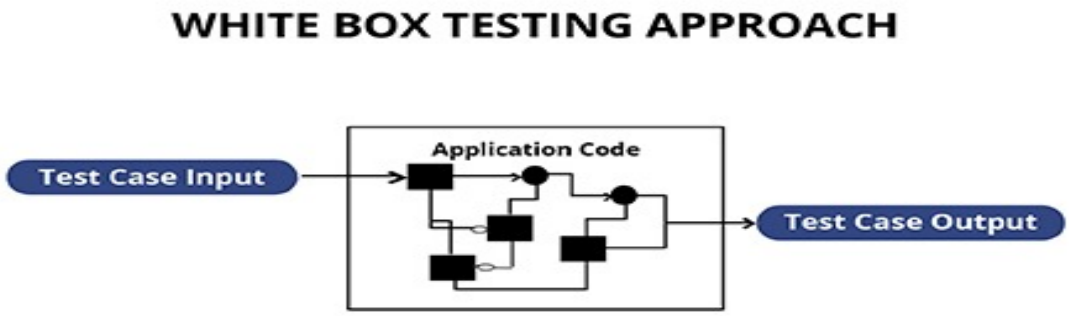
White Box Testing

Software Testing

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

White Box Testing

- A strategy testing based on the internal paths, structure, and implementation of the System Under Test
- Can be applied at all levels of system development - unit, integration, and system



White Box Testing Techniques

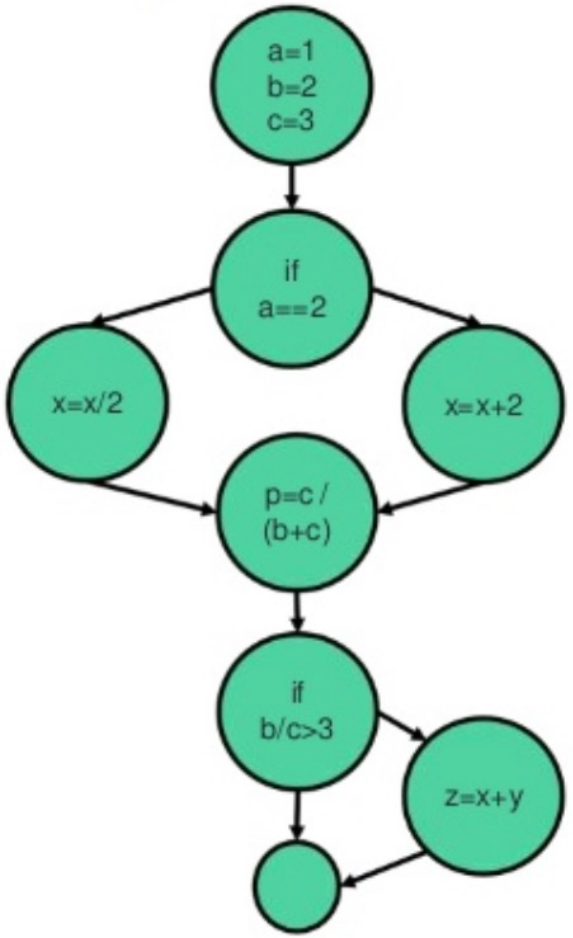
- Control Flow Testing
  - Identify the **execution paths** through a module of **program code**
- Data Flow Testing
  - Identify paths in the program that go from the **assignment** to the **use** of a **variable** in a module of program code

Control Flow Testing

- Create and executes test cases to **cover the execution paths** through a module or program code
- **Path**: *a sequence of statement execution that begins at an entry and ends at an exit*

