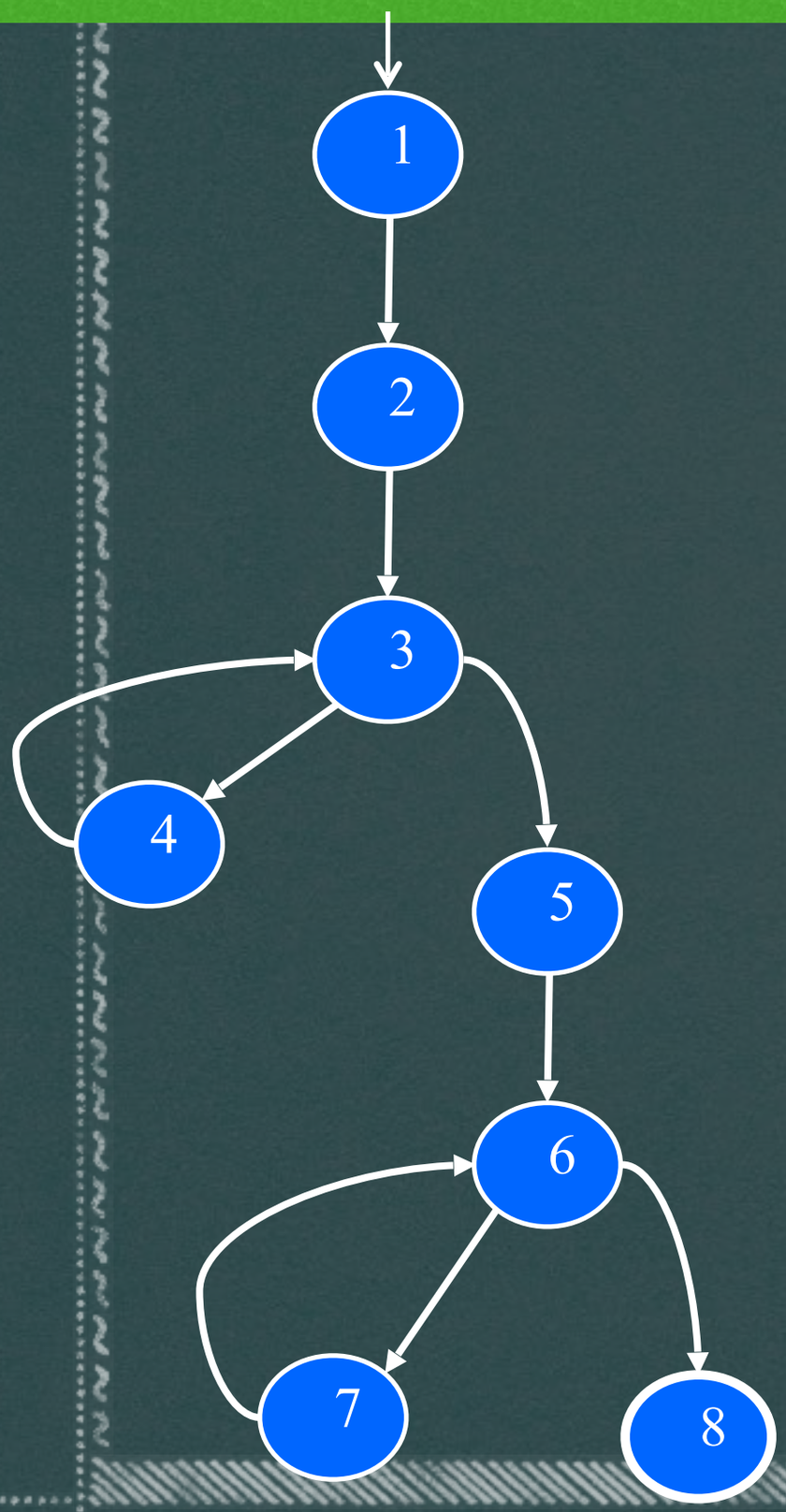
Control Flow TRs and Test Paths – EPC



Edge-Pair Coverage	
TR	Test Paths
A. [1, 2, 3] B. [2, 3, 4] C. [2, 3, 5] D. [3, 4, 3] E. [3, 5, 6] F. [4, 3, 5] G. [5, 6, 7] H. [5, 6, 8] I. [6, 7, 6] J. [7, 6, 8] K. [4, 3, 4] L. [7, 6, 7]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 5, 6, 8] iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]

TP	TRs toured	*sidetrips*
i	A, B, D, E, F, G, I, J	C, H
ii	A, C, E, H	

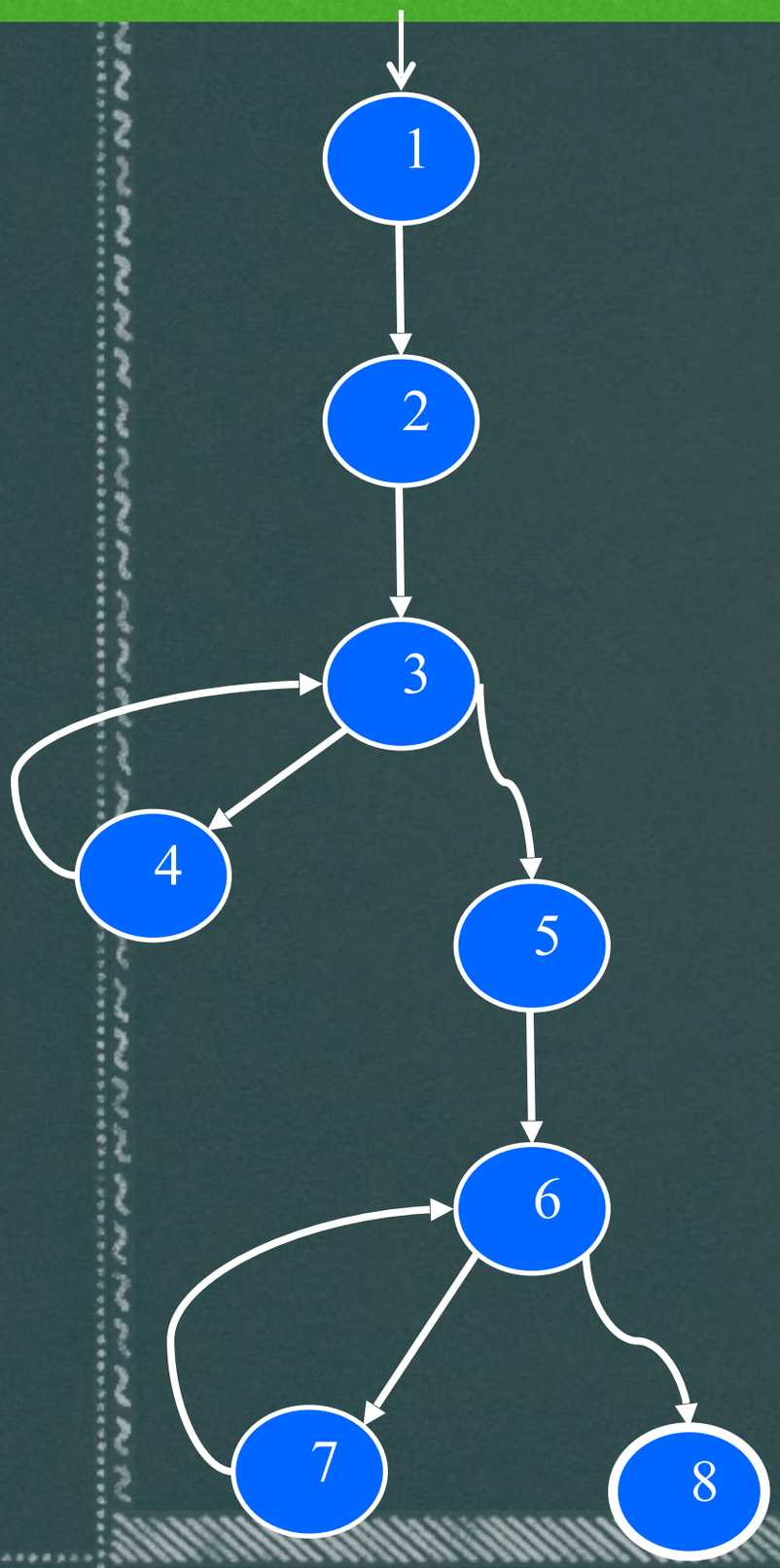
Control Flow TRs and Test Paths – EPC



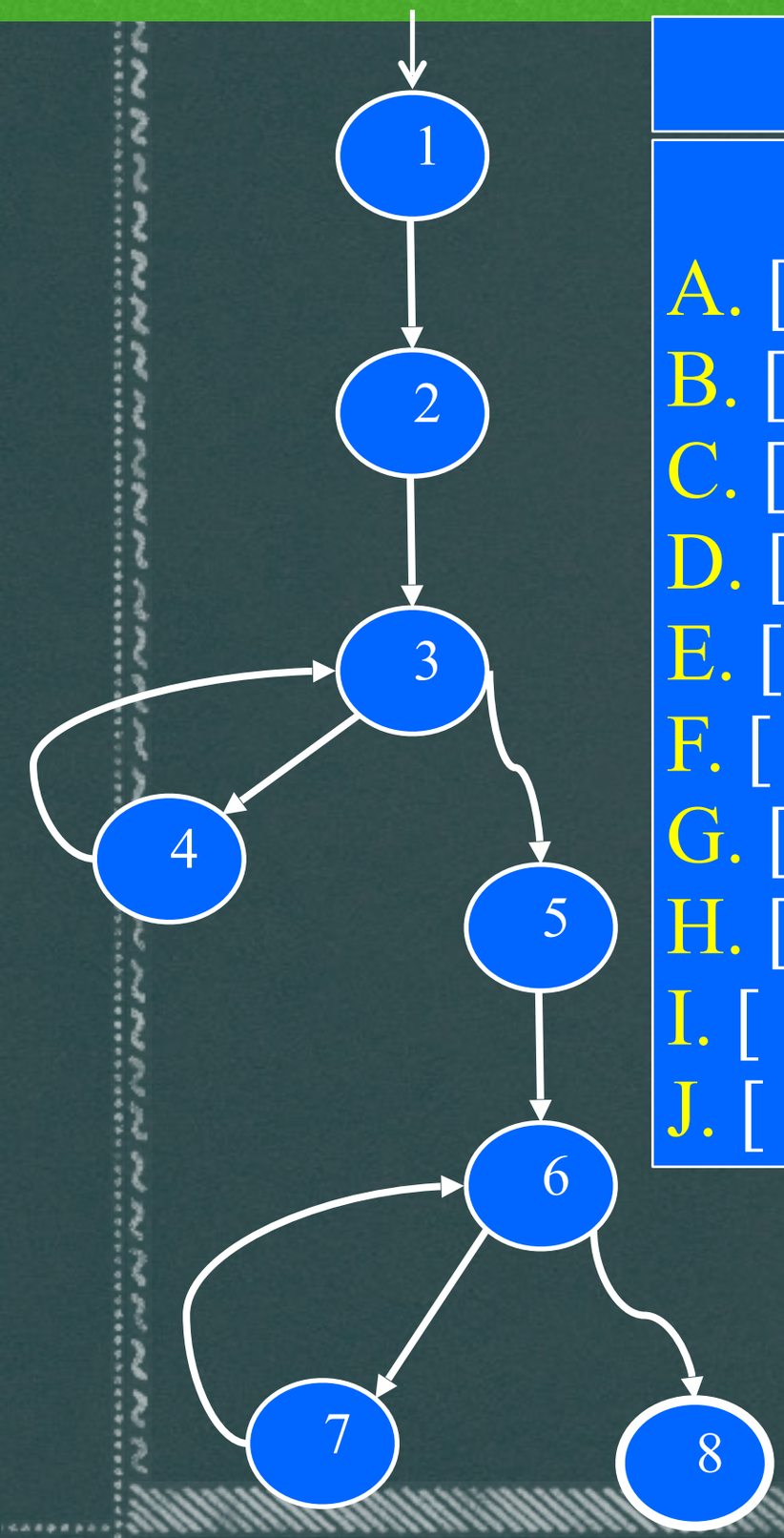
Edge-Pair Coverage	
TR	Test Paths
A. [1, 2, 3] B. [2, 3, 4] C. [2, 3, 5] D. [3, 4, 3] E. [3, 5, 6] F. [4, 3, 5] G. [5, 6, 7] H. [5, 6, 8] I. [6, 7, 6] J. [7, 6, 8] K. [4, 3, 4] L. [7, 6, 7]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 5, 6, 8] iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]

TP	TRs toured	*sidetrips*
i	A, B, D, E, F, G, I, J	C, H
ii	A, C, E, H	
iii	A, B, D, E, F, G, I, J, K, L	C, H

