Graph Coverage For Source Code

Control Flow Graph (CFG).

- To apply one of the graph criteria, the first step is to define the graph, and for source code, the most common graph is called a *control flow graph (CFG)*.

Control flow graphs associate

- An edge with each possible branch in the program
- A node with sequences of statements

Basic Block

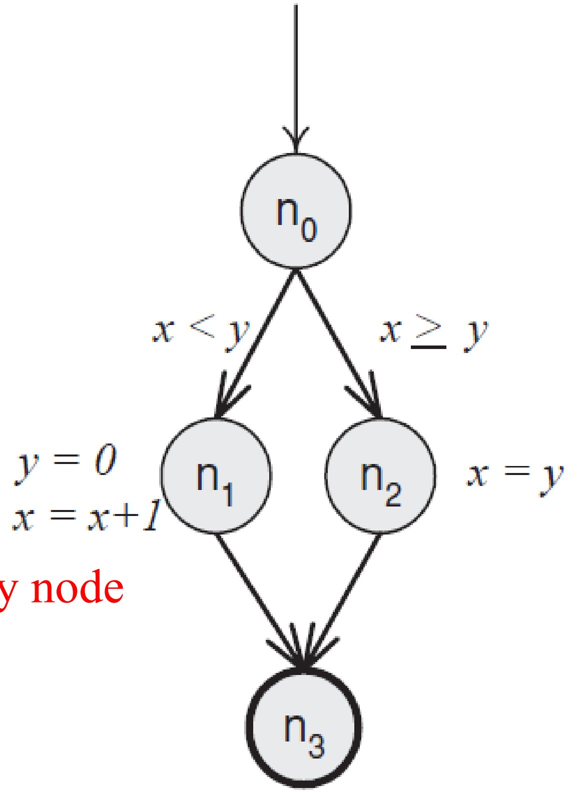
- A *basic block* is a maximum sequence of program statements such that if any one statement of the block is executed, all statements in the block are executed. A basic block has only one entry point and one exit point.

The application is direct with only the names being changed. Node coverage is often called *statement coverage* or *basic block coverage*, and edge coverage is often called *branch coverage*

Basic Block Example

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

decision node



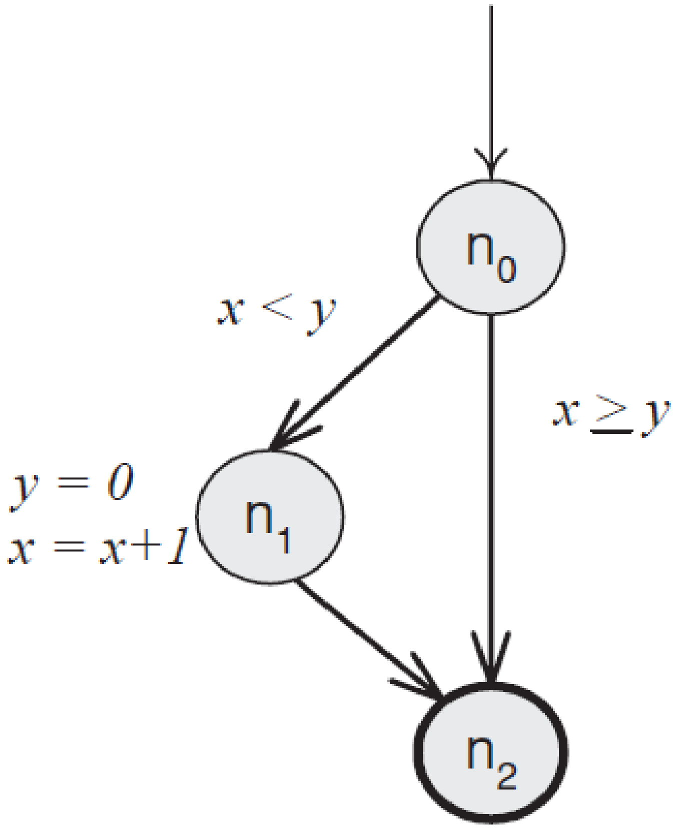
“dummy node

junction node

Basic Block Example

- note that a test with *x* < *y* traverses all of the nodes in this control flow graph, but not all of the edges.