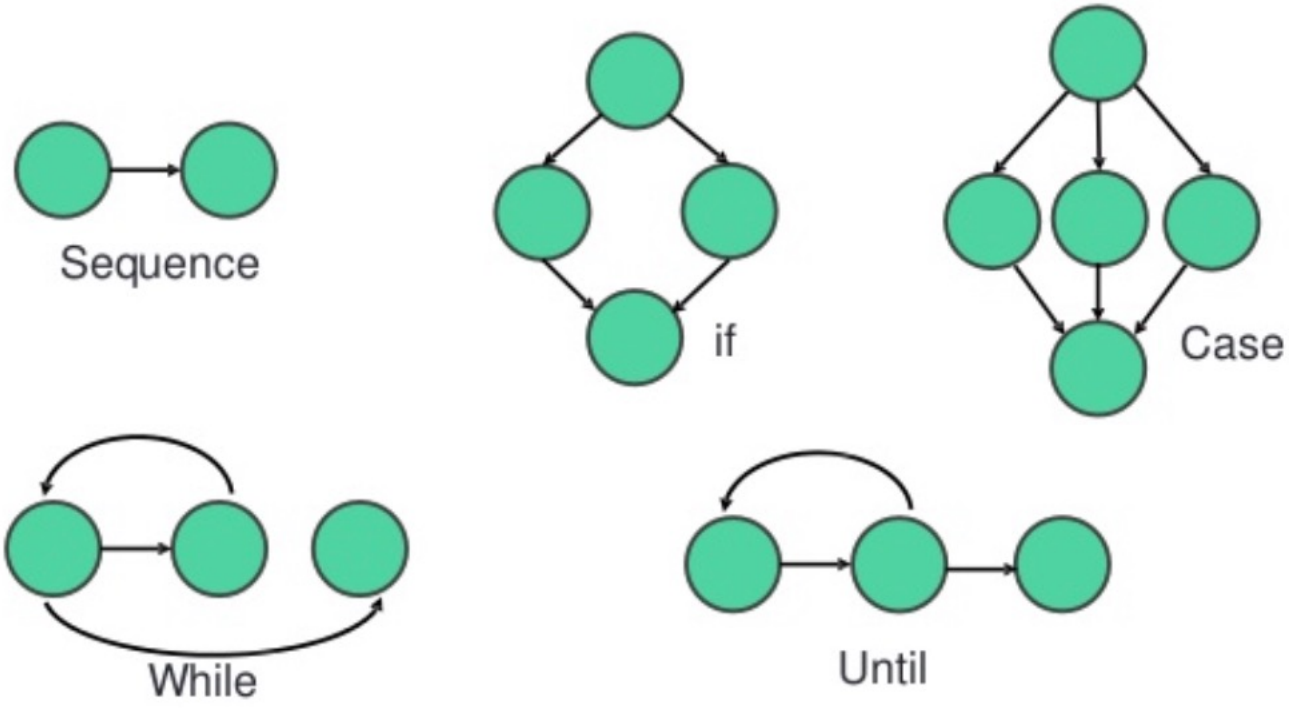
Technique: Control Flow Graph

```
a = 1;
b = 2;
c = 3;
if(a == 2){
      x = x + 2;
}
else {
      x = x / 2;
}
p = c /(b + c);
if(b/c > 3){
      z = x + y;
}
```



How many test cases?