Control Flow TRs and Test Paths – PPC



Control Flow TRs and Test Paths – PPC



Prime Path Coverage

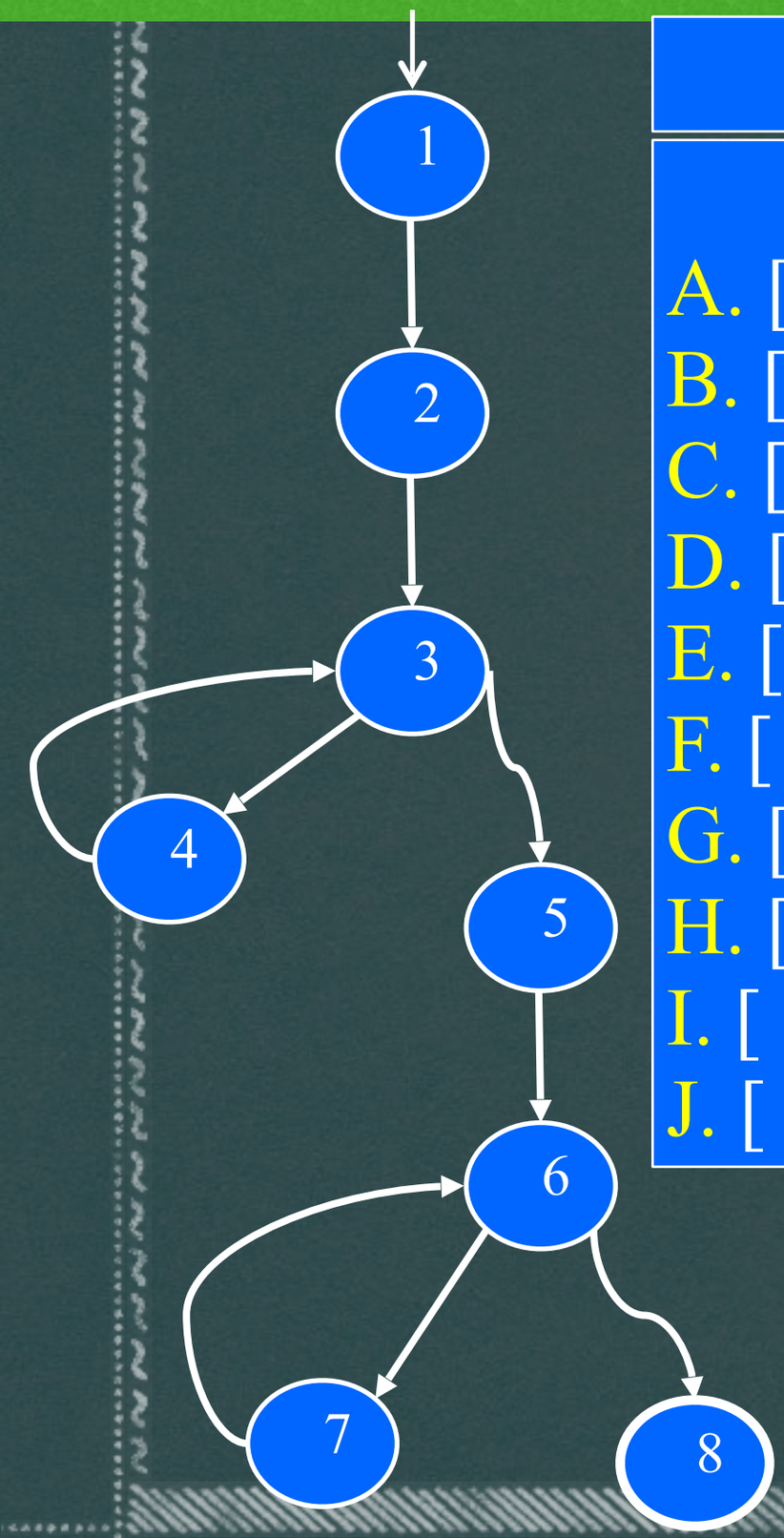
TR

A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

Test Paths

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

Control Flow TRs and Test Paths – PPC

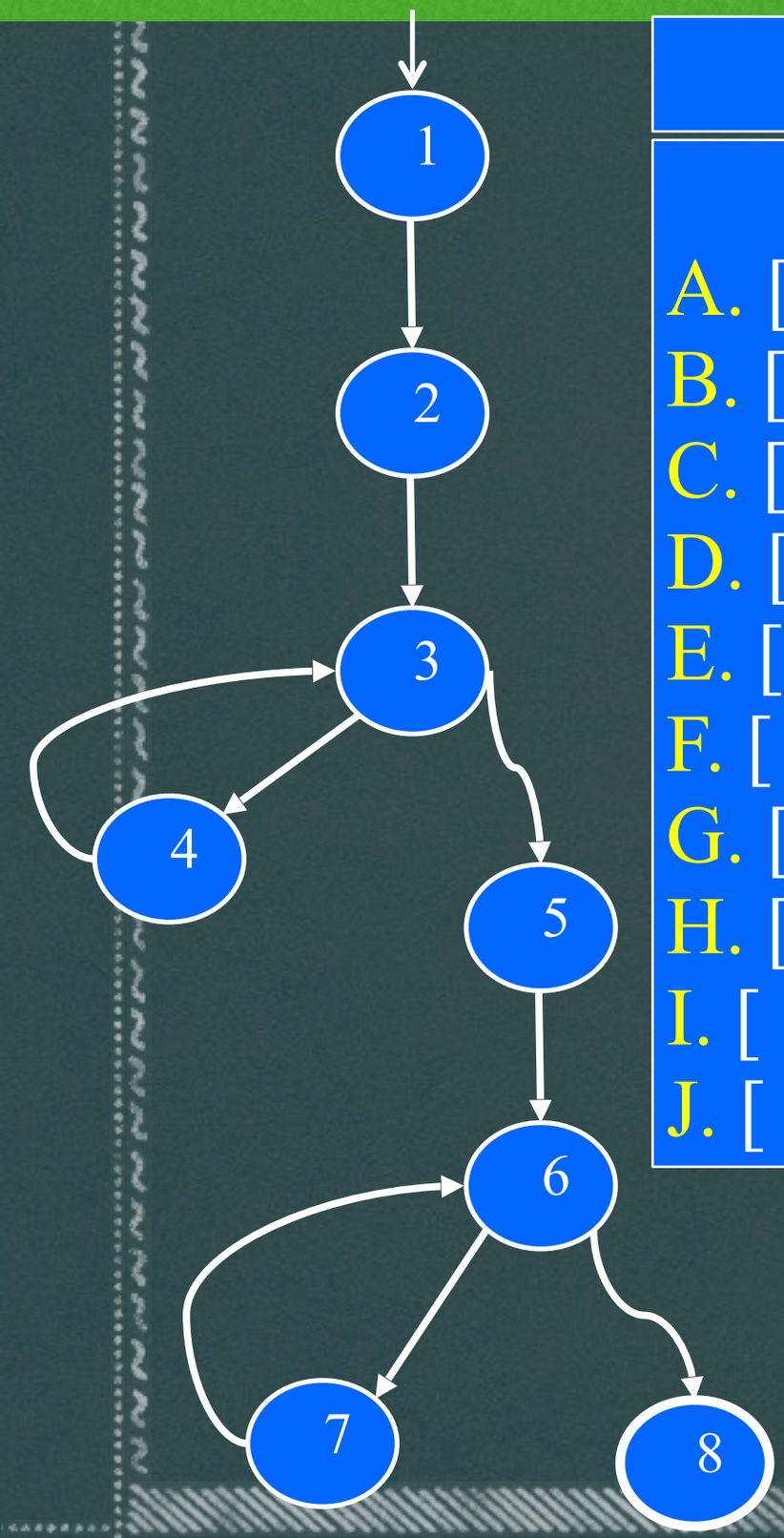


Prime Path Coverage

TR	Test Paths
A. [3, 4, 3]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
B. [4, 3, 4]	ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
C. [7, 6, 7]	iii. [1, 2, 3, 4, 3, 5, 6, 8]
D. [7, 6, 8]	iv. [1, 2, 3, 5, 6, 7, 6, 8]
E. [6, 7, 6]	v. [1, 2, 3, 5, 6, 8]
F. [1, 2, 3, 4]	
G. [4, 3, 5, 6, 7]	
H. [4, 3, 5, 6, 8]	
I. [1, 2, 3, 5, 6, 7]	
J. [1, 2, 3, 5, 6, 8]	

TP	TRs toured	*sidetrips*

Control Flow TRs and Test Paths – PPC



Prime Path Coverage

TR

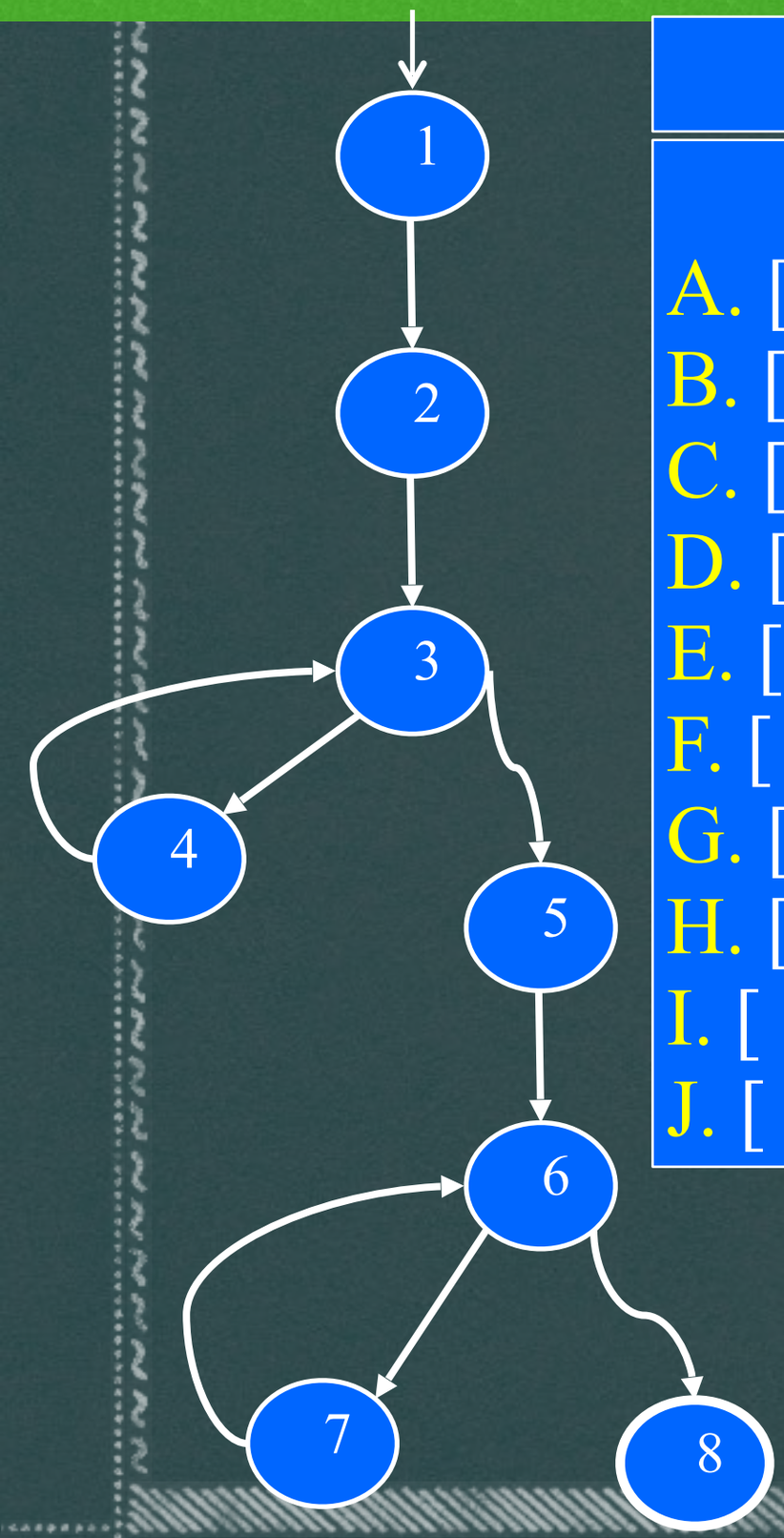
A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

Test Paths

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

TP	TRs toured	*sidetrips*
i	A, D, E, F, G	H, I, J

Control Flow TRs and Test Paths – PPC



Prime Path Coverage

TR

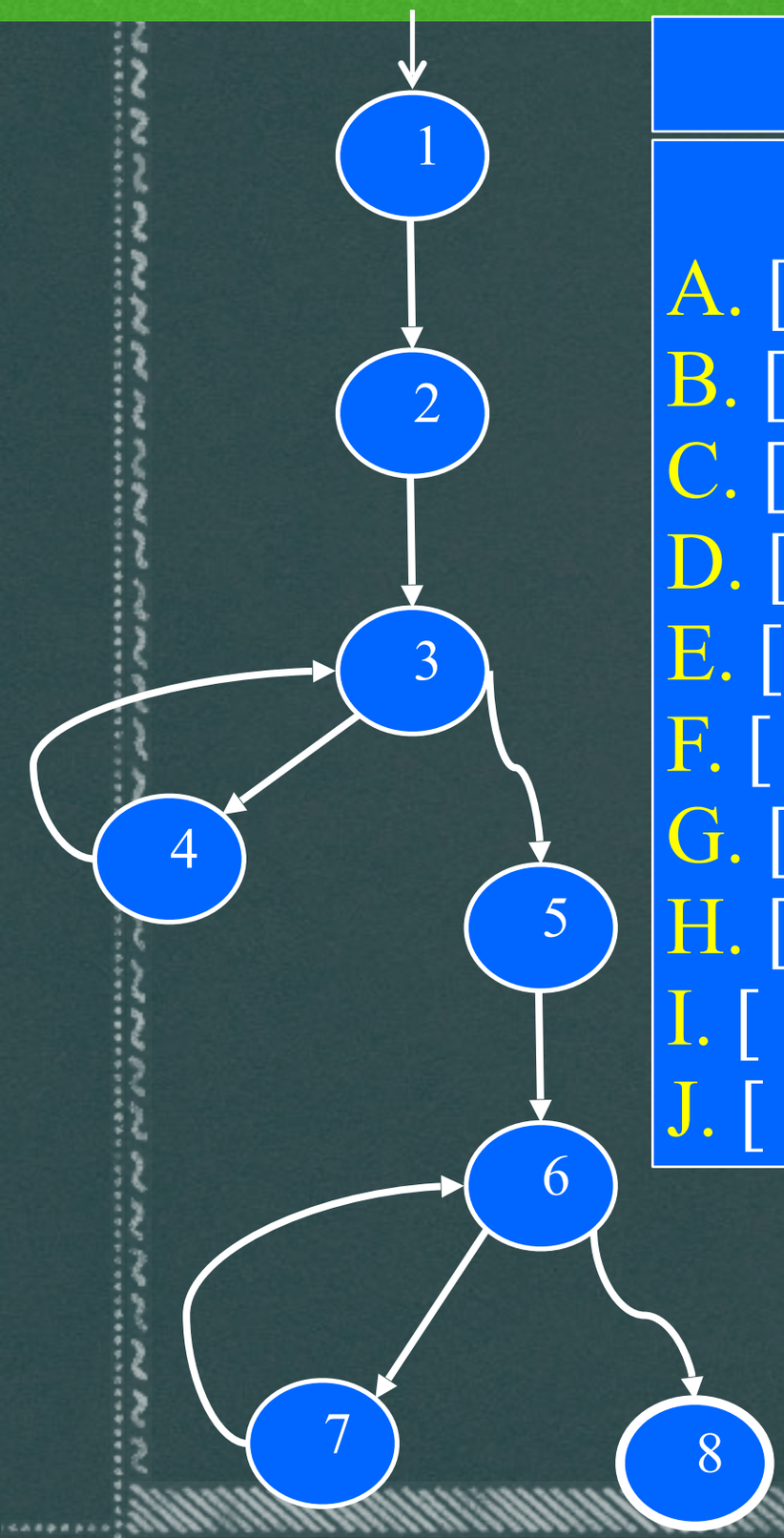
A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

Test Paths

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

TP	TRs toured	*sidetrips*
i	A, D, E, F, G	H, I, J
ii	A, B, C, D, E, F, G,	H, I, J

Control Flow TRs and Test Paths – PPC



Prime Path Coverage

TR

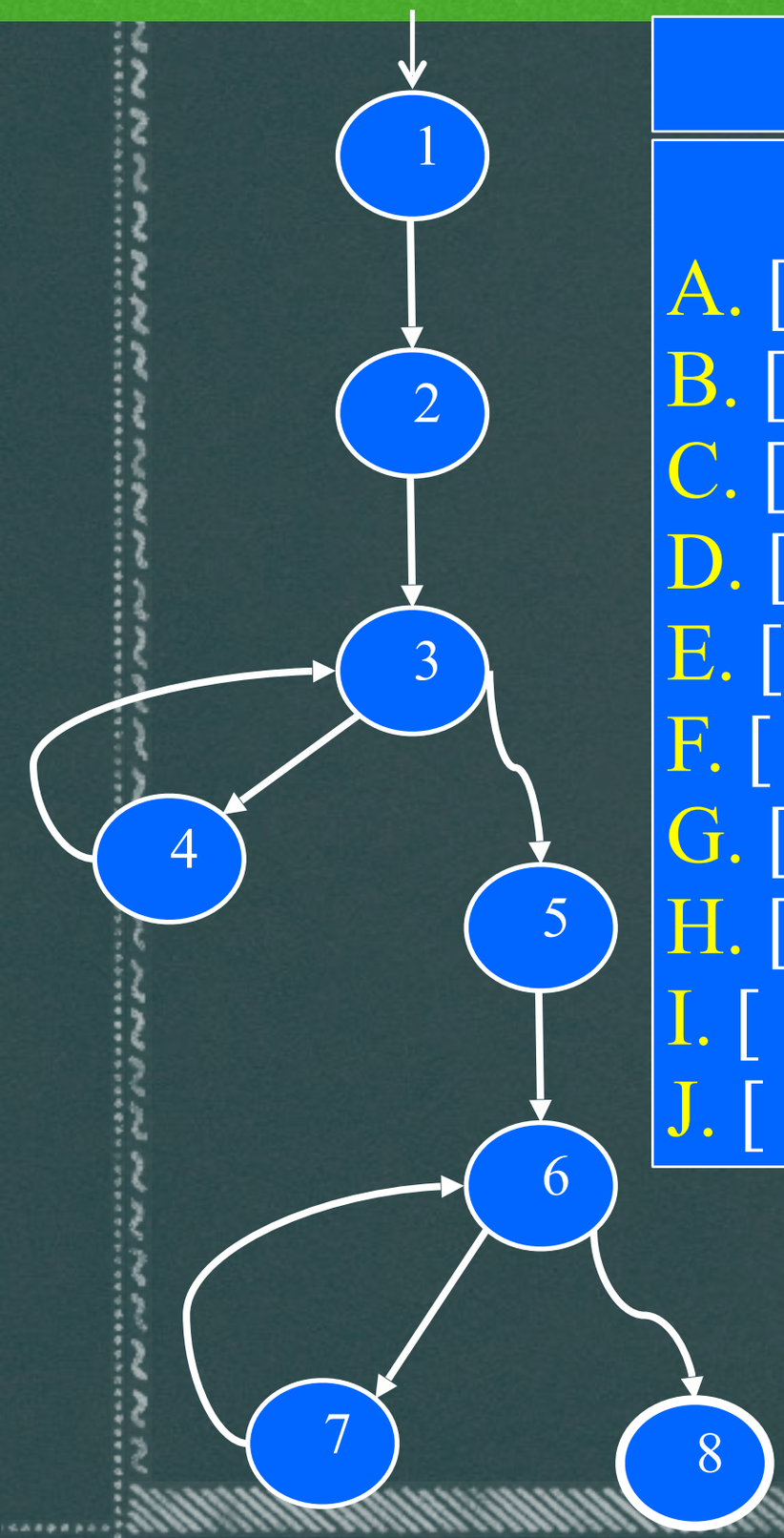
A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