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

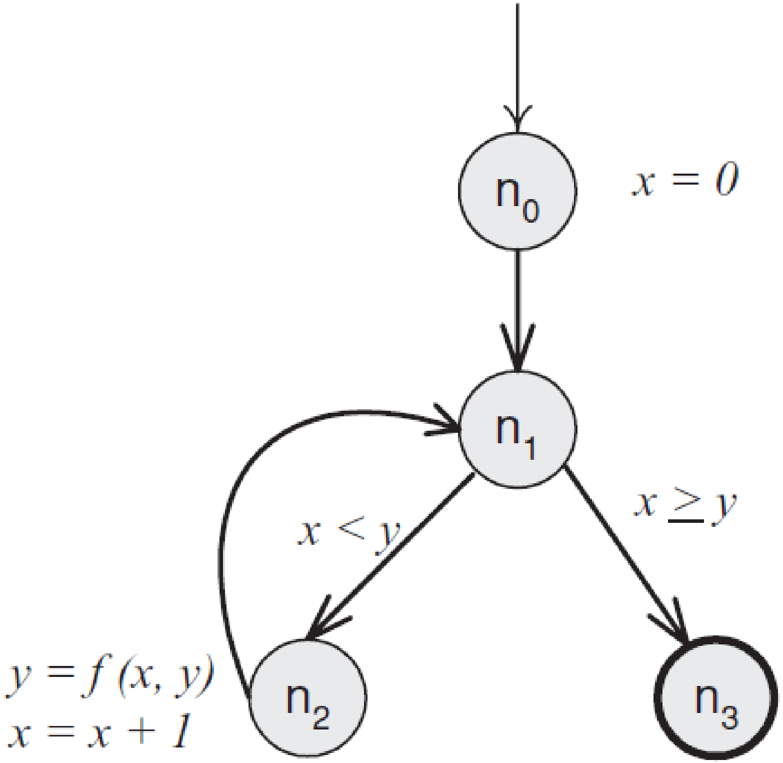


if statement without an else clause

Loops : A While Loop

- a while loop with an initializing statement

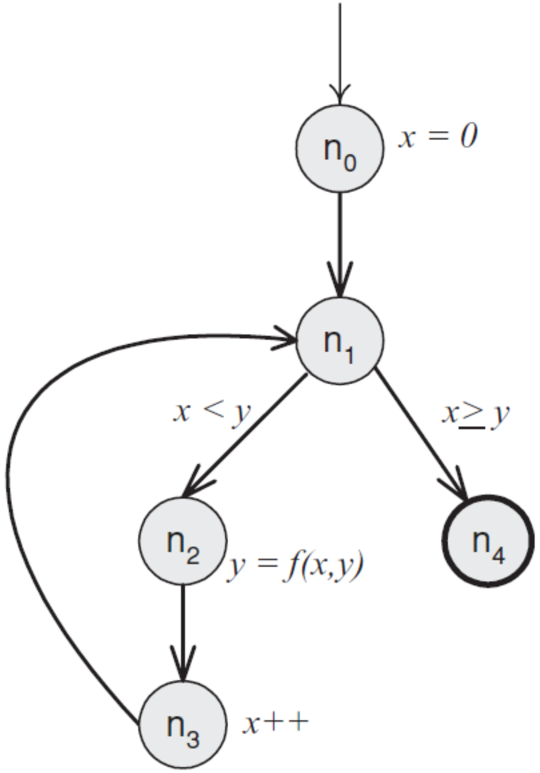
```
x = 0;
while (x < y)
{
   y = f (x, y);
   x = x+1;
}
```