Technique: Control Flow Graph



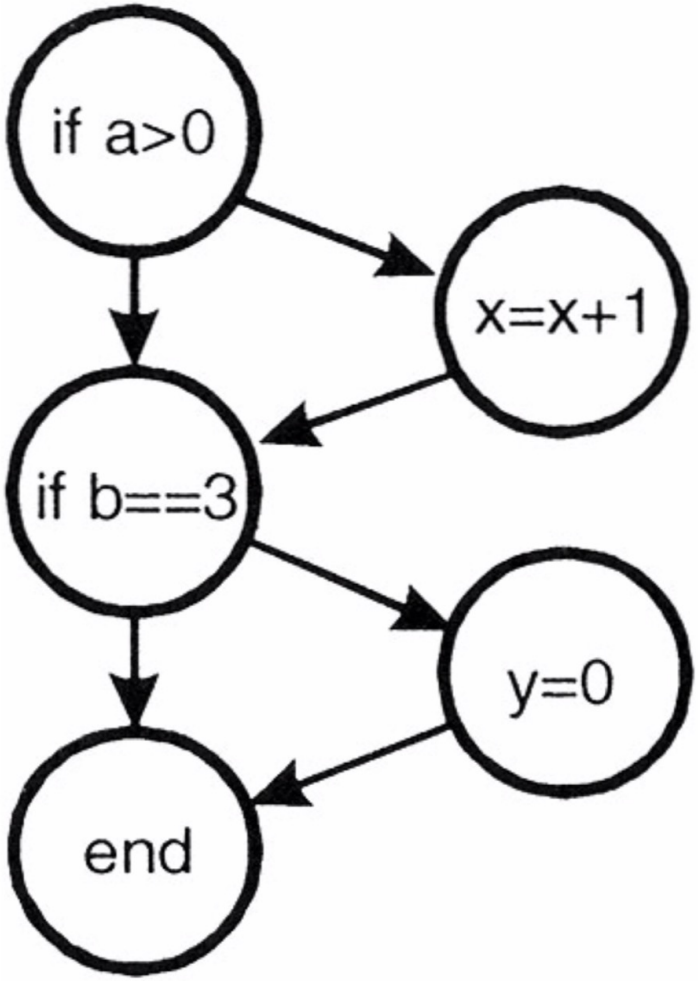
Code Coverage

The percentage of the code that has been tested

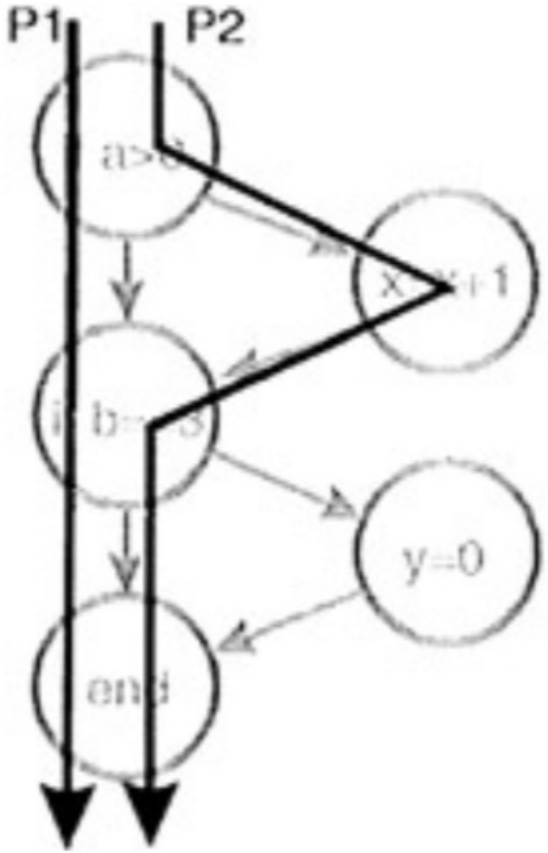
1. Statement Coverage
2. Branch/Decision Coverage
3. Condition Coverage
4. Multiple Condition Coverage
5. Path Coverage

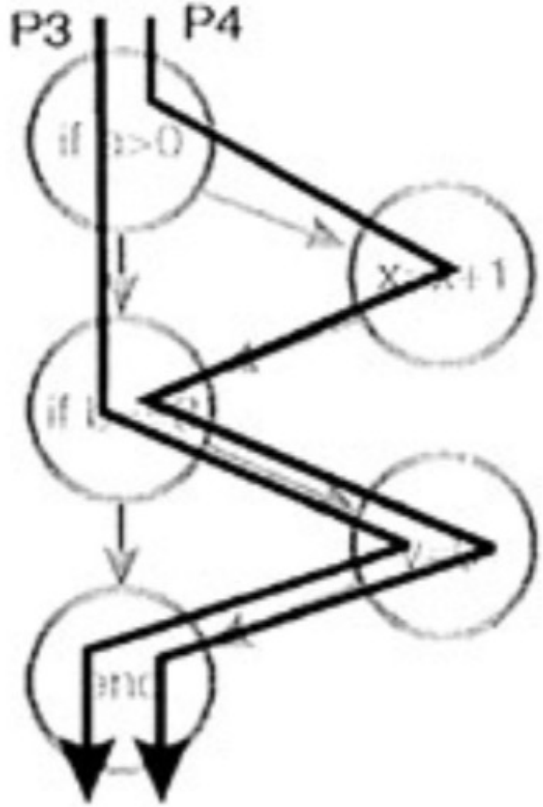
Example

```
1    if (a > 0) {
2        x = x + 1;
3    }
4    if (b == 3) {
5        y = 0;
6    }
```