Test Paths

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

TP	TRs toured	*sidetrips*
i	A, D, E, F, G	H, I, J
ii	A, B, C, D, E, F, G,	H, I, J
iii	A, F, H	J

Control Flow TRs and Test Paths – PPC



Prime Path Coverage

TR

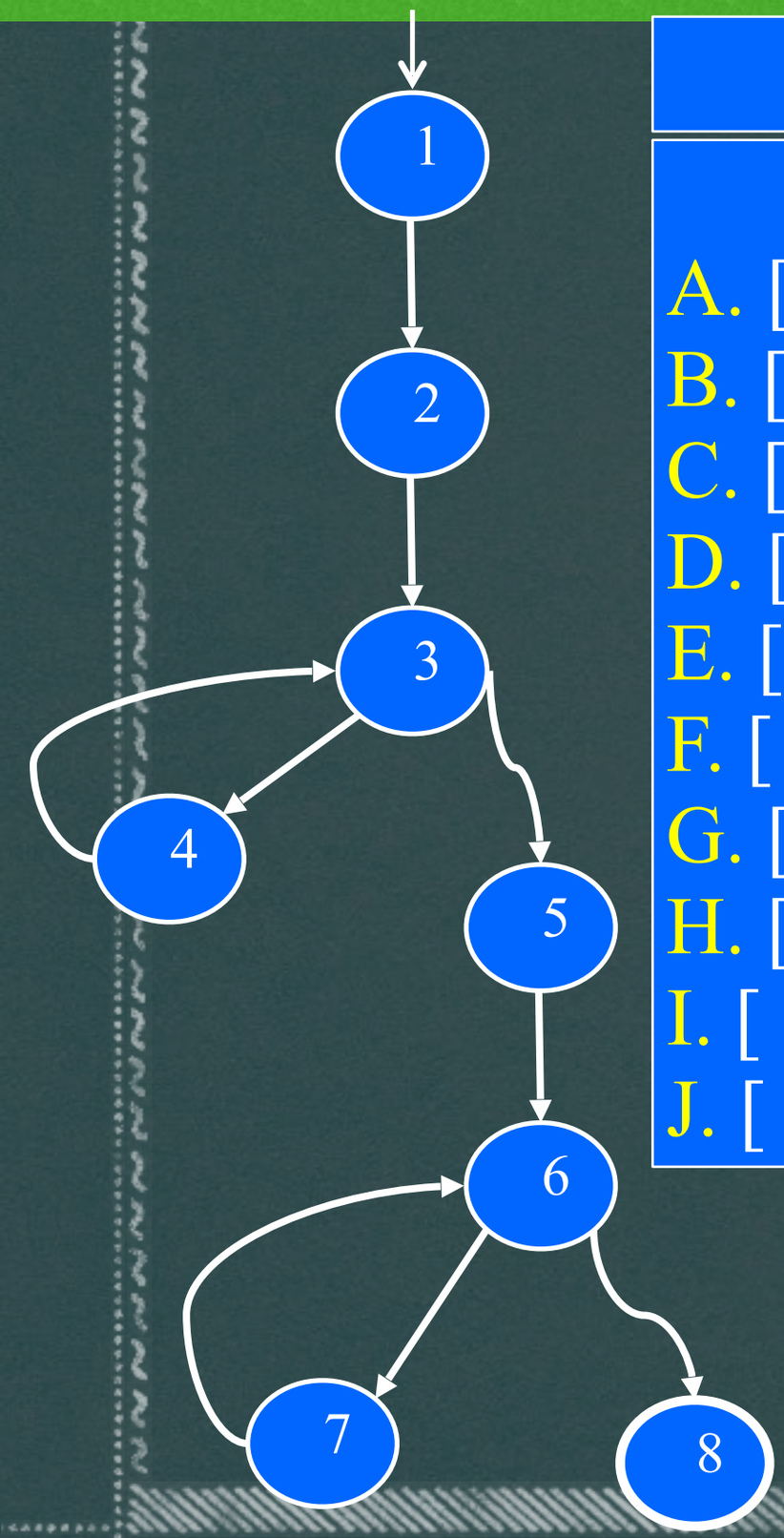
A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

Test Paths

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

TP	TRs toured	*sidetrips*
i	A, D, E, F, G	H, I, J
ii	A, B, C, D, E, F, G,	H, I, J
iii	A, F, H	J
iv	D, E, I	J

Control Flow TRs and Test Paths – PPC



Prime Path Coverage

TR

A. [3, 4, 3]
B. [4, 3, 4]
C. [7, 6, 7]
D. [7, 6, 8]
E. [6, 7, 6]
F. [1, 2, 3, 4]
G. [4, 3, 5, 6, 7]
H. [4, 3, 5, 6, 8]
I. [1, 2, 3, 5, 6, 7]
J. [1, 2, 3, 5, 6, 8]

Test Paths

i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]
iii. [1, 2, 3, 4, 3, 5, 6, 8]
iv. [1, 2, 3, 5, 6, 7, 6, 8]
v. [1, 2, 3, 5, 6, 8]

TP	TRs toured	*sidetrips*
i	A, D, E, F, G	H, I, J
ii	A, B, C, D, E, F, G,	H, I, J
iii	A, F, H	J
iv	D, E, I	J
v	J	

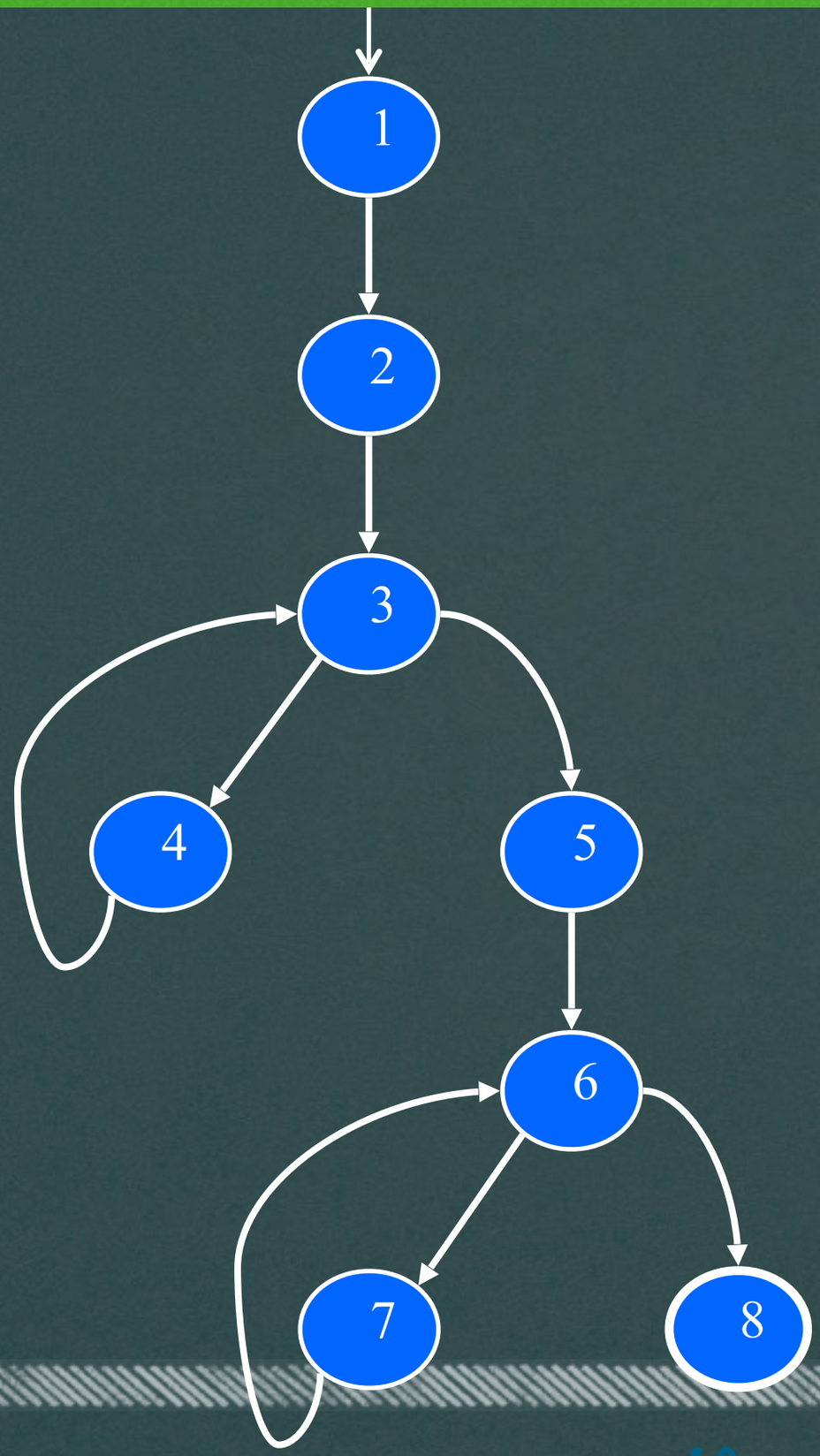
Data Flow Coverage for Source

- **def** : a location where a value is stored into memory
  - x appears on the left side of an assignment (x = 44;)
  - x is an actual parameter in a call and the method changes its value
  - x is a formal parameter of a method (implicit def when method starts)
  - x is an input to a program
- **use** : a location where variable's value is accessed
  - x appears on the right side of an assignment
  - x appears in a conditional test
  - x is an actual parameter to a method
  - x is an output of the program
  - x is an output of a method in a return statement
- If a def and a use appear on the same node, then it is only a DU-pair if the def occurs after the use and the node is in a loop