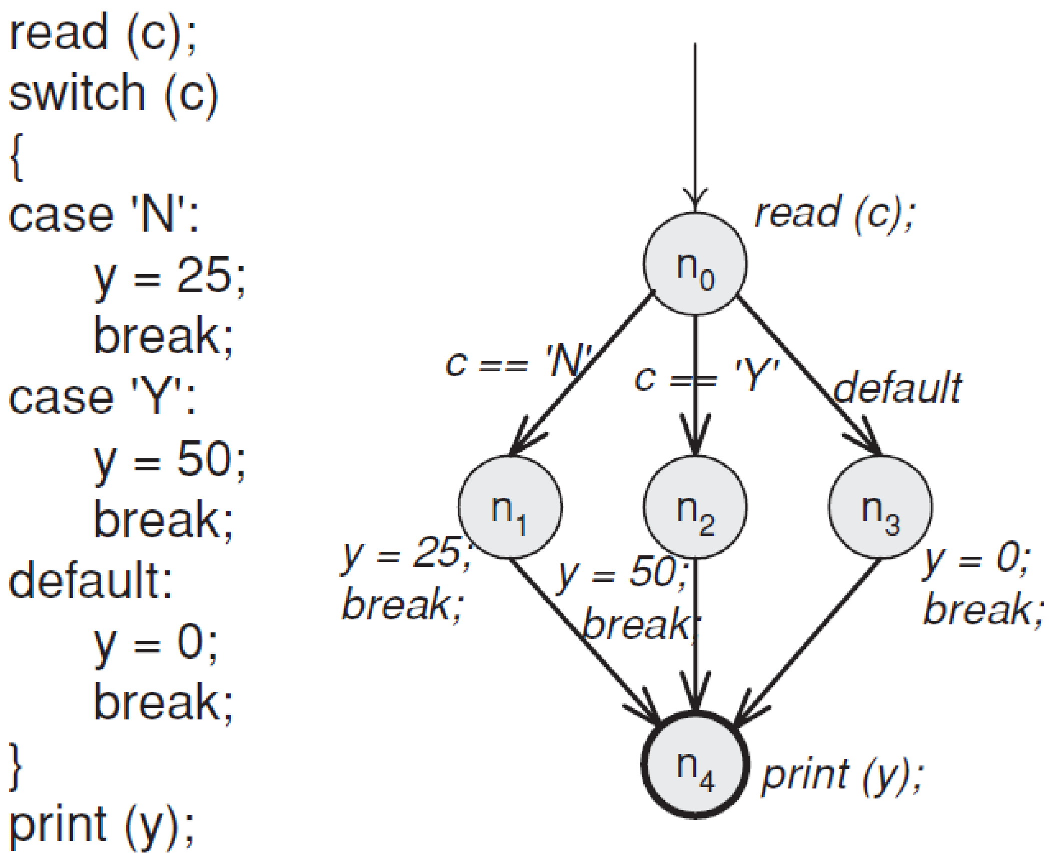
Loop: For loop

- Consider X++ in a different not

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



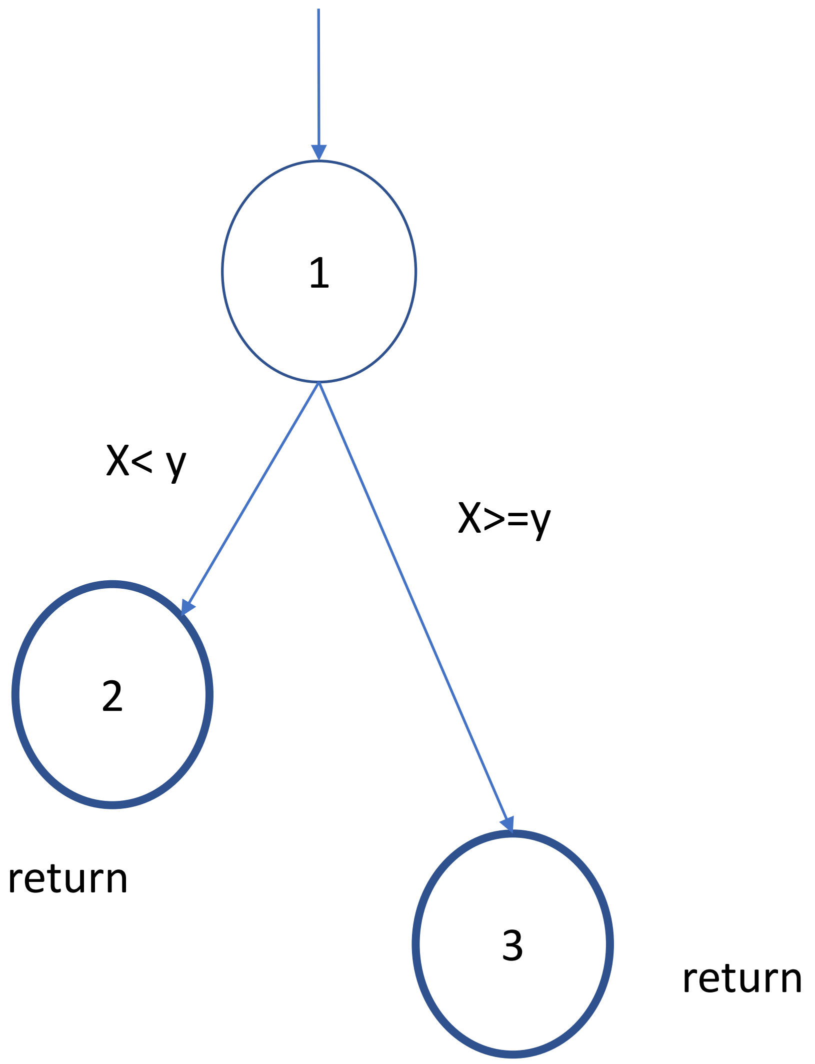
Case Statement