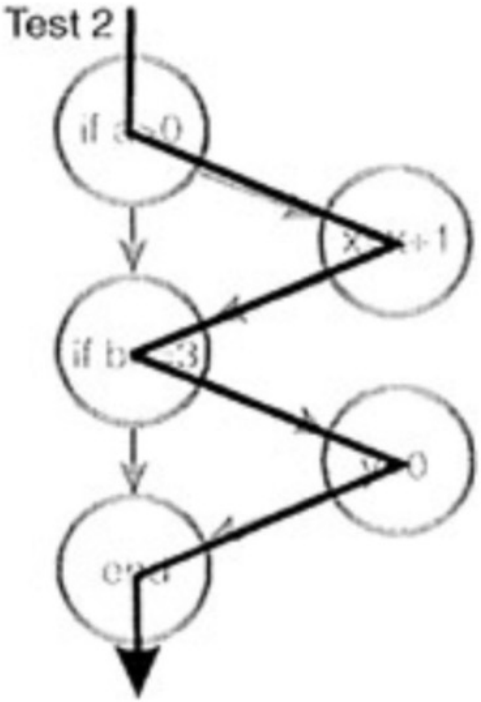
Execution paths





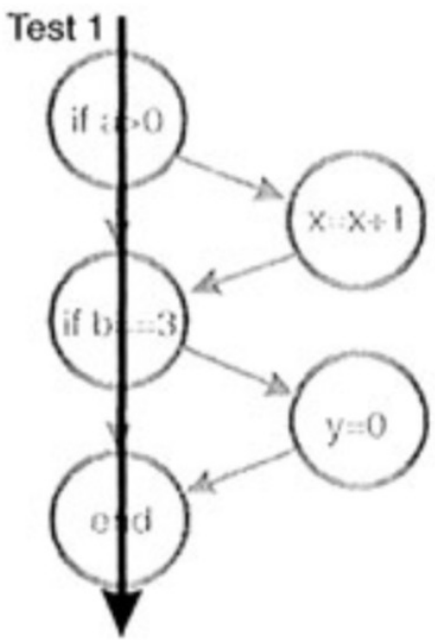
Statement Coverage

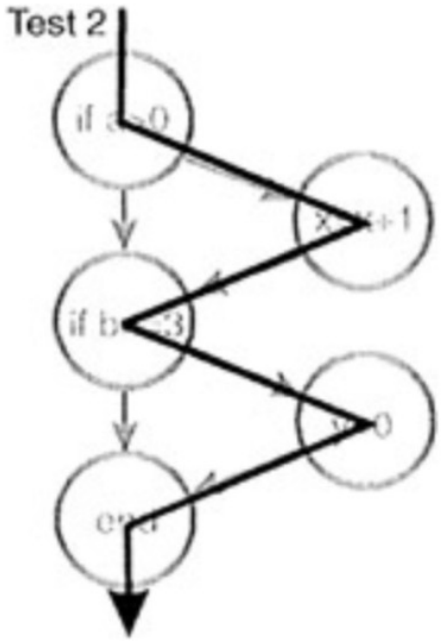
- Every statement is tested at least one
- 1 test case: 100% Statement Coverage
  - a=6, b=3



Branch/Decision Coverage

- Each decision that has a TRUE and FALSE outcome is evaluated at least one
- 2 test cases: 100% Decision Coverage
  - a=0, b=2
  - a=4, b=3