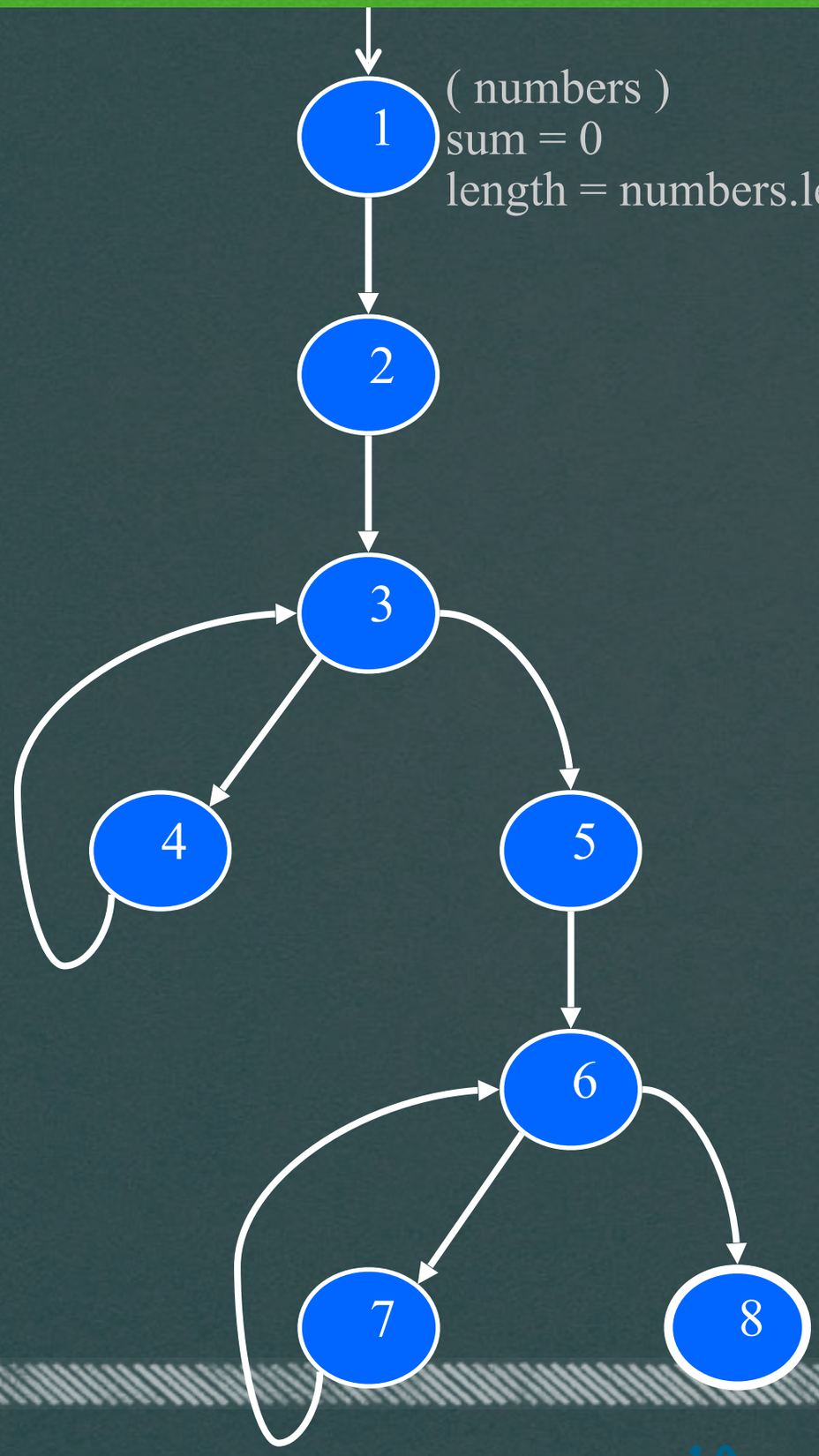
Example Data Flow – Stats

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

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

    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 ("standard deviation: " + sd);
}
```

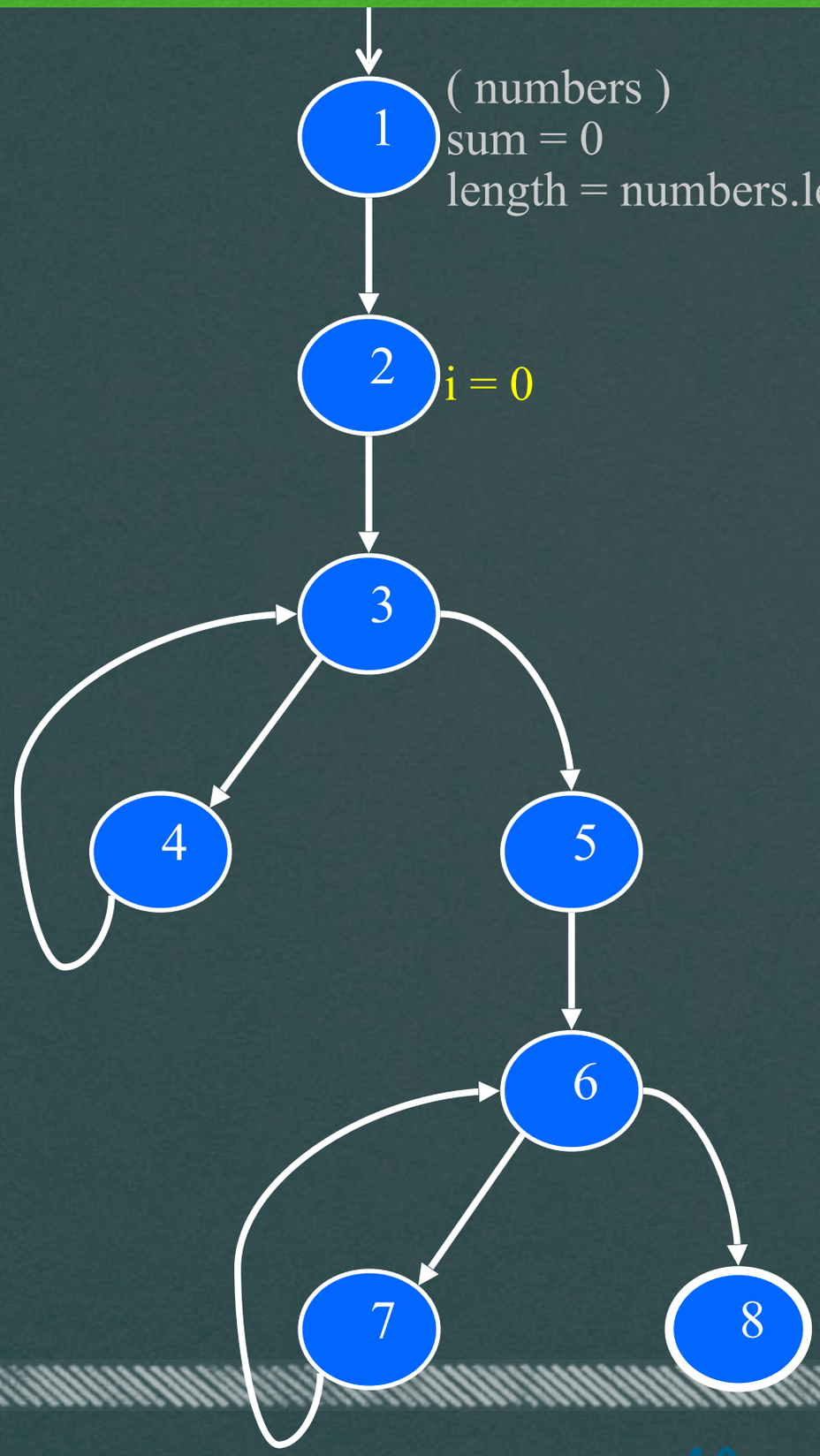
Control Flow Graph for Stats



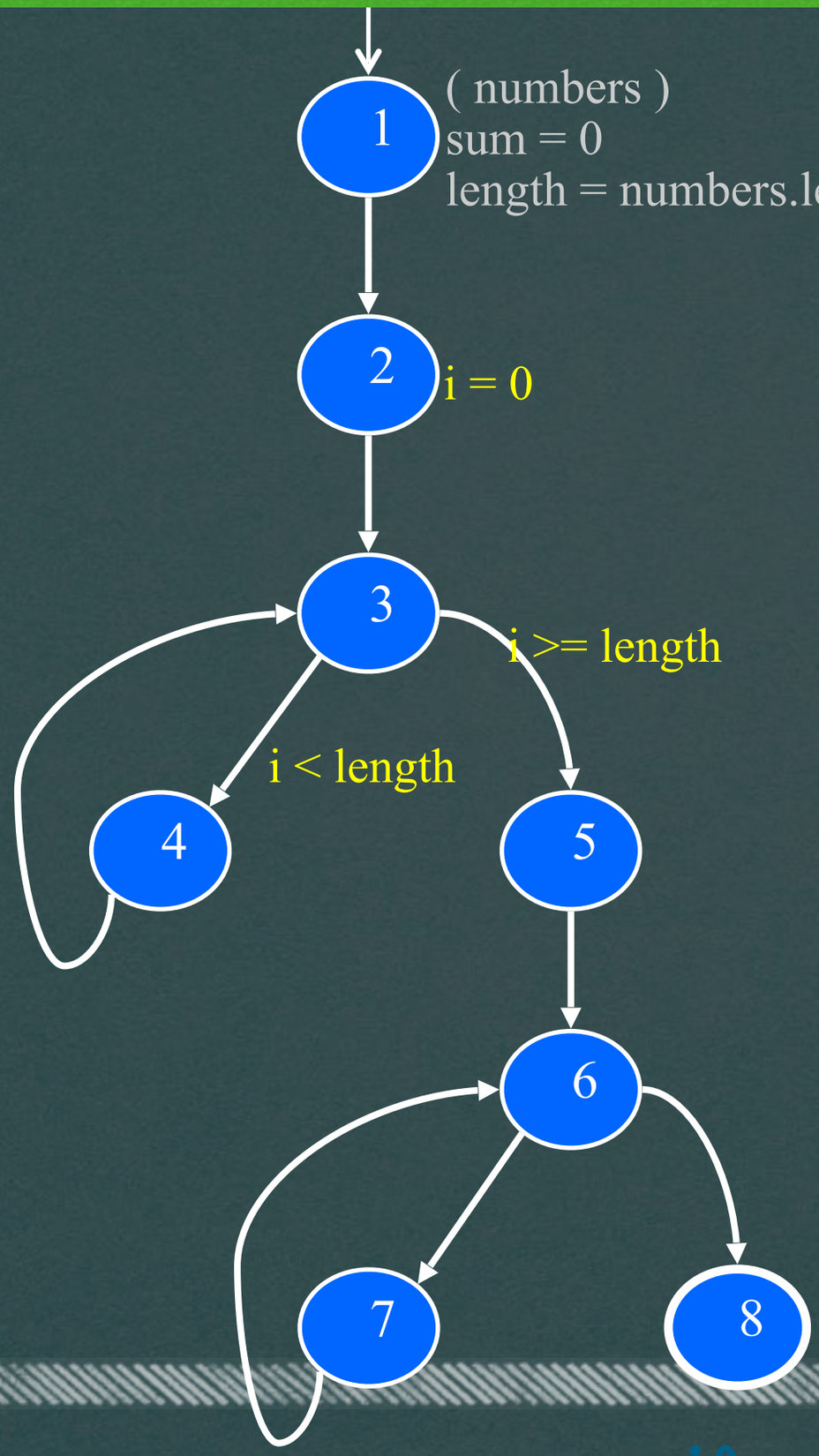
Control Flow Graph for Stats



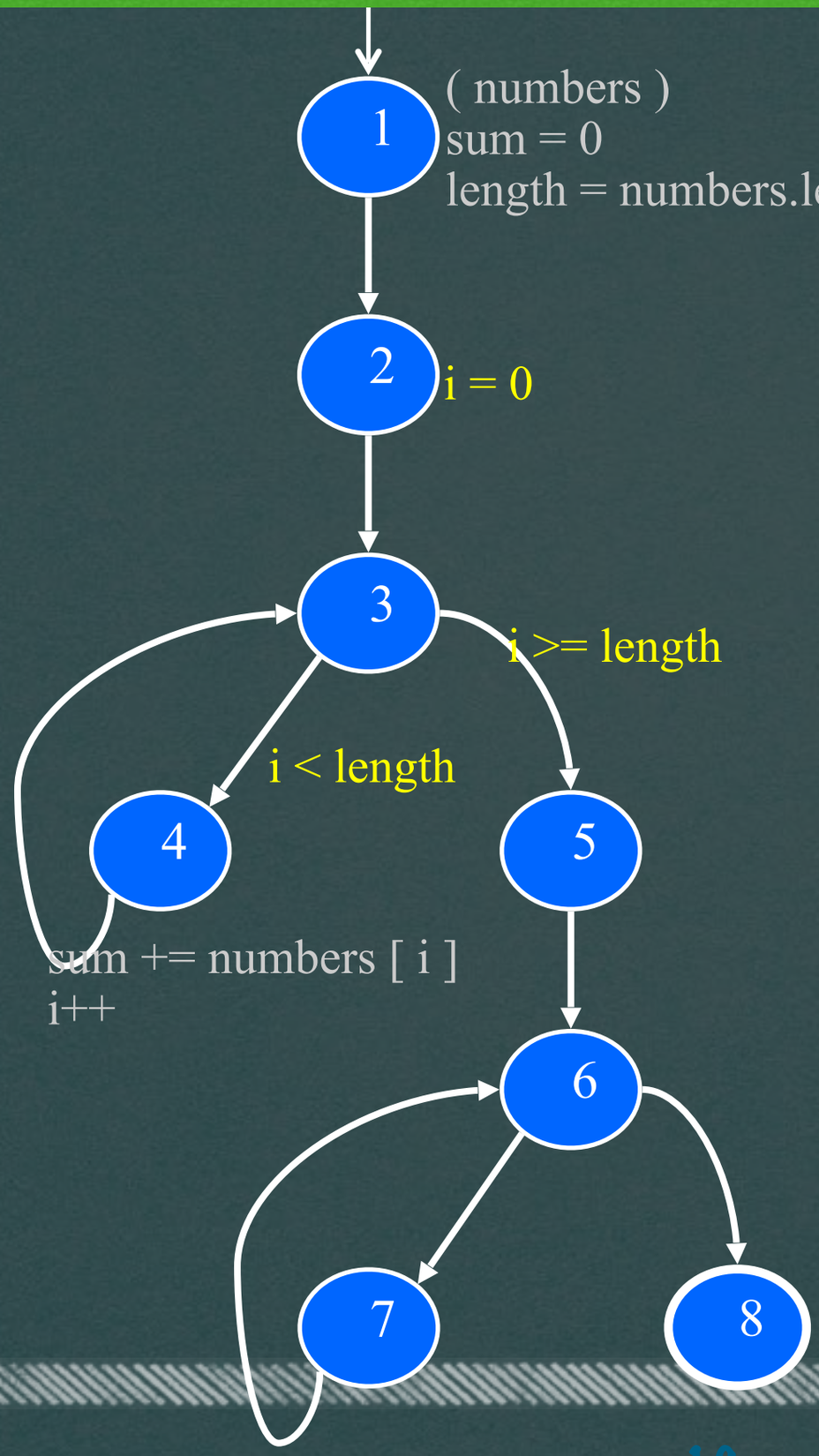
Control Flow Graph for Stats



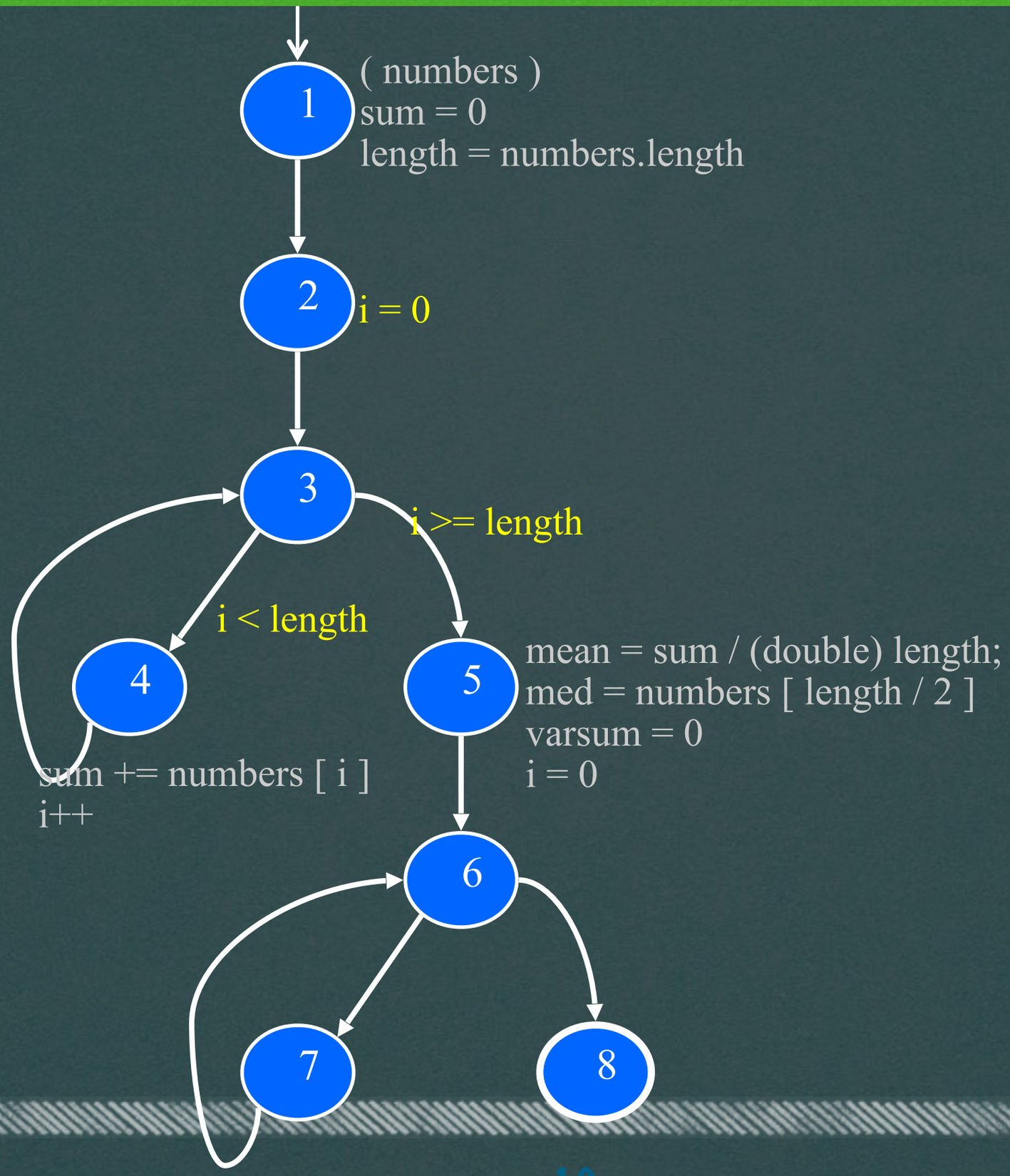
Control Flow Graph for Stats



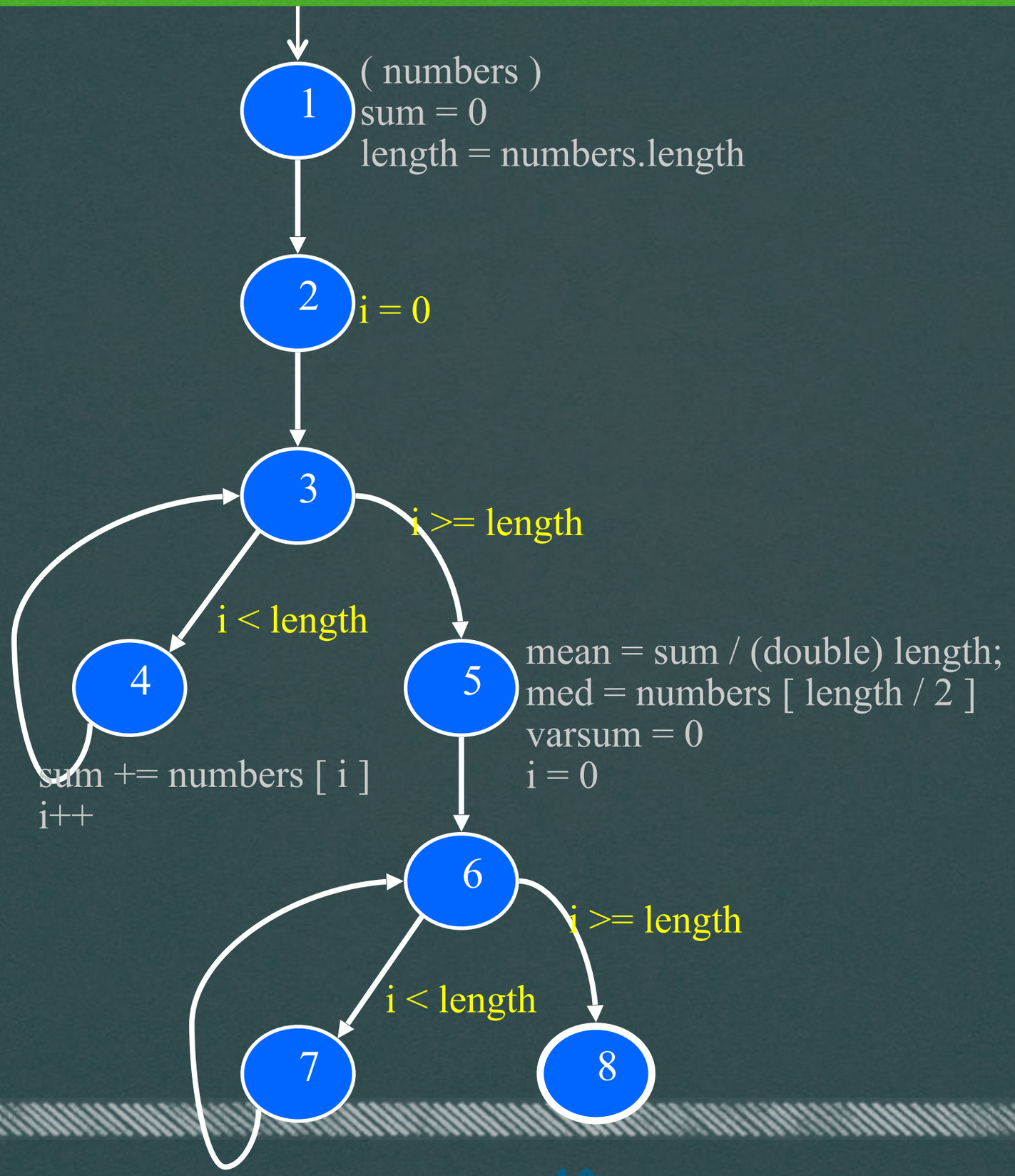
Control Flow Graph for Stats



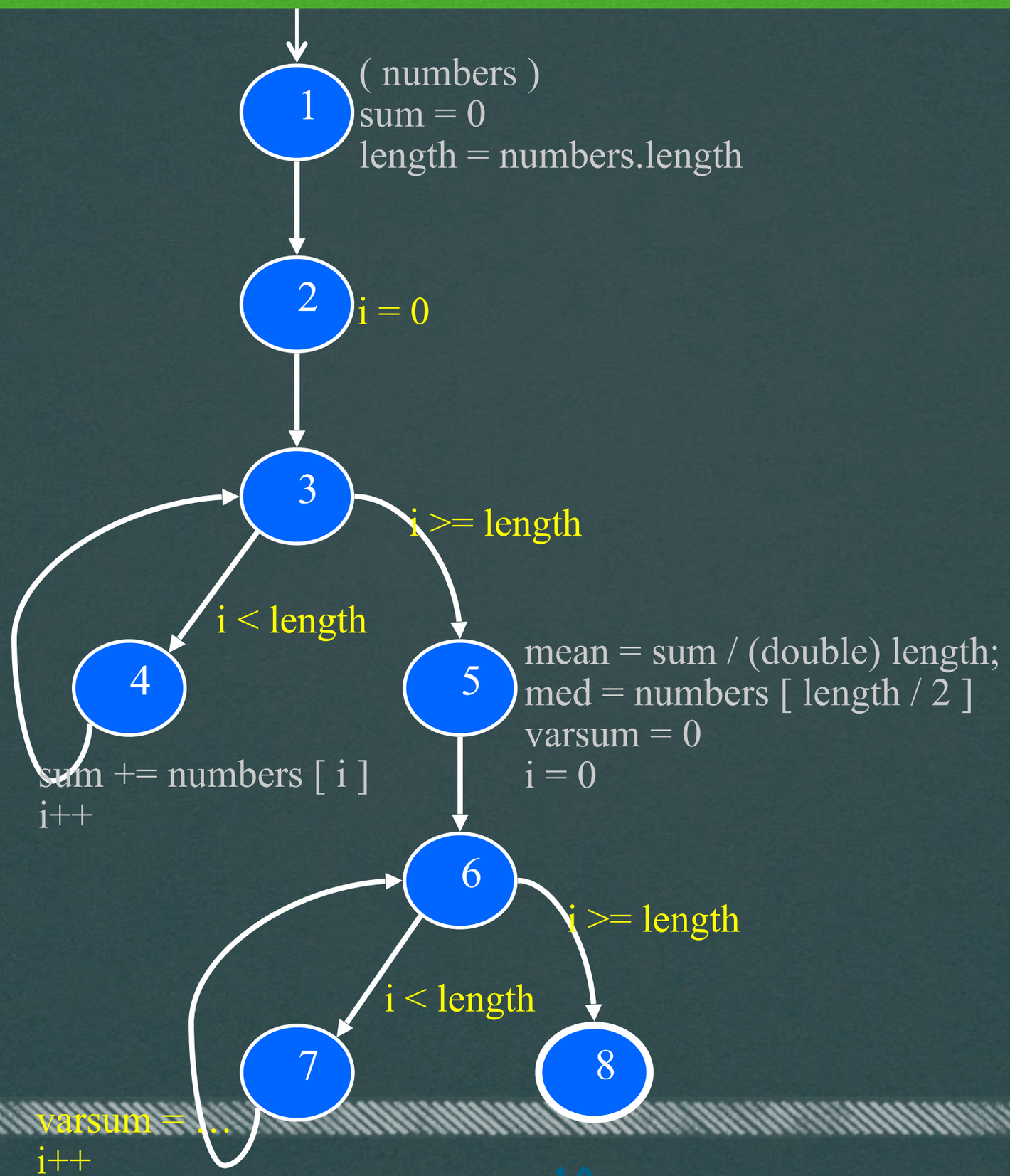
Control Flow Graph for Stats



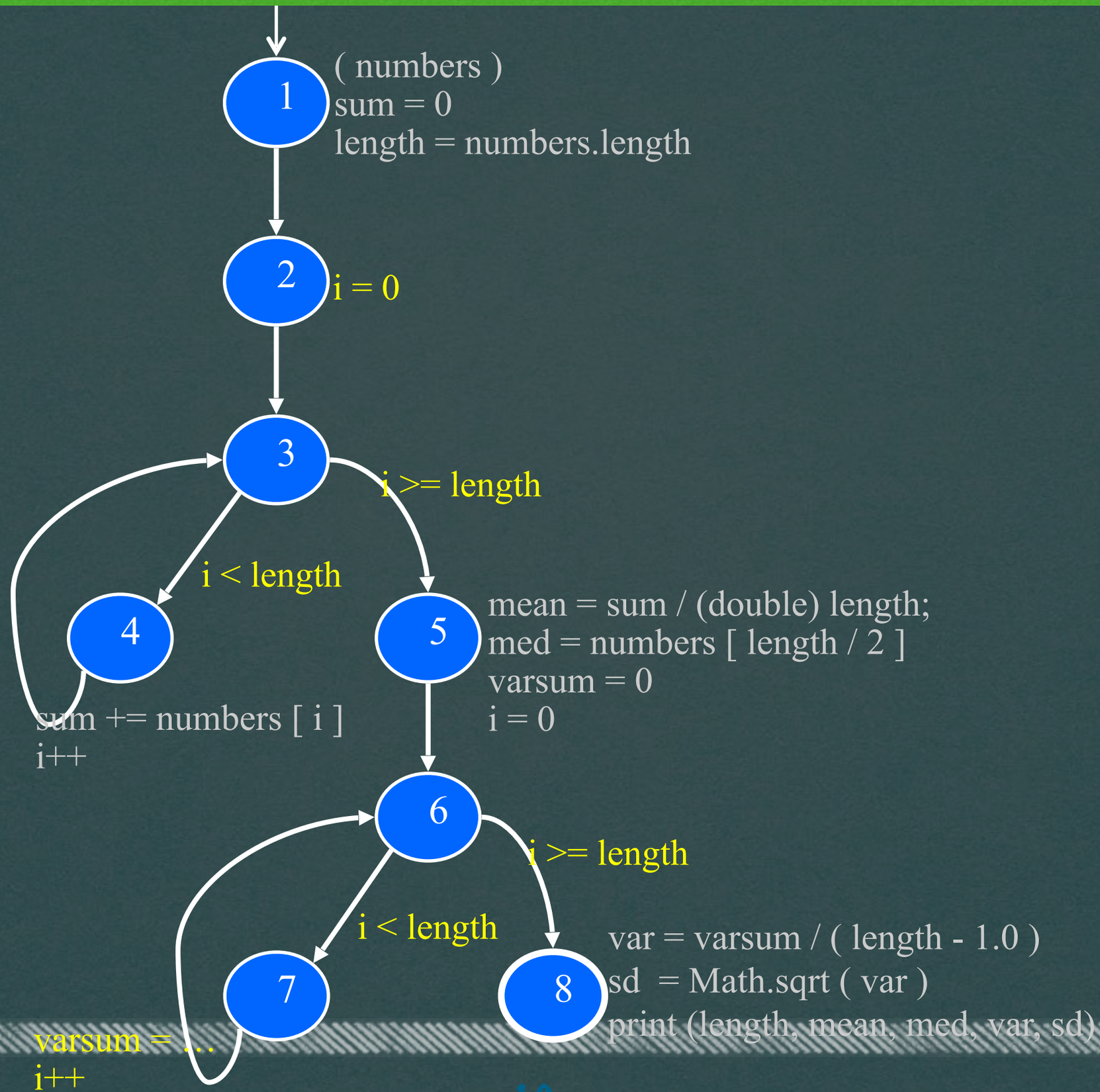
Control Flow Graph for Stats



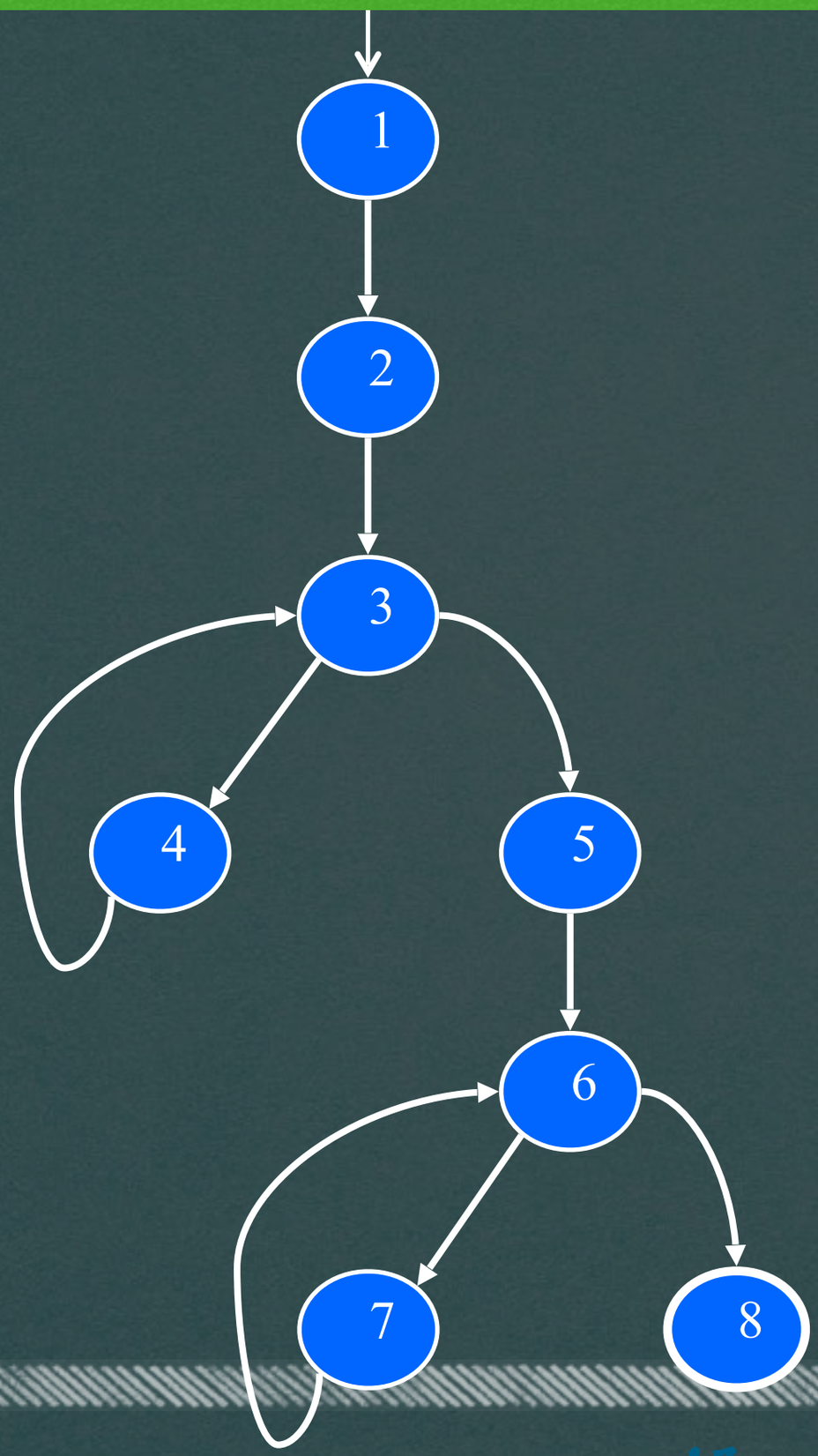
Control Flow Graph for Stats



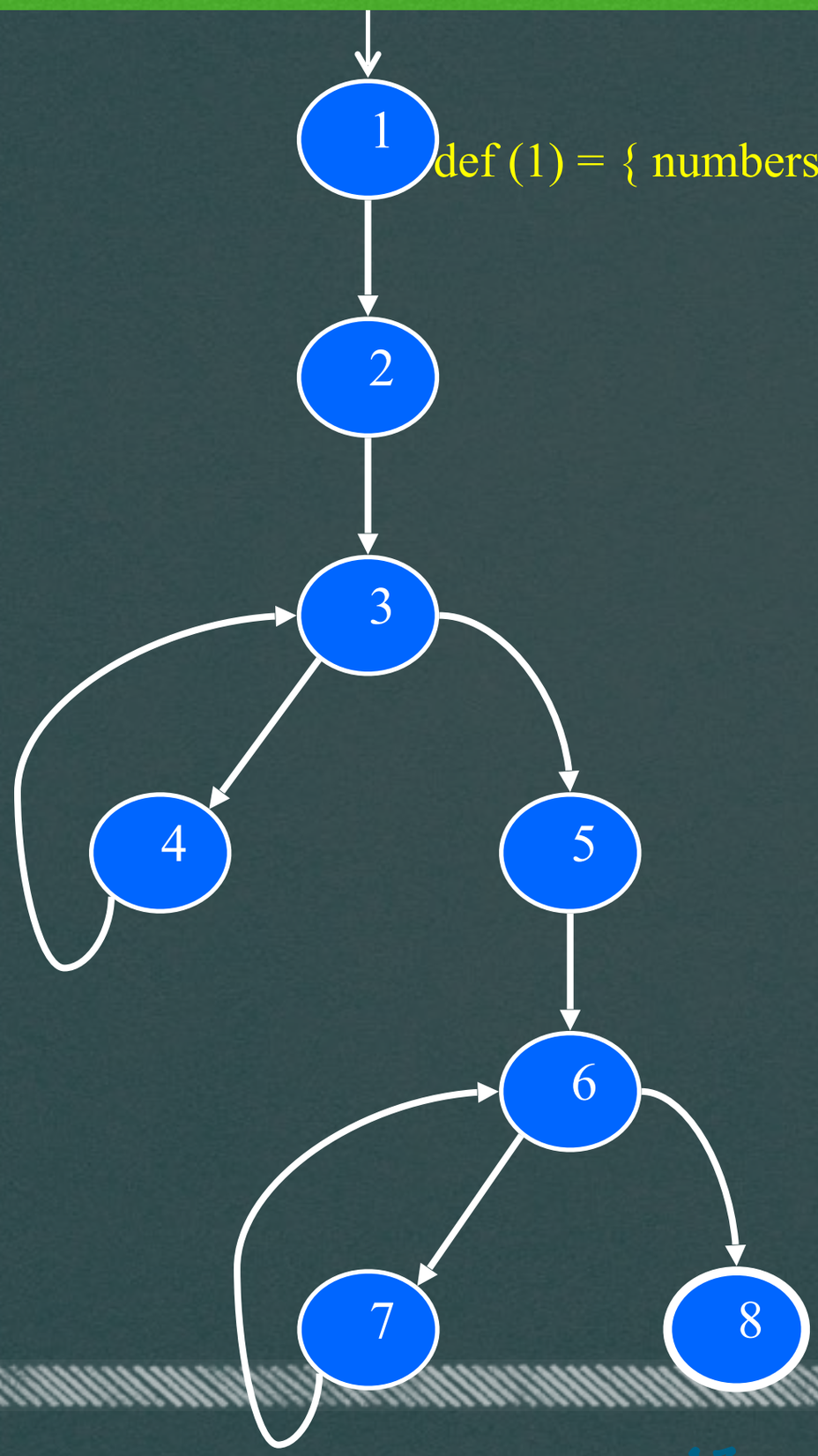
Control Flow Graph for Stats



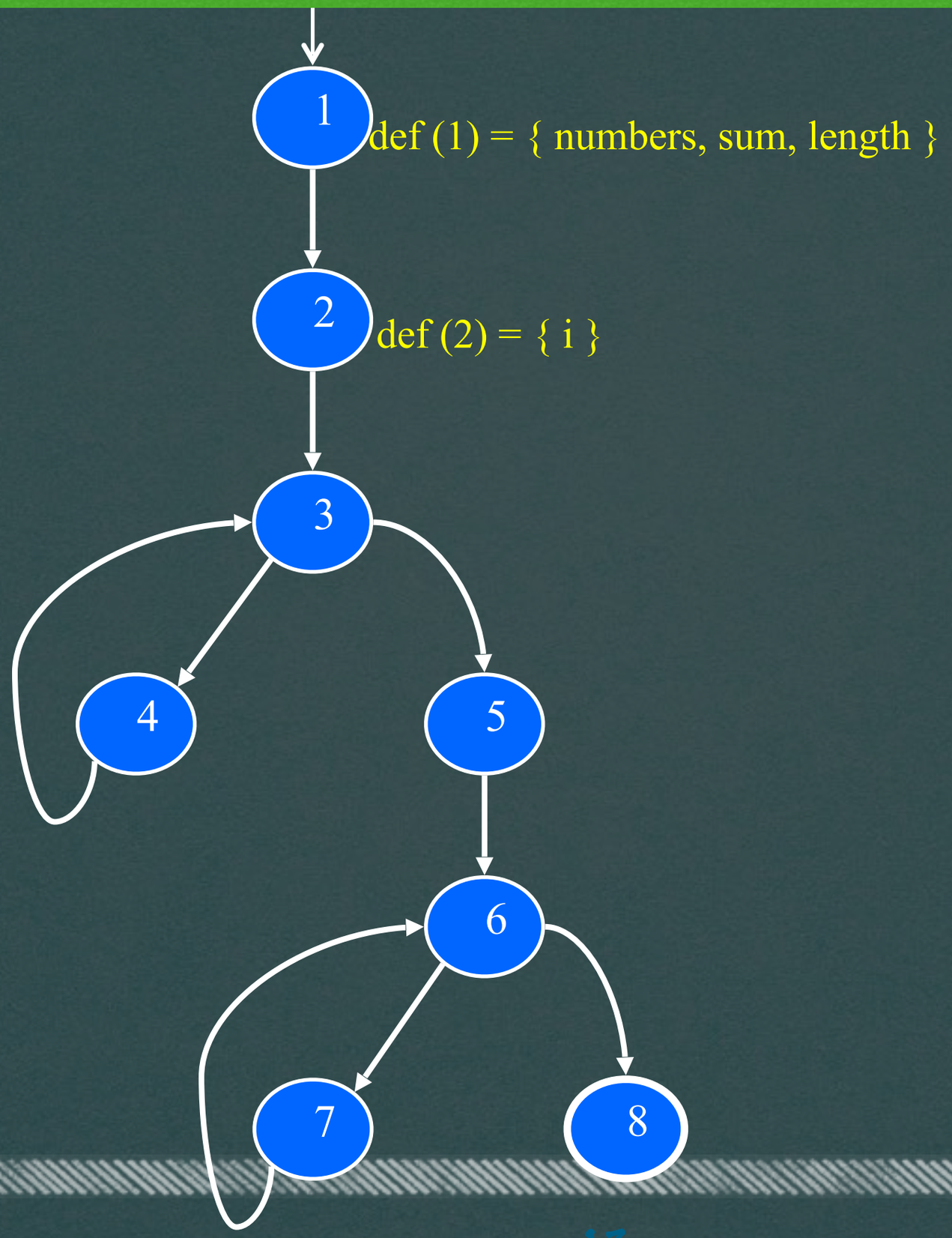
CFG for Stats – With Defs & Uses



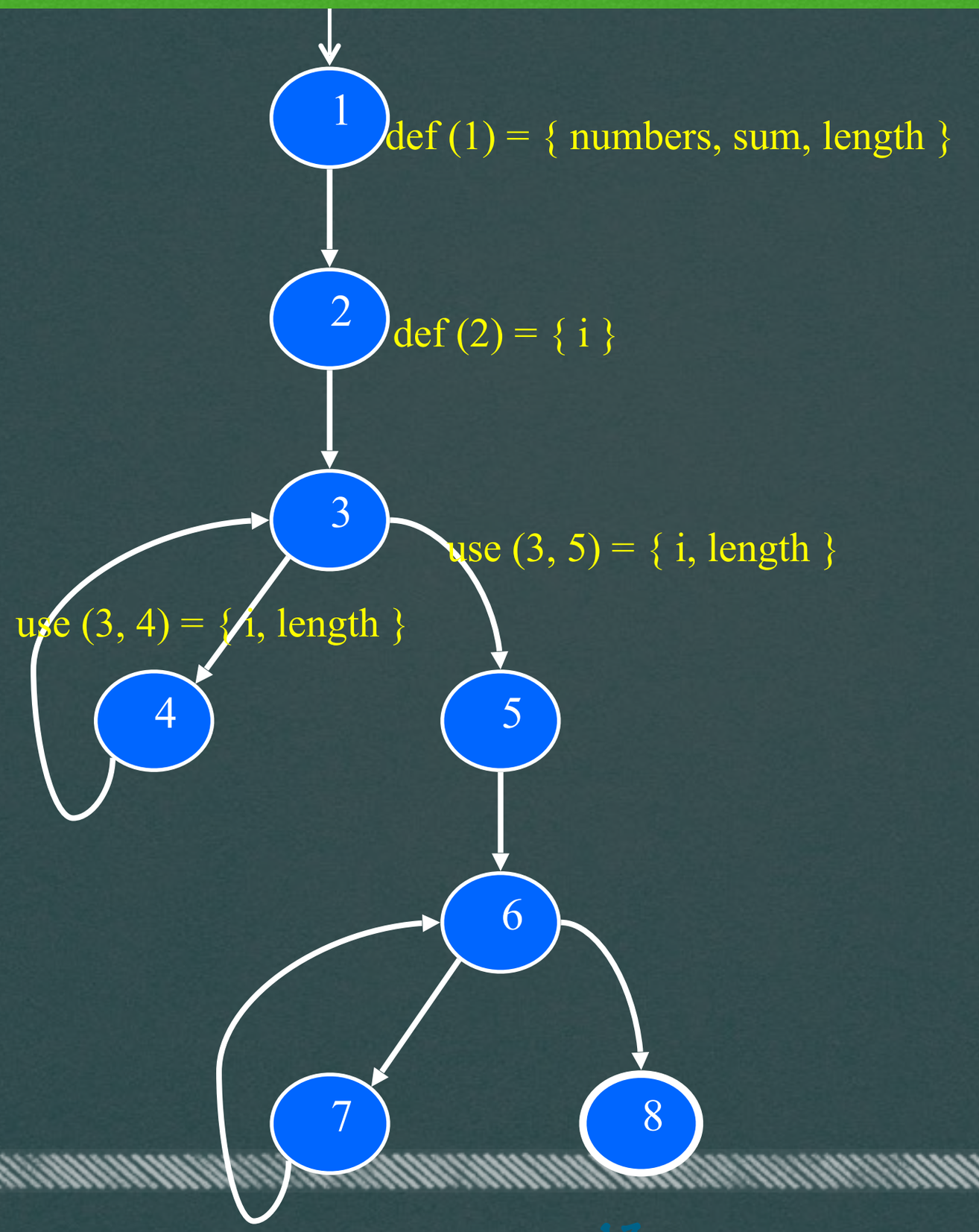
CFG for Stats – With Defs & Uses



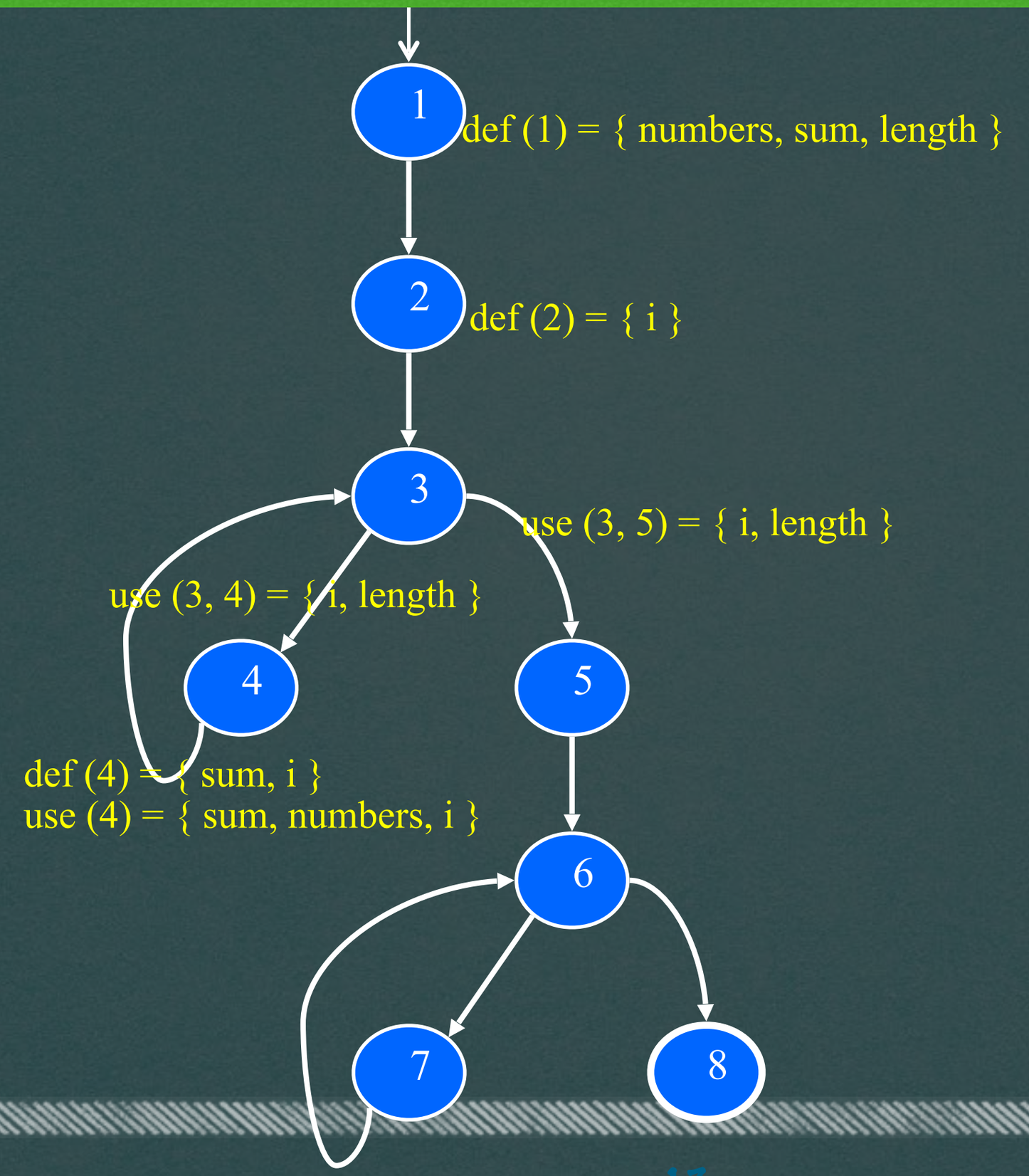
CFG for Stats – With Defs & Uses



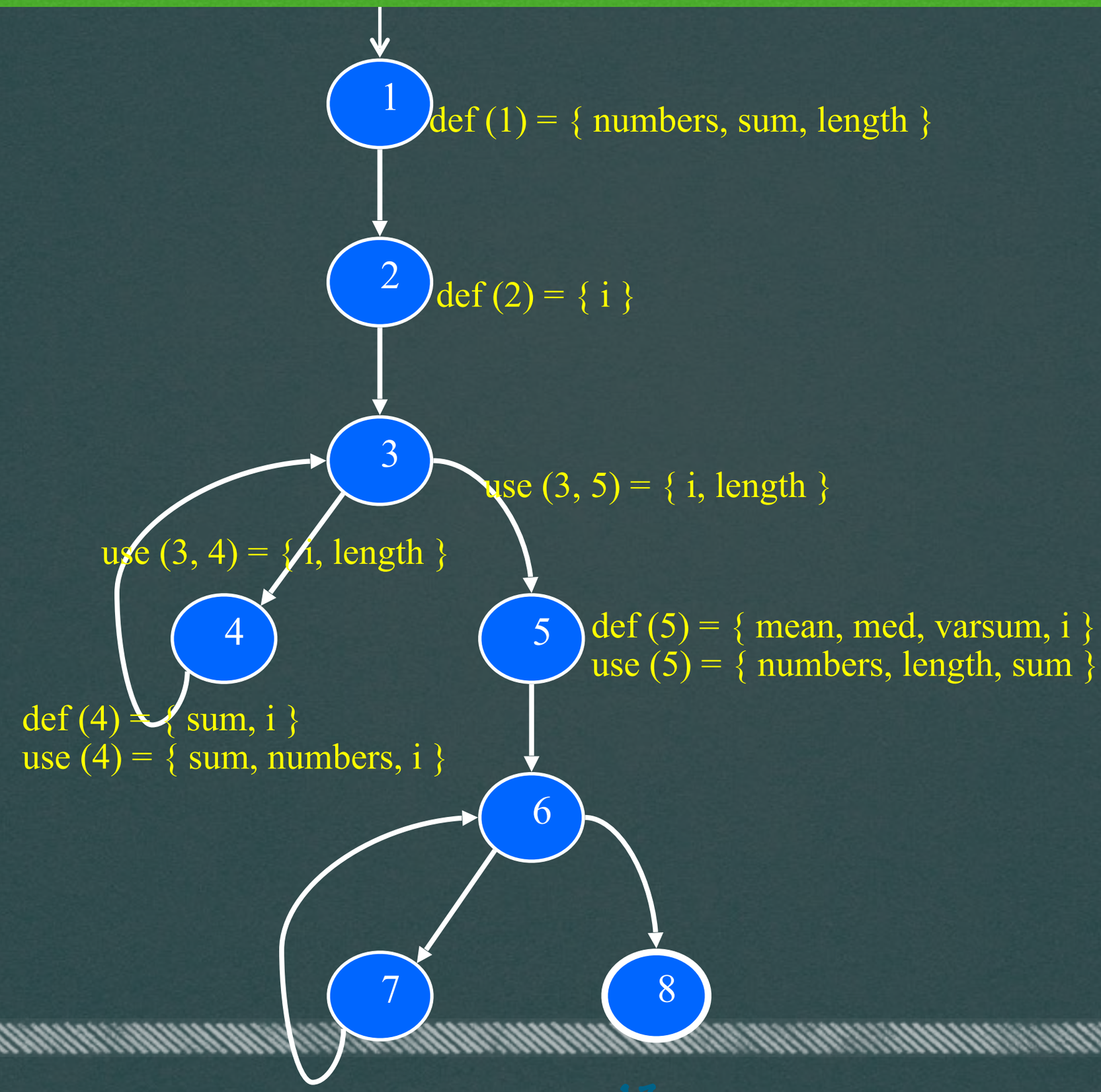
CFG for Stats – With Defs & Uses



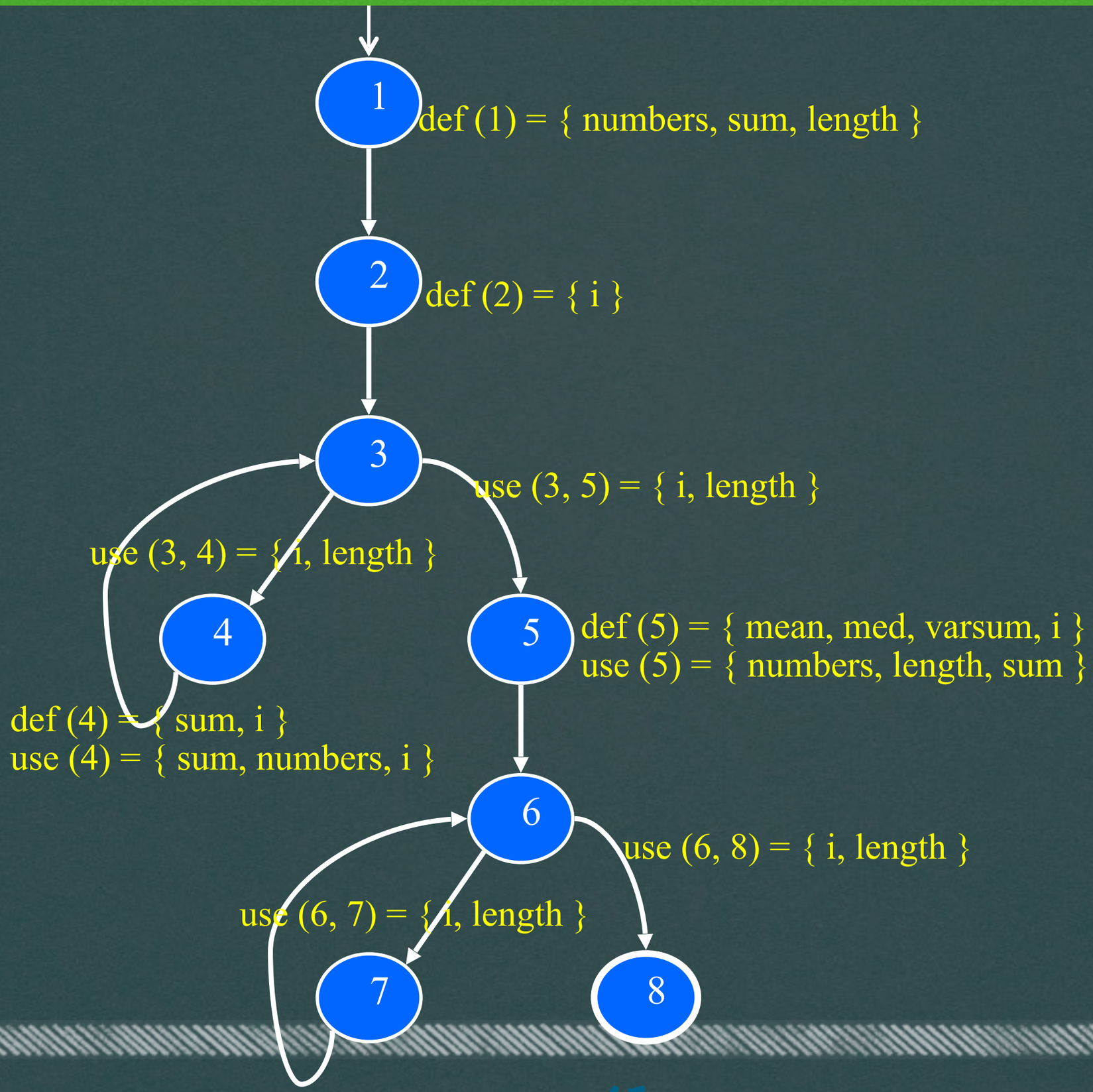
CFG for Stats – With Defs & Uses



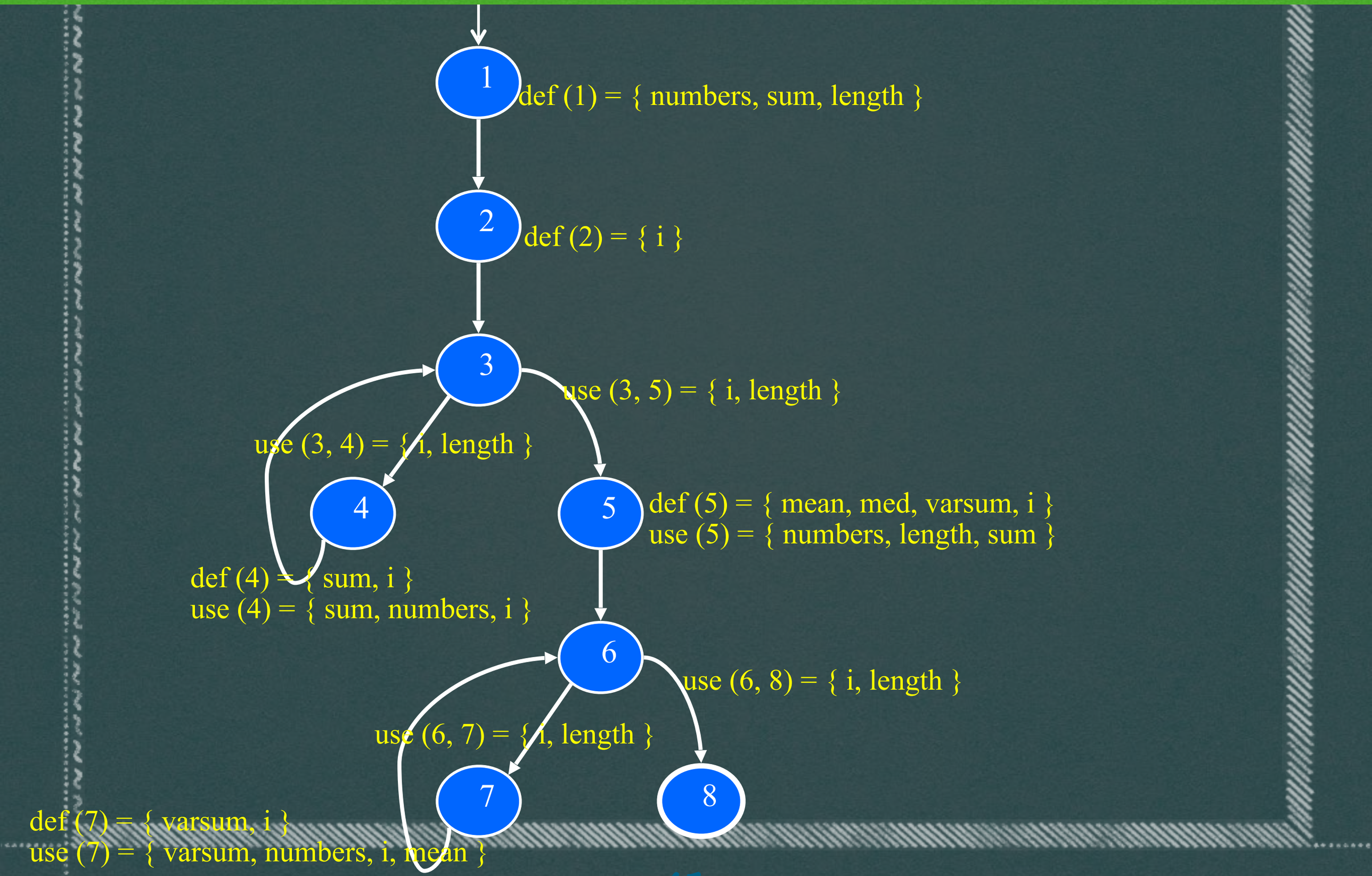
CFG for Stats – With Defs & Uses



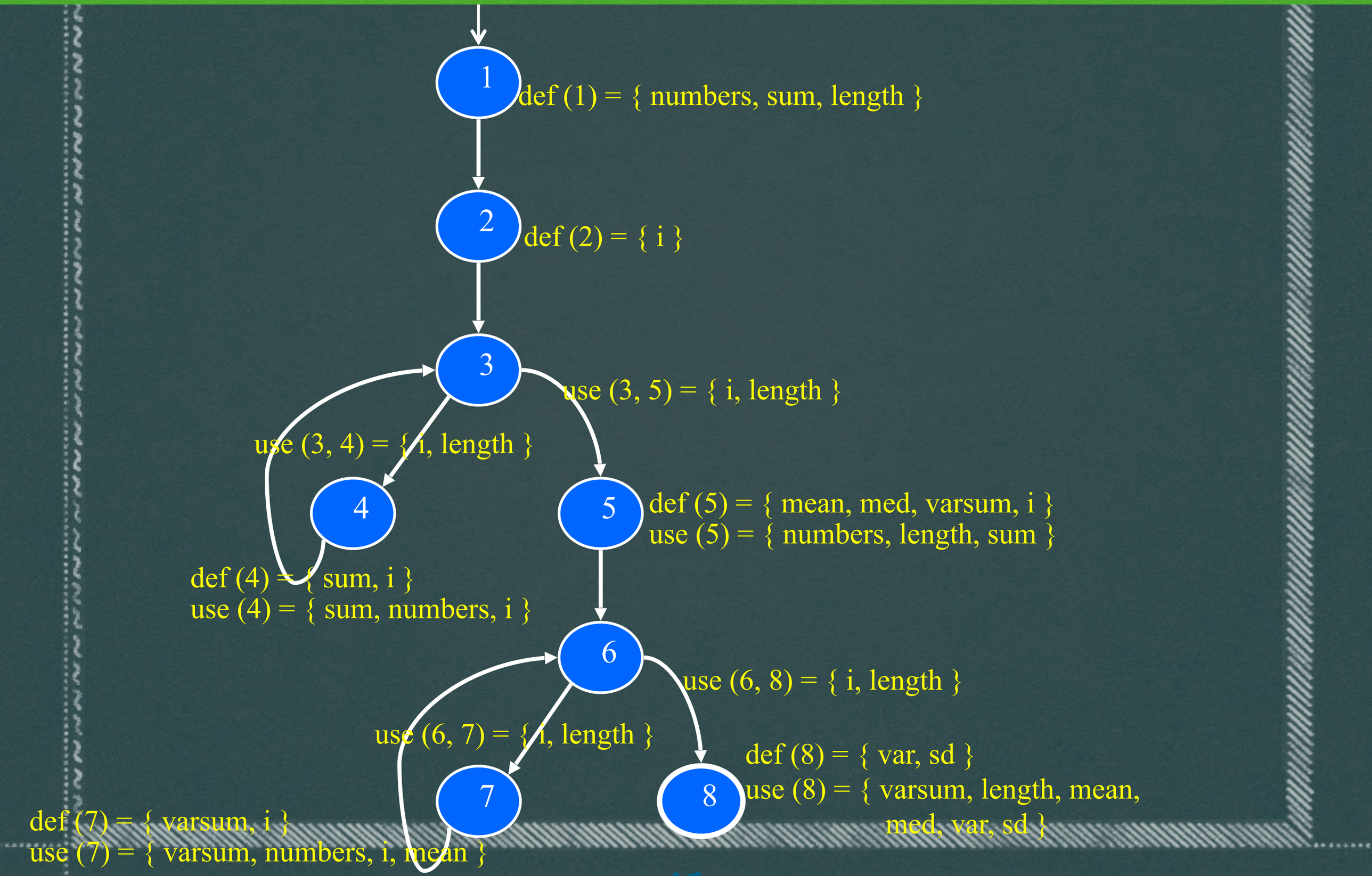
CFG for Stats – With Defs & Uses



CFG for Stats – With Defs & Uses



CFG for Stats – With Defs & Uses



Defs and Uses Tables for Stats

Node	Def	Use
1	{ numbers, sum, length }	
2	{ i }	
3		
4	{ sum, i }	{ numbers, i, sum }
5	{ mean, med, varsum, i }	{ numbers, length, sum }
6		
7	{ varsum, i }	{ varsum, numbers, i, mean }