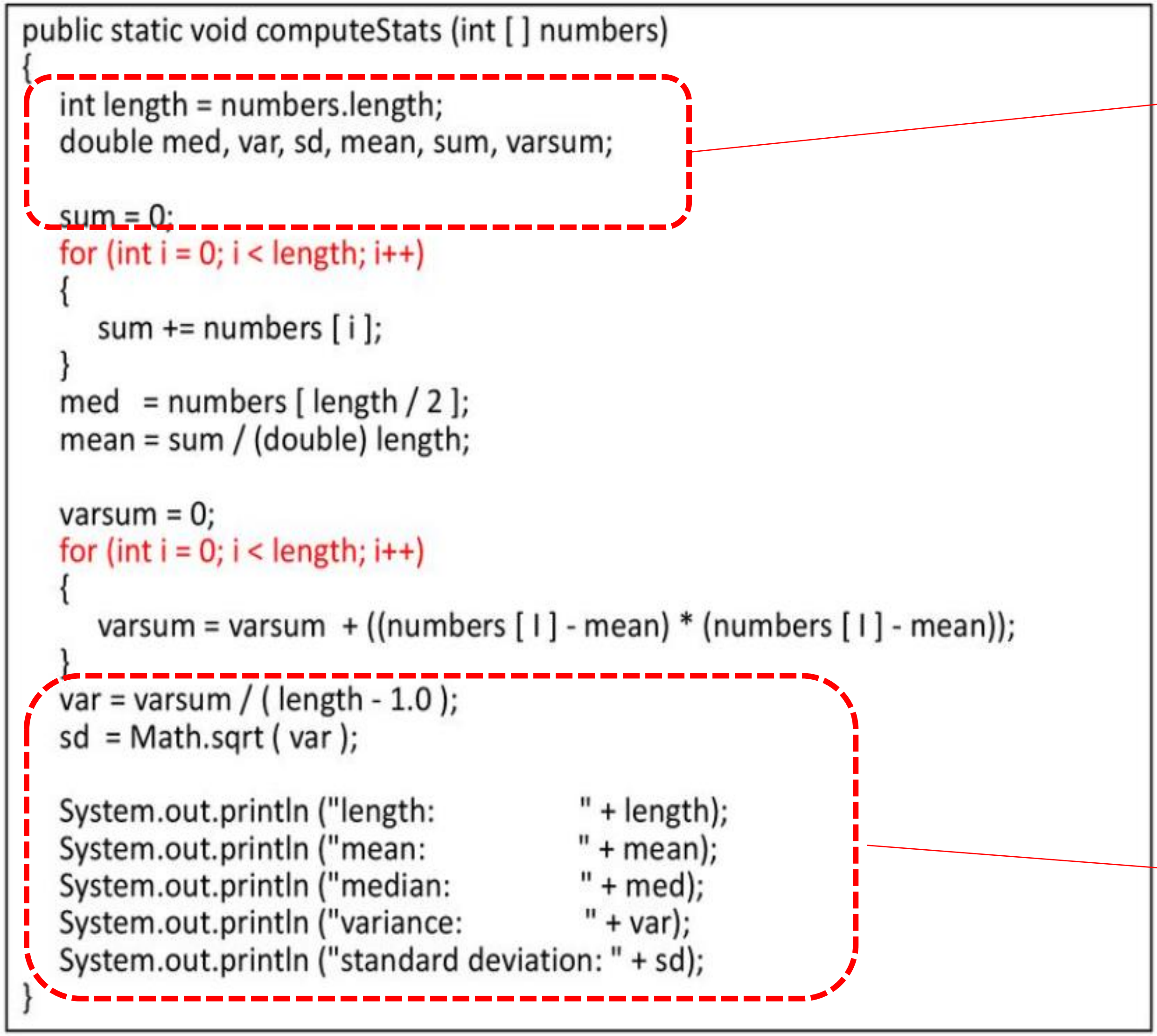
```
read (c);
switch (c)
{
case 'N':
        y = 25;
        break;
case 'Y':
        y = 50;
        break;
default:
        y = 0;
        break;
}
print (y);
```

