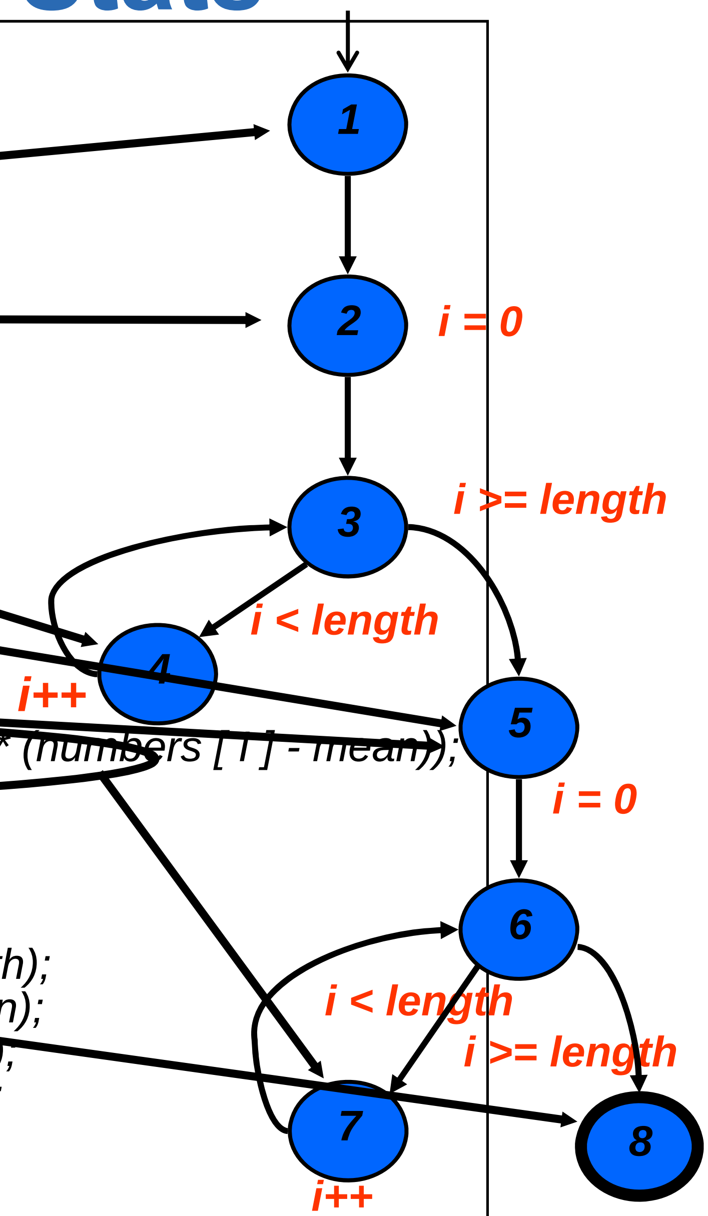
Example Control Flow – Stats

```
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);
}
```

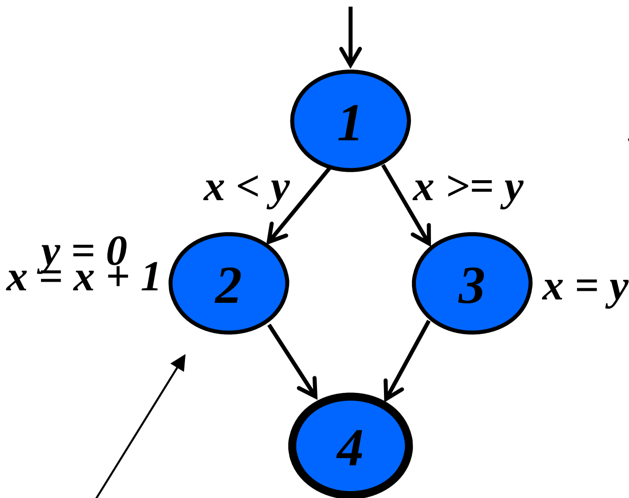
Control Flow Graph for Stats

```
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);
}
```