8	{ var, sd }	{ varsum, length, var, mean, med, var, sd }

Edge	Use
(1, 2)	
(2, 3)	
(3, 4)	{ i, length }
(4, 3)	
(3, 5)	{ i, length }
(5, 6)	
(6, 7)	{ i, length }
(7, 6)	
(6, 8)	{ i, length }

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) (2, 7) (2, (6,7)) (2, (6,8)) (4, 4) (4, (3,4)) (4, (3,5)) (4, 7) (4, (6,7)) (4, (6,8)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) (2, 7) (2, (6,7)) (2, (6,8)) (4, 4) (4, (3,4)) (4, (3,5)) (4, 7) (4, (6,7)) (4, (6,8)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

defs come before uses, do not count as DU pairs

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) (2, 7) (2, (6,7)) (2, (6,8)) (4, 4) (4, (3,4)) (4, (3,5)) (4, 7) (4, (6,7)) (4, (6,8)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

defs come before uses, do not count as DU pairs

defs after use in loop, these are valid DU pairs

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) ~~(2, 7) (2, (6,7)) (2, (6,8))~~ (4, 4) (4, (3,4)) (4, (3,5)) ~~(4, 7) (4, (6,7)) (4, (6,8))~~ (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

defs come before uses, do not count as DU pairs

defs after use in loop, these are valid DU pairs

No def-clear path … different scope for i

DU Pairs for Stats

variable	DU Pairs
numbers	(1, 4) (1, 5) (1, 7)
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))
med	(5, 8)
var	(8, 8)
sd	(8, 8)
mean	(5, 7) (5, 8)
sum	(1, 4) (1, 5) (4, 4) (4, 5)
varsum	(5, 7) (5, 8) (7, 7) (7, 8)
i	(2, 4) (2, (3,4)) (2, (3,5)) ~~(2, 7) (2, (6,7)) (2, (6,8))~~ (4, 4) (4, (3,4)) (4, (3,5)) ~~(4, 7) (4, (6,7)) (4, (6,8))~~ (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))

defs come before uses, do not count as DU pairs

defs after use in loop, these are valid DU pairs

No def-clear path … different scope for i

No path through graph from nodes 5 and 7 to 4 or 3