Exercise

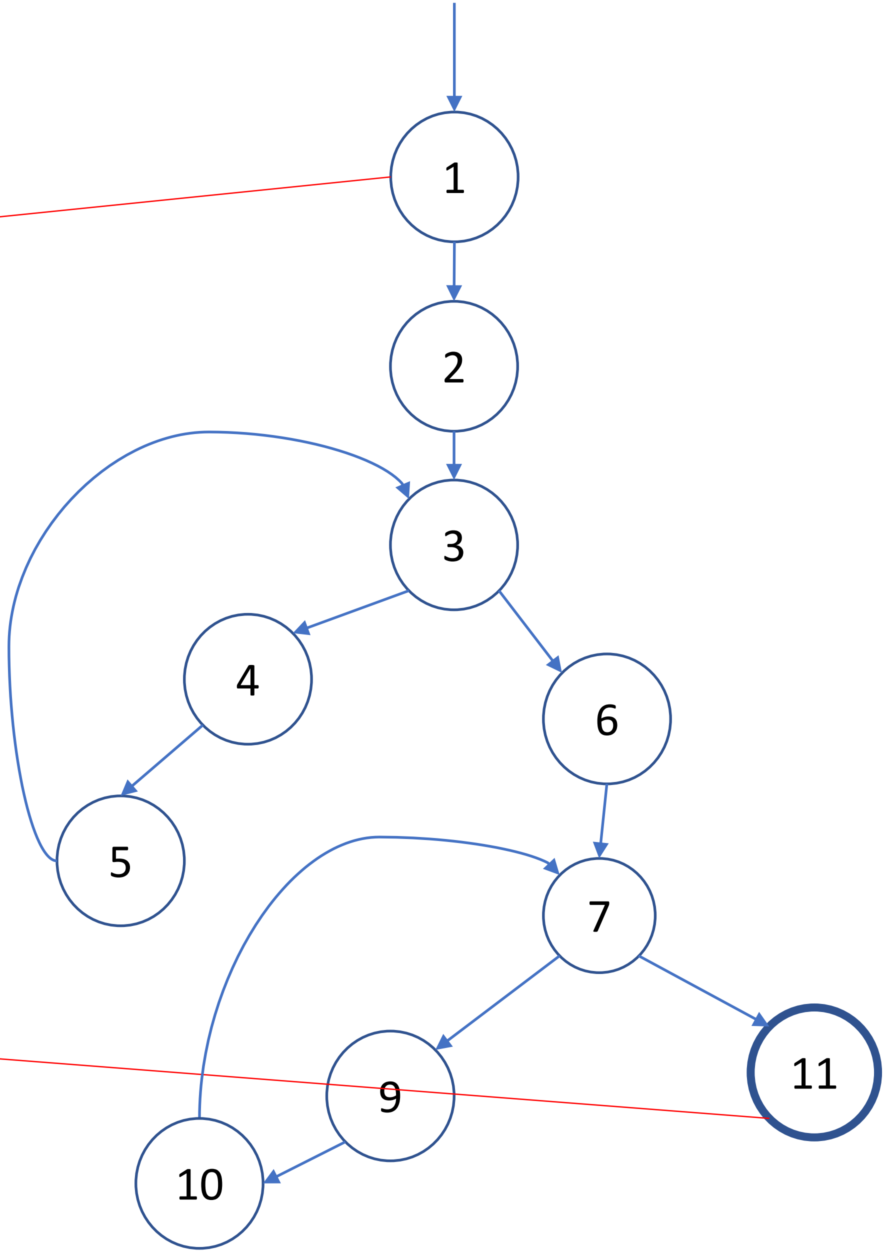
```
If (x < y) {
    return;
}
print(x);
return;
```



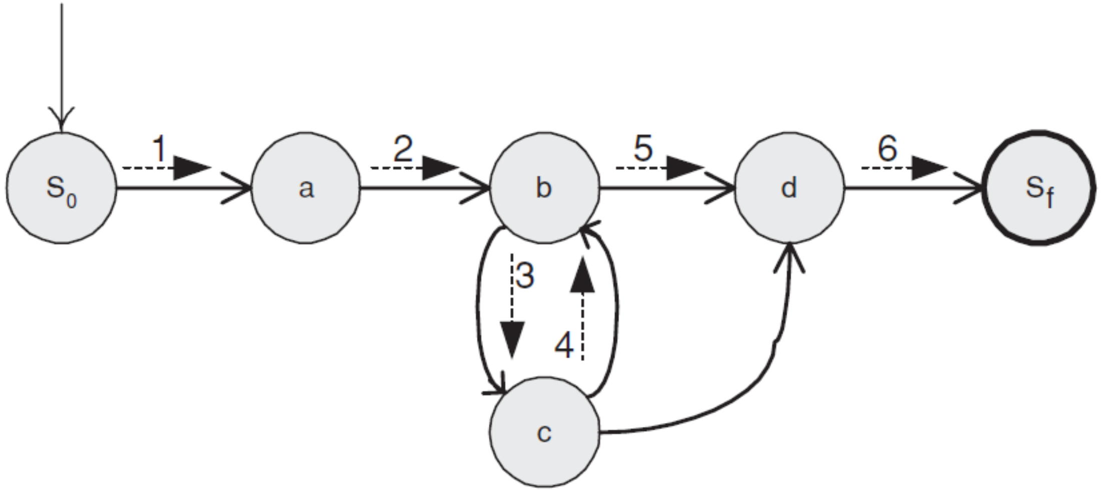
Draw the CFG for this code



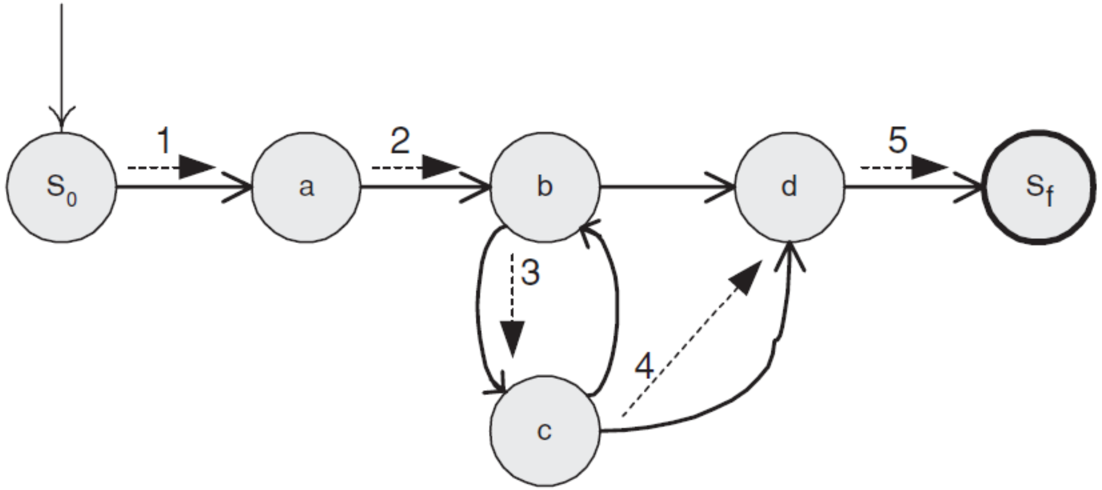
```
public static void computeStats (int [ ] numbers)
{
   int length = numbers.length;
   double med, var, sd, mean, sum, varsum;

   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);
}
```