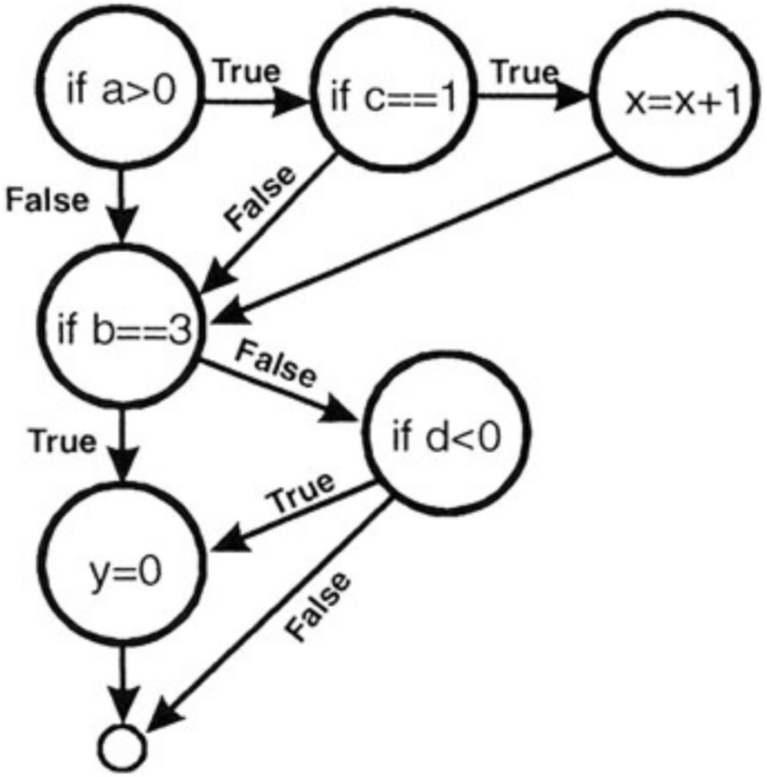
Condition Coverage

- Each condition that has a TRUE and FALSE outcome that makes up a decision is evaluated at least one
- 2 test cases:
  - a=1, c=1, b=3, d=-1
  - a=0, c=2, b=2, d=0

```
1    if (a > 0 && c == 1) {
2        x = x + 1;
3    }
4    if (b == 3 || d < 0) {
5        y = 0;
6    }
```

Multiple Condition Coverage

```
1    if (a > 0 && c == 1) {
2        x = x + 1;
3    }
4    if (b == 3 || d < 0) {
5        y = 0;
6    }
```