DU Paths for Stats

variable	DU Pairs	DU Paths
numbers	(1, 4) (1, 5) (1, 7)	[1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7]
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))	[1, 2, 3, 5] [1, 2, 3, 5, 6, 8] [1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7] [1, 2, 3, 5, 6, 8]
med	(5, 8)	[5, 6, 8]
var	(8, 8)	*No path needed*
sd	(8, 8)	*No path needed*
sum	(1, 4) (1, 5) (4, 4) (4, 5)	[1, 2, 3, 4] [1, 2, 3, 5] [4, 3, 4] [4, 3, 5]

variable	DU Pairs	DU Paths
mean	(5, 7) (5, 8)	[5, 6, 7] [5, 6, 8]
varsum	(5, 7) (5, 8) (7, 7) (7, 8)	[5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 8]
i	(2, 4) (2, (3,4)) (2, (3,5)) (4, 4) (4, (3,4)) (4, (3,5)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))	[2, 3, 4] [2, 3, 4] [2, 3, 5] [4, 3, 4] [4, 3, 4] [4, 3, 5] [5, 6, 7] [5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 7] [7, 6, 8]

DU Paths for Stats – No Duplicates

There are 38 DU paths for Stats, but only 12 unique

[1, 2, 3, 4]	[4, 3, 4]
[1, 2, 3, 5]	[4, 3, 5]
[1, 2, 3, 5, 6, 7]	[5, 6, 7]
[1, 2, 3, 5, 6, 8]	[5, 6, 8]
[2, 3, 4]	[7, 6, 7]
[2, 3, 5]	[7, 6, 8]

DU Paths for Stats – No Duplicates

There are 38 DU paths for Stats, but only 12 unique

[1, 2, 3, 4]	[4, 3, 4]
★ [1, 2, 3, 5]	[4, 3, 5]
★ [1, 2, 3, 5, 6, 7]	[5, 6, 7]
★ [1, 2, 3, 5, 6, 8]	[5, 6, 8] ★
[2, 3, 4]	[7, 6, 7]
★ [2, 3, 5]	[7, 6, 8]

★ 5 expect a loop not to be “entered”

DU Paths for Stats – No Duplicates

There are 38 DU paths for Stats, but only 12 unique

[1, 2, 3, 4]	[4, 3, 4]
[1, 2, 3, 5]	[4, 3, 5]
[1, 2, 3, 5, 6, 7]	[5, 6, 7]
[1, 2, 3, 5, 6, 8]	[5, 6, 8]
[2, 3, 4]	[7, 6, 7]
[2, 3, 5]	[7, 6, 8]

5 expect a loop not to be “entered”

5 require at least one iteration of a loop

DU Paths for Stats – No Duplicates

There are 38 DU paths for Stats, but only 12 unique