Sidetrip and Detour

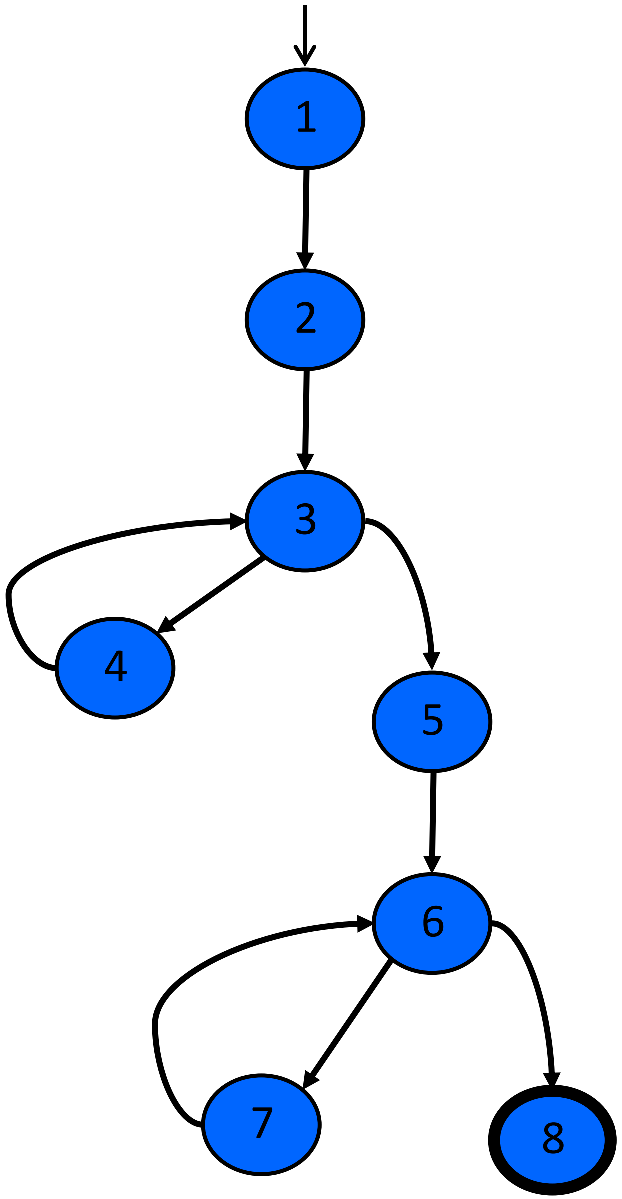


Graph being toured with a sidetrip



Graph being toured with a detour

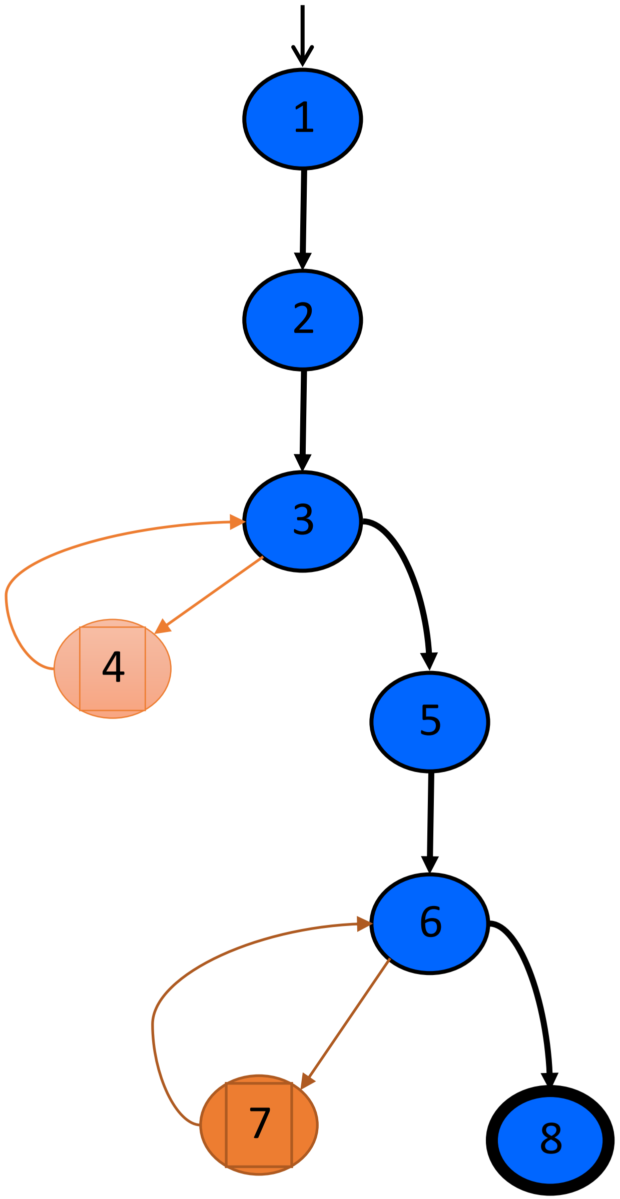
Control Flow TRs and Test Paths—EC



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

Remember a test path shall start at n0 and ends in nf

Control Flow TRs and Test Paths—EPC



Edge-Pair Coverage