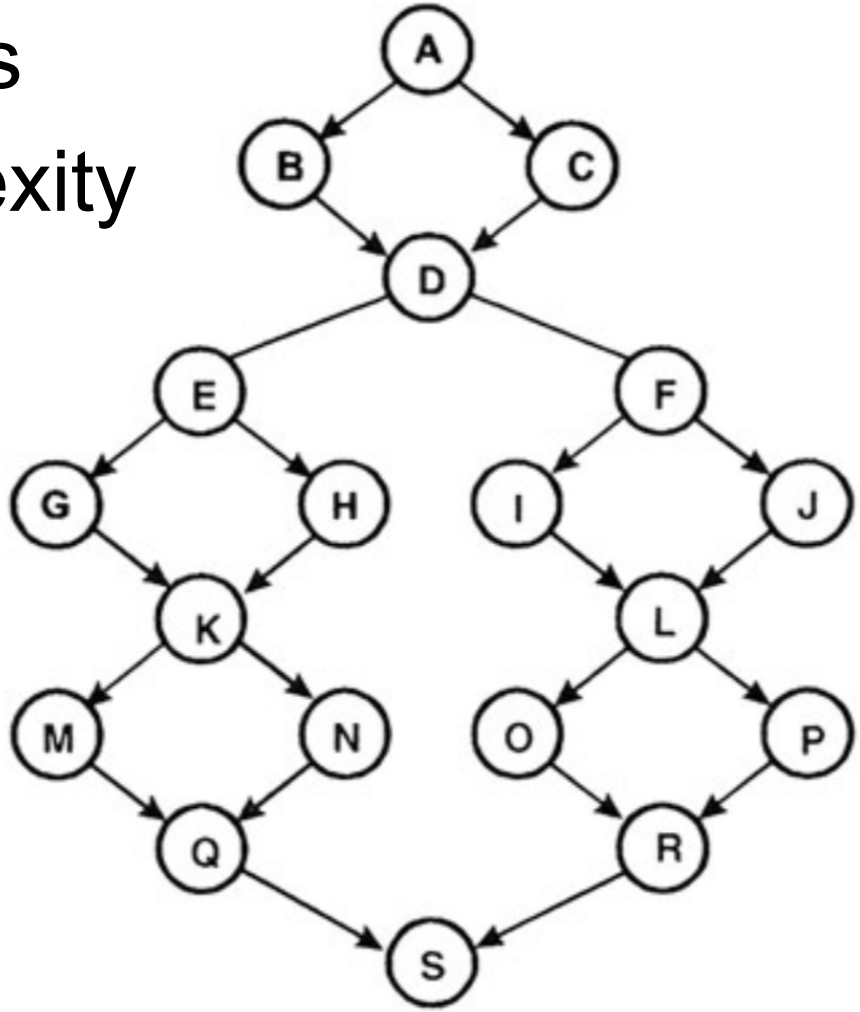
- 4 test cases:
  - a=1, c=1, b=3, d=-1
  - a=0, c=1, b=3, d=0
  - a=1, c=2, b=2, d=-1
  - a=0, c=2, b=2, d=0

Path Coverage

- Structure Testing / Basic Path Testing
  1. Derive the control flow graph
  2. Compute the graph's Cyclomatic Complexity
     - C = edges – nodes + 2
  3. Select a set of C basis paths
  4. Create a test case for each basis path
  5. Execute these tests