[1, 2, 3, 4]	[4, 3, 4]
[1, 2, 3, 5]	[4, 3, 5]
[1, 2, 3, 5, 6, 7]	[5, 6, 7]
[1, 2, 3, 5, 6, 8]	[5, 6, 8]
[2, 3, 4]	[7, 6, 7]
[2, 3, 5]	[7, 6, 8]

5 expect a loop not to be “entered”

5 require at least one iteration of a loop

2 require at least two iteration of a loop

Test Cases and Test Paths

Test Cases and Test Paths

Test Case : numbers = (44) ; length = 1

Test Path : [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]

Additional DU Paths covered (no sidetrips)

[1, 2, 3, 4] [2, 3, 4] [4, 3, 5] [5, 6, 7] [7, 6, 8]

The five stars ✦ that require at least one iteration of a loop

Test Cases and Test Paths

Test Case : numbers = (44) ; length = 1
Test Path : [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
Additional DU Paths covered (no sidetrips)
[1, 2, 3, 4] [2, 3, 4] [4, 3, 5] [5, 6, 7] [7, 6, 8]
The five stars ✦ that require at least one iteration of a loop

Test Case : numbers = (2, 10, 15) ; length = 3
Test Path : [1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8]
DU Paths covered (no sidetrips)
[4, 3, 4] [7, 6, 7]
The two stars ✹ that require at least two iterations of a loop

Test Cases and Test Paths

Test Case : numbers = (44) ; length = 1
Test Path : [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
Additional DU Paths covered (no sidetrips)
[1, 2, 3, 4] [2, 3, 4] [4, 3, 5] [5, 6, 7] [7, 6, 8]
The five stars ✦ that require at least one iteration of a loop

Test Case : numbers = (2, 10, 15) ; length = 3
Test Path : [1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8]
DU Paths covered (no sidetrips)
[4, 3, 4] [7, 6, 7]
The two stars ✹ that require at least two iterations of a loop

Other DU paths ★ require arrays with length 0 to skip loops
But the method fails with divide by zero on the statement …