Statement/Basic Block Coverage

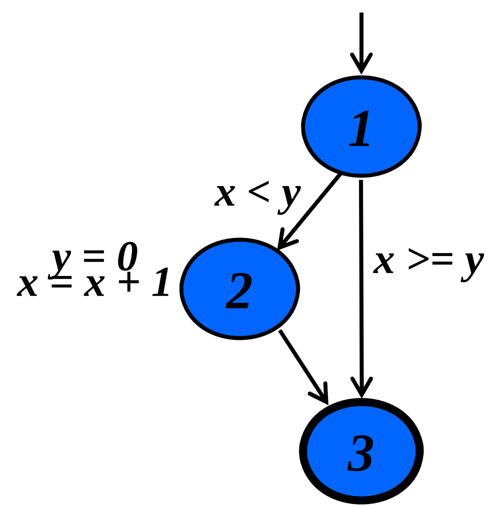
```
if (x < y)
{
  y = 0;
  x = x + 1;
}
else
{
  x = y;
}
```



Statement coverage: Cover every node of these graphs

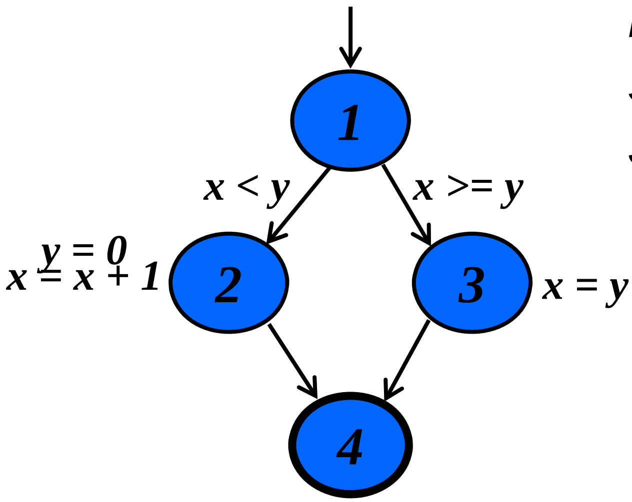
Treat as one node because if one statement executes the other must also execute (code is a basic block)

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



Branch Coverage

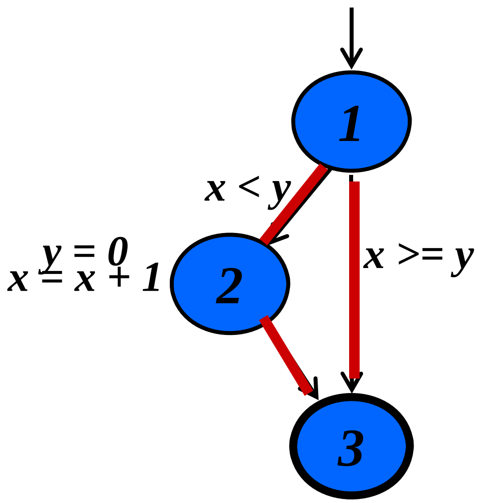
```
if (x < y)
{
  y = 0;
  x = x + 1;
}
else
{
  x = y;
}
```



Branch coverage vs. statement coverage: Same for if-then-else

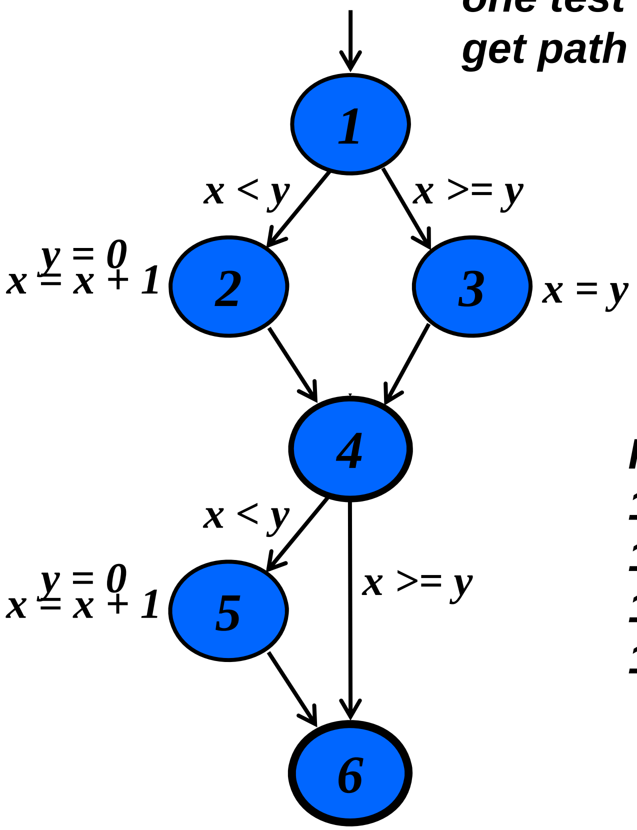
But consider this if-then structure. For branch coverage can't just cover all nodes, but must cover all edges – get to node 3 both after 2 and without executing 2!