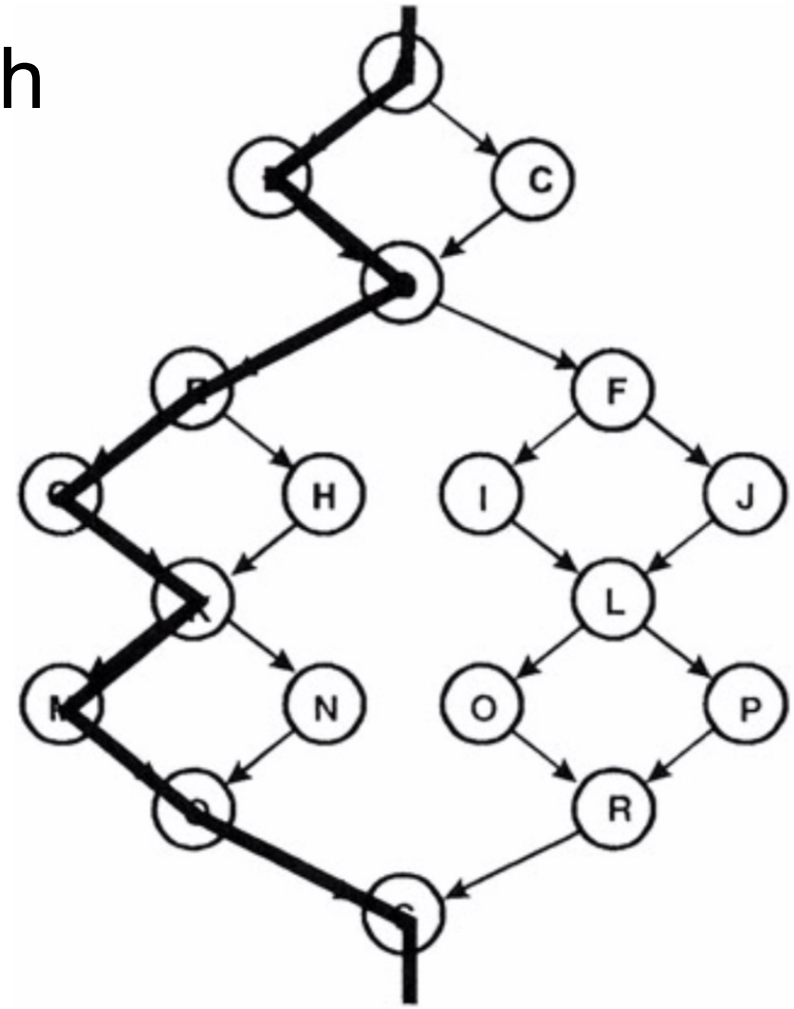
Basic Path Testing

- 24 edges, 19 nodes
- Cyclomatic Complexity
  - 24 – 19 + 2 = 7



Create a set of basis paths

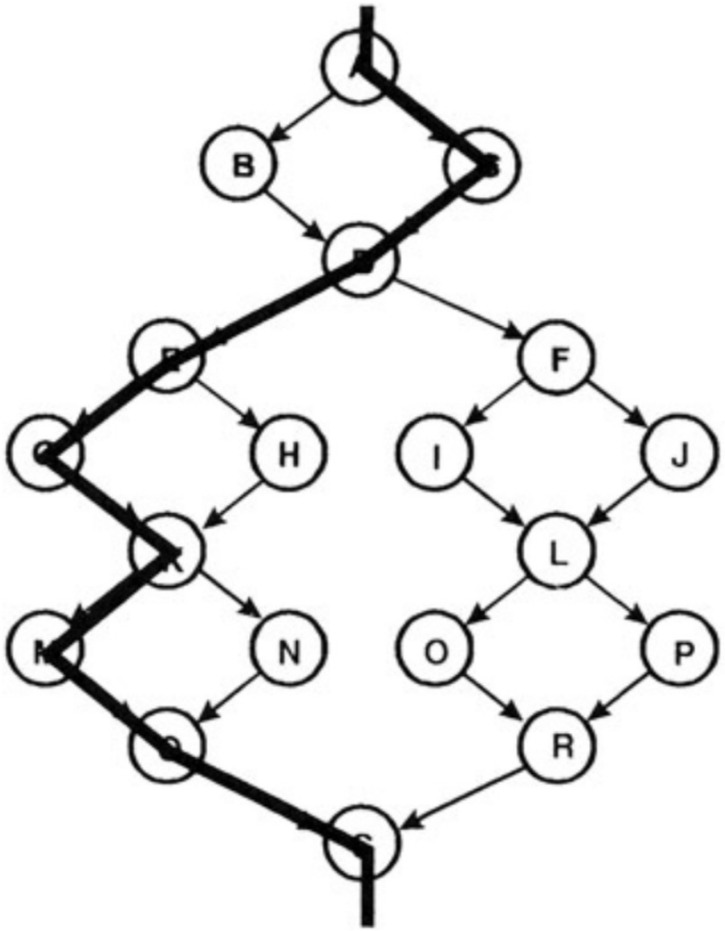
1. Pick a “baseline” path
   - ABDEGKMQS



Choose the next path

2. Change the outcome of the first decision along the baseline path while keeping the maximum number of other decisions the same