```
mean = sum / (double) length;
```

Test Cases and Test Paths

Test Case : numbers = (44) ; length = 1
Test Path : [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]
Additional DU Paths covered (no sidetrips)
[1, 2, 3, 4] [2, 3, 4] [4, 3, 5] [5, 6, 7] [7, 6, 8]
The five stars ✦ that require at least one iteration of a loop

Test Case : numbers = (2, 10, 15) ; length = 3
Test Path : [1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8]
DU Paths covered (no sidetrips)
[4, 3, 4] [7, 6, 7]
The two stars ✹ that require at least two iterations of a loop

Other DU paths ★ require arrays with length 0 to skip loops
But the method fails with divide by zero on the statement …

```
mean = sum / (double) length;
```

A fault was found

Example: TestPat

```
public int pat (char[] subject, char[] pattern)
{
// Post: if pattern is not a substring of
    subject, return -1
//       else return (zero-based) index where
    the pattern (first)
//       starts in subject
   final int NOTFOUND = -1;
   int  iSub = 0, rtnIndex = NOTFOUND;
   boolean isPat  = false;
   int subjectLen = subject.length;
   int patternLen = pattern.length;
```

```
while (isPat == false && iSub + patternLen - 1 <
    subjectLen)
  {
    if (subject [iSub] == pattern [0])
    {
      rtnIndex = iSub; // Starting at zero
      isPat = true;
      for (int iPat = 1; iPat < patternLen; iPat ++)
      {
        if (subject[iSub + iPat] != pattern[iPat])
        {
          rtnIndex = NOTFOUND;
          isPat = false;
          break;  // out of for loop
        }
      }
    }
    iSub ++;
  }
  return (rtnIndex);
}
}
```

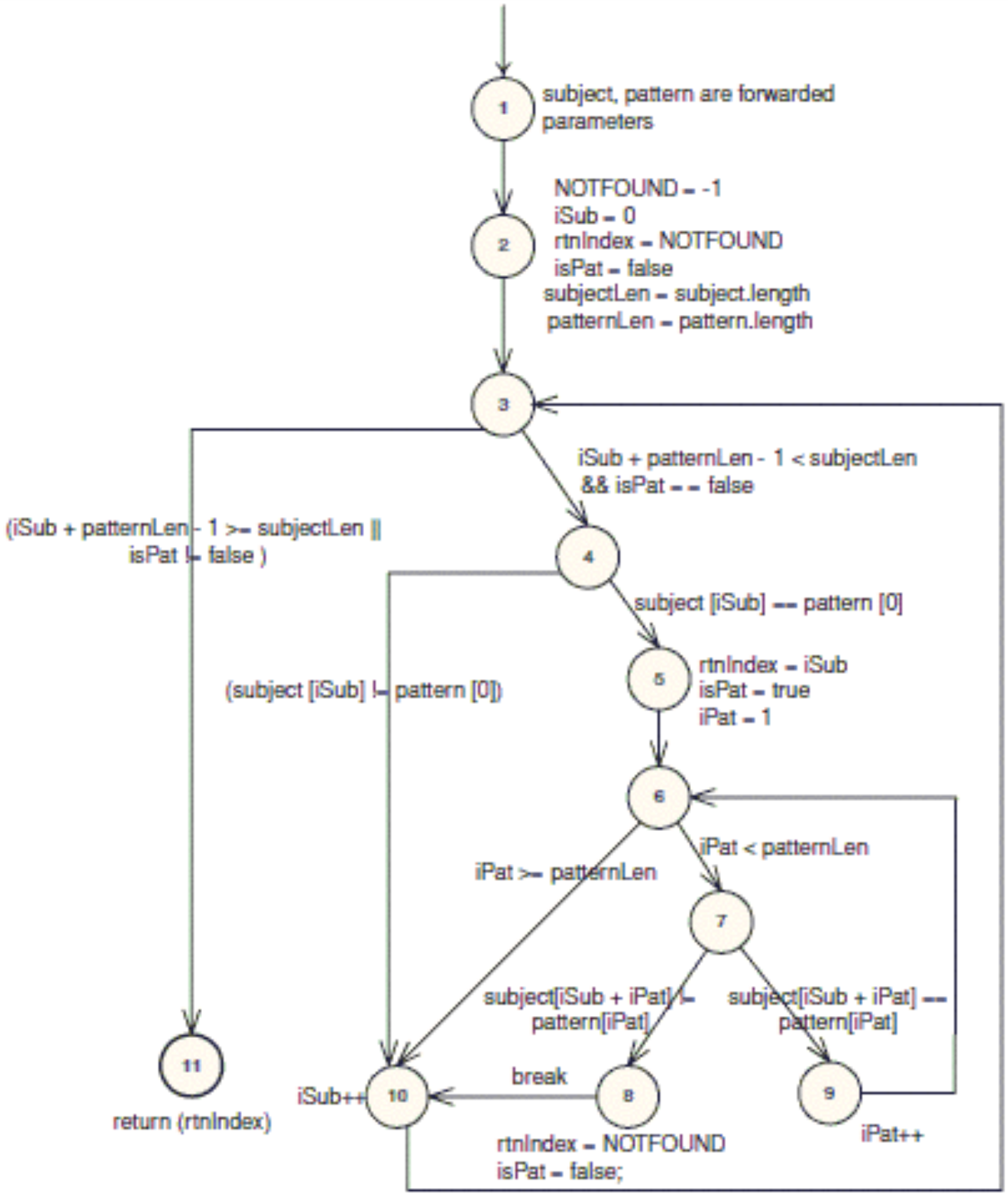


Figure 2.12. A graph showing an example of du-paths.

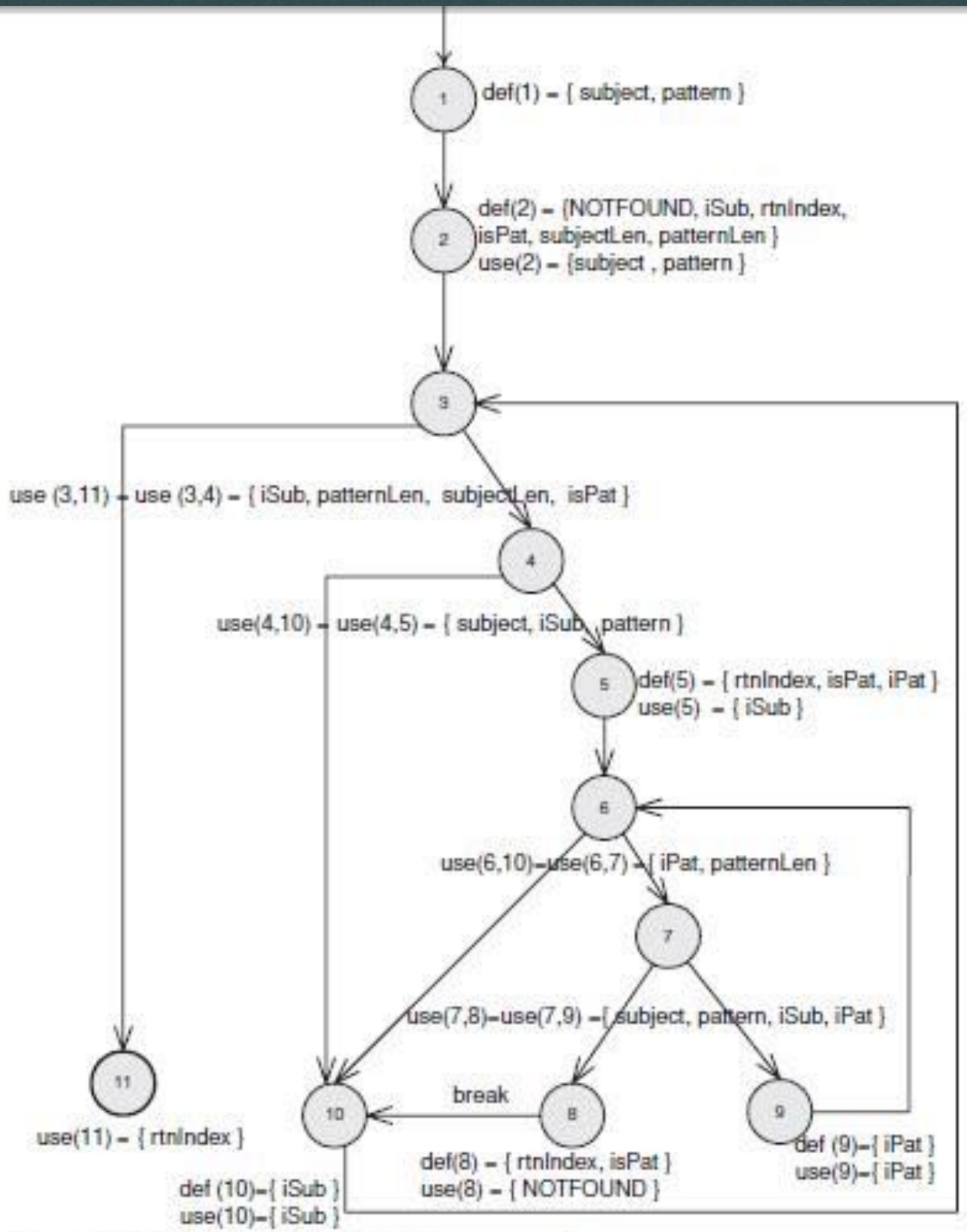


Figure 2.13. Graph showing explicit def and use sets.

Table 2.1. Defs and uses at each node in the CFG for TestPat

node	def	use
1	{subject, pattern}	
2	{NOTFOUND, isPat, iSub, rtnIndex, subjectLen, patternLen}	{subject, pattern}
3		
4		
5	{rtnIndex, isPat, iPat}	{iSub}
6		
7		
8	{rtnIndex, isPat}	{NOTFOUND}
9	{iPat}	{iPat}
10	{iSub}	{iSub}
11		{rtnIndex}

Table 2.2. Defs and uses at each edge in the CFG for TestPat.

edge	use
(1, 2)	
(2, 3)	
(3, 4)	{iSub, patternLen, subjectLen, isPat}
(3, 11)	{iSub, patternLen, subjectLen, isPat}
(4, 5)	{subject, iSub, pattern}
(4, 10)	{subject, iSub, pattern}
(5, 6)	
(6, 7)	{iPat, patternLen}
(6, 10)	{iPat, patternLen}
(7, 8)	{subject, iSub, iPat, pattern}
(7, 9)	{subject, iSub, iPat, pattern}
(8, 10)	
(9, 6)	
(10, 3)	

Table 2.3. Du-path sets for each variable in TestPat

variable	du-path set	du-paths	prefix?
NOTFOUND	du (2, NOTFOUND)	[2,3,4,5,6,7,8]	
rtnIndex	du (2, rtnIndex)	[2,3,11]	
	du (5, rtnIndex)	[5,6,10,3,11]	
	du (8, rtnIndex)	[8,10,3,11]	
iSub	du (2, iSub)	[2,3,4]	Yes
		[2,3,4,5]	Yes
		[2,3,4,5,6,7,8]	Yes
		[2,3,4,5,6,7,9]	
		[2,3,4,5,6,10]	
		[2,3,4,5,6,7,8,10]	
		[2,3,4,10]	
		[2,3,11]	
	du (10, iSub)	[10,3,4]	Yes
		[10,3,4,5]	Yes
		[10,3,4,5,6,7,8]	Yes
		[10,3,4,5,6,7,9]	
		[10,3,4,5,6,10]	
		[10,3,4,5,6,7,8,10]	
		[10,3,4,10]	
		[10,3,11]	
iPat	du (5, iPat)	[5,6,7]	Yes
		[5,6,10]	
		[5,6,7,8]	
		[5,6,7,9]	
	du (9, iPat)	[9,6,7]	Yes
		[9,6,10]	
		[9,6,7,8]	
		[9,6,7,9]	
isPat	du (2, isPat)	[2,3,4]	
		[2,3,11]	
	du (5, isPat)	[5,6,10,3,4]	
		[5,6,10,3,11]	
	du (8, isPat)	[8,10,3,4]	
		[8,10,3,11]	
subject	du (1, subject)	[1,2]	Yes
		[1,2,3,4,5]	Yes
		[1,2,3,4,10]	
		[1,2,3,4,5,6,7,8]	
		[1,2,3,4,5,6,7,9]	
pattern	du (1, pattern)	[1,2]	Yes
		[1,2,3,4,5]	Yes
		[1,2,3,4,10]	
		[1,2,3,4,5,6,7,8]	
		[1,2,3,4,5,6,7,9]	
subject_len	du (2, subjectLen)	[2,3,4]	
		[2,3,11]	
patternLen	du (2, patternLen)	[2,3,4]	Yes
		[2,3,11]	
		[2,3,4,5,6,7]	
		[2,3,4,5,6,10]	

Table 2.4. Test paths to satisfy all du-paths coverage on TestPat

test case (subject,pattern,output)	test path(t)
(a, bc, −1)	[1,2,3,11]
(ab, a, 0)	[1,2,3,4,5,6,10,3,11]
(ab, ab, 0)	[1,2,3,4,5,6,7,9,6,10,3,11]
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]
(ab, b, 1)	[1,2,3,4,10,3,4,5,6,10,3,11]
(ab, c, −1)	[1,2,3,4,10,3,4,10,3,11]
(abc, abc, 0)	[1,2,3,4,5,6,7,9,6,7,9,6,10,3,11]
(abc, abd, −1)	[1,2,3,4,5,6,7,9,6,7,8,10,3,11]
(abc, ac −1)	[1,2,3,4,5,6,7,8,10,3,4,10,3,11]
(abc, ba, −1)	[1,2,3,4,10,3,4,5,6,7,8,10,3,11]
(abc, bc, 1)	[1,2,3,4,10,3,4,5,6,7,9,6,10,3,11]

Table 2.5. Test paths and du-paths covered on TestPat.

test case (subject,pattern, output)	test path(t)	du-path toured
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]	[2,3,4,5,6,7,8](NOTFOUND)
(a, bc, −1)	[1,2,3,11]	[2,3,11](rtnIndex)
(ab, a, 0)	[1,2,3,4,5,6,10,3,11]	[5,6,10,3,11](rtnIndex)
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]	[8,10,3,11](rtnIndex)
(ab, ab, 0)	[1,2,3,4,5,6,7,9,6,10,3,11]	[2,3,4,5,6,7,9] (iSub)
(ab, a, 0)	[1,2,3,4,5,6,10,3,11]	[2,3,4,5,6,10](iSub)
(ab, ac, −1)	[1,2,3,4,5,6,7,8,10,3,11]	[2,3,4,5,6,7,8,10](iSub)
(ab, c, −1)	[1,2,3,4,10,3,4,10,3,11]	[2,3,4,10](iSub)
(a, bc, −1)	[1,2,3,11]	[2,3,11] (iSub)
(abc, bc, 1)	[1,2,3,4,10,3,4,5,6,7,9,6,10,3,11]	[10,3,4,5,6,7,9](iSub)
(ab, b, 1)	[1,2,3,4,10,3,4,5,6,10,3,11]	[10,3,4,5,6,10](iSub)
(abc, ba, −1)	[1,2,3,4,10,3,4,5,6,7,8,10,3,11]	[10,3,4,5,6,7,8,10](iSub)
(ab, c, −1)	[1,2,3,4,10,3,4,10,3,11]	[10,3,4,10](iSub)

خلاصه

- Applying the graph test criteria to control flow graphs is relatively straightforward
  - Most of the developmental research work was done with CFGs
- A few subtle decisions must be made to translate control structures into the graph
- Some tools will assign each statement to a unique node
  - These slides and the book uses basic blocks
  - Coverage is the same, although the bookkeeping will differ

Quick Sort کوییز دوم

```
QuickSort(a,beg,end) //a is Array
{
if(beg<end)
  {
  p=beg, pivot=a[beg];

  int p=beg, pivot=a[beg], loc;

  for(loc=beg+1;loc<=end;loc++)
  {
    if(pivot>a[loc])
    {
      a[p]=a[loc];
      a[loc]=a[p+1];
      a[p+1]=pivot;
      p=p+1;
    }
  }

  QuickSort(a,beg,p-1);
  QuickSort(a,p+1,end);
  }
}
```

کوییز دوم Quick Sort

```
QuickSort(a,beg,end) //a is Array
{
if(beg<end)
  {
  p=beg, pivot=a[beg];

  int p=beg, pivot=a[beg], loc;

  for(loc=beg+1;loc<=end;loc++)
  {
    if(pivot>a[loc])
    {
      a[p]=a[loc];
      a[loc]=a[p+1];
      a[p+1]=pivot;
      p=p+1;
    }
  }

  QuickSort(a,beg,p-1);
  QuickSort(a,p+1,end);
  }
}
```

۱. گراف معادل

۲. مسیرهای اصلی

۳. تعیین مسیرهای آزمون

۴. تعیین آزمونه‌ها

پایان