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



Path Coverage

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

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



How many paths through this code are there? Need one test case for each to get path coverage

To get statement and branch coverage, we only need two test cases:
1 2 4 5 6 and 1 3 4 6

Path coverage needs two more:
1 2 4 5 6
1 3 4 6
1 2 4 6
1 3 4 5 6

In general: exponential in the number of conditional branches!

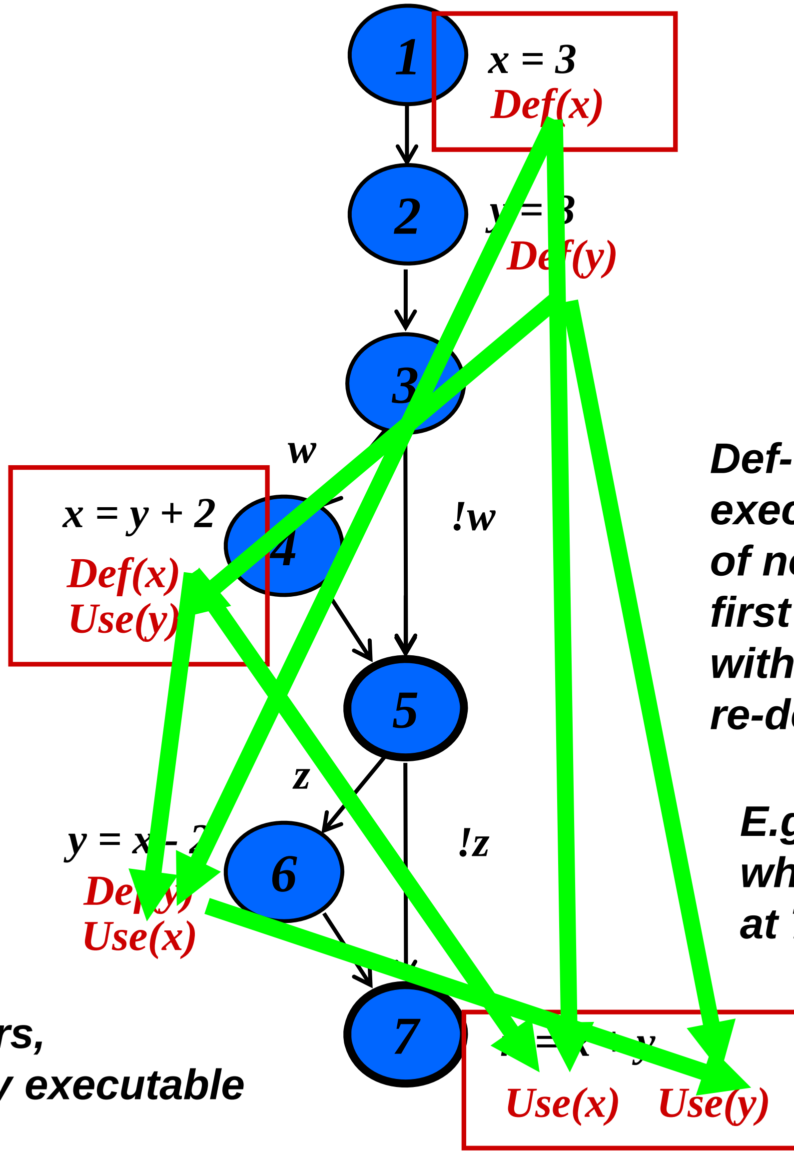
Data Flow (Def-Use) Coverage

x = 3;
y = 3;

if (w) {
x = y + 2;
}

if (z) {
y = x – 2;
}

n = x + y



Annotate program with locations where variables are defined and used (very basic static analysis)

Def-use pair coverage requires executing all possible pairs of nodes where a variable is first defined and then used, without any intervening re-definitions