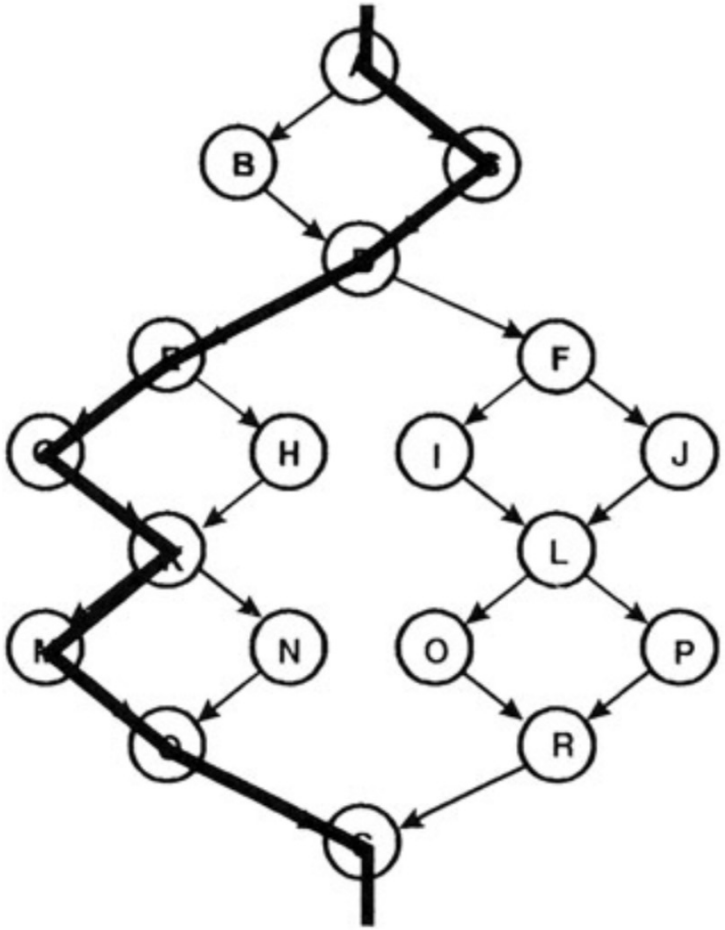
- ACDEGKMQS



Generate the third path

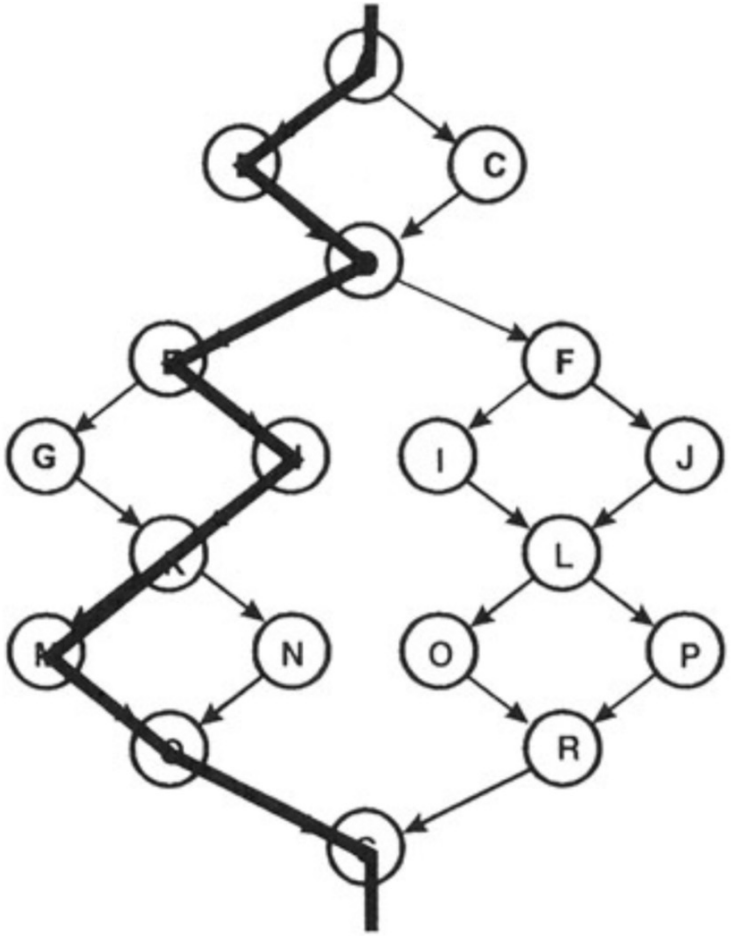
3. Begin again with the baseline but vary the second decision rather than the first

- ABDFILORS