TR

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]

Test Paths

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]

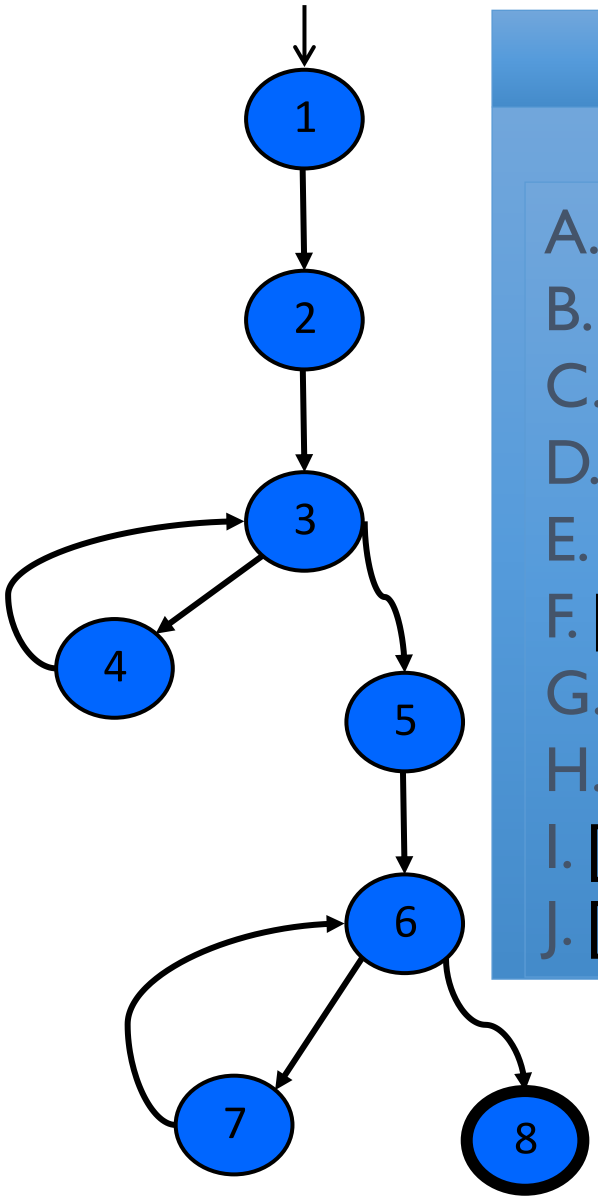
Visits the loop twice

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

K,L

TP iii makes TP i redundant. A *minimal* set of TPs is cheaper.

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]