E.g., this path covers the pair where x is defined at 1 and used at 7: 1 2 3 5 6 7

But this path does NOT: 1 2 3 4 5 6 7

May be many pairs, some not actually executable

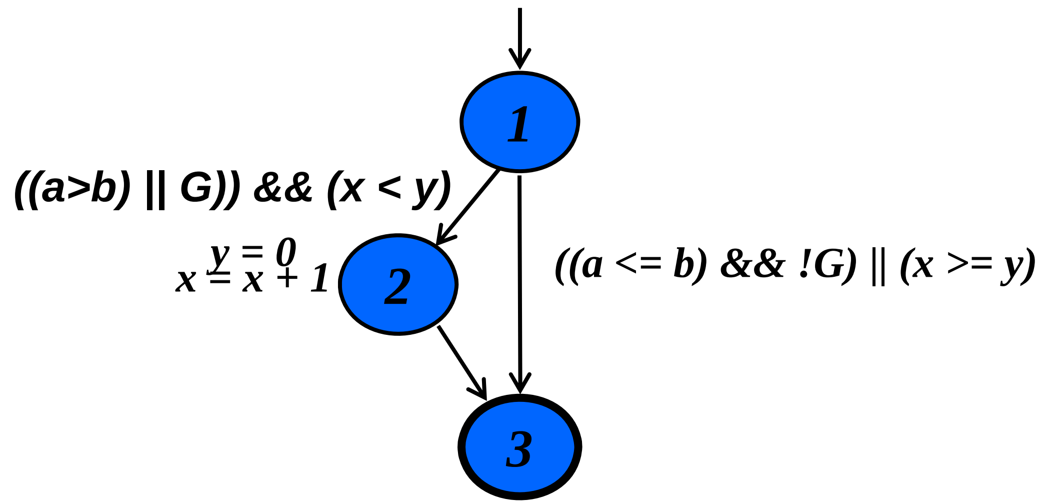
Logic Coverage

What if, instead of:

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

we have:

```
if (((a>b) || G)) && (x < y))
{
  y = 0;
  x = x + 1;
}
```



Now, branch coverage will guarantee that we cover all the edges, but does not guarantee we will do so for all the different logical reasons

We want to test the logic of the guard of the if statement

Active Clause Coverage

$((a > b) \text{ or } G) \text{ and } (x < y)$

1	T	F	T	T
2	F	F	T	F
3	F	T	T	T
4	F	F	T	F
5	T	T	T	T
6	T	T	F	F

With these values for G and (x<y), (a>b) determines the value of the predicate

With these values for (a>b) and (x<y), G determines the

With these values for (a>b) and G, (x<y) determines the value of the predicate

duplicate

8

Input Domain Partitioning

- Partition scheme *q* of domain *D*
- The partition *q* defines a set of blocks, $Bq = b_1$, b_2 , … b_Q
- The partition must satisfy two properties:
  1. blocks must be pairwise disjoint (no overlap)
  2. together the blocks cover the domain *D* (complete)

Coverage then means using at least one input from each of b_1, b_2, b_3, . . .

9

Syntax-Based Coverage

- Usually known as *mutant testing*
- Bit different kind of creature than the other coverages we've looked at
- Idea: generate many syntactic *mutants* of the original program
- Coverage: how many mutants does a test suite kill (detect)?

Syntax-Based Coverage

Program P

MUTANTS OF P

100% coverage means you kill all the mutants with your test suite

Using gcov to Collect Coverage

- GCC comes with a tool for collecting and analyzing coverage, called gcov

- Compile with some additional items:
  - -ftest-coverage -fprofile-arcs

- When the executable runs, it will produce files (gcda files) that record how often each line ran

Using gcov to Collect Coverage

- To look at the coverage, type:
  - gcov <sourcefile>
  - Will show % coverage, and produce <sourcefile>.gcov, annotated copy of code
- Can also do branch coverage:
  - gcov –b <sourcefile>
- Makefiles from this class automatically compile with gcov

Using gcov to Collect Coverage

- Important points:
  - If you compile with optimization, results may be strange – try -O0
  - If you haven't run the program, gcov <sourcefile> won't do anything! It has no coverage data
  - The number of times a line/branch runs can be helpful, in addition to looking for "####" to indicate things that are never covered at all
    - `grep '####' filename.c.gcov`