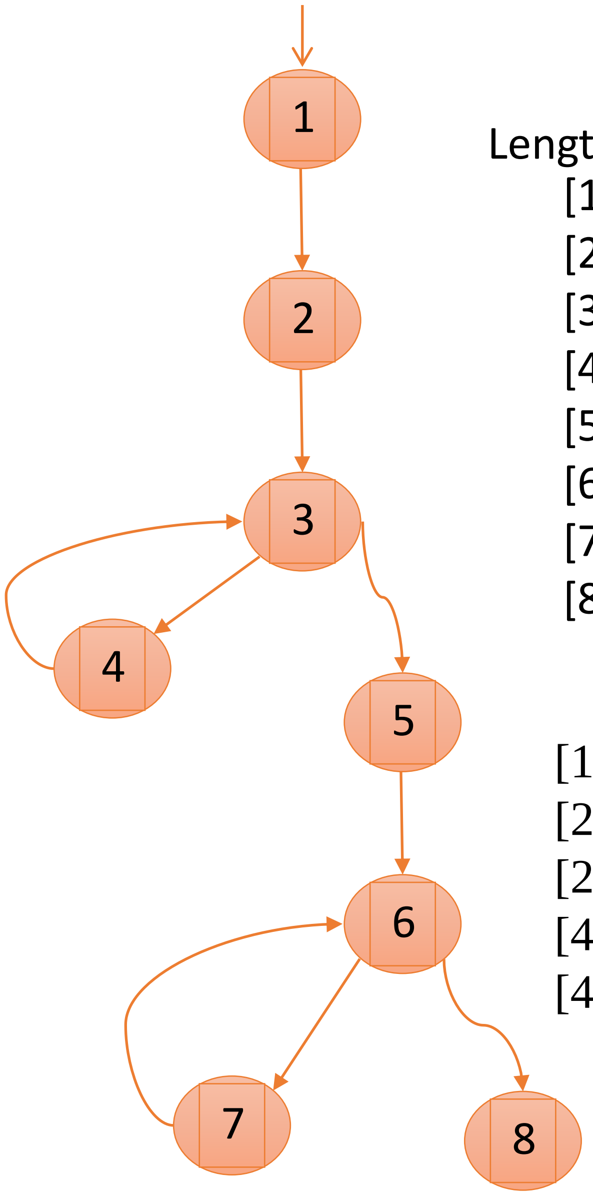
Test Paths

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, F, I	J
v	J	

TP ii makes
TP i redundant.

Control Flow TRs and Test Paths—PPC



Length 0

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

Length 1

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

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

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

Length 2

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

Length 3

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

Length 4

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

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

Length 5

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

Length 6

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

Length 7

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

Length 8

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

Length 9

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

Control Flow TRs and Test Paths—PPC

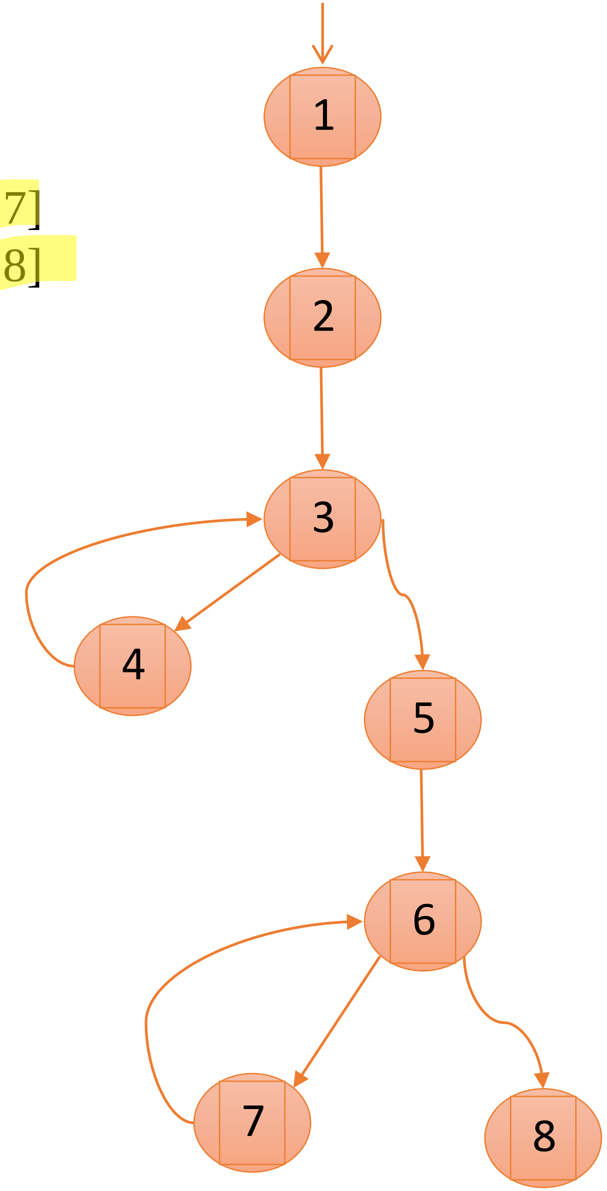
~~[1,2]~~
~~[2,3]~~
~~[3,4]~~
~~[4,3]~~
~~[3,5]~~
~~[5,6]~~
~~[6,7]~~
~~[7,6]~~
~~[6,8]~~

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

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

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

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



10 requirements are needed for Prime Paths

1. [1,2,3,5,6,7]
2. [1,2,3,5,6,8]
3. [4,3,5,6,8]
4. [4,3,5,6,7]
5. [1,2,3,4]
6. [3,4,3]
7. [4,3,4]
8. [7,6,7]
9. [7,6,8]
10. [6,7,6]

Data Flow Coverage

- Node coverage: Execute every statement
- Edge coverage: Execute every branch
- Data flow coverage: Augment the CFG
  - defs: statement that assign values to variables
  - uses : statements that use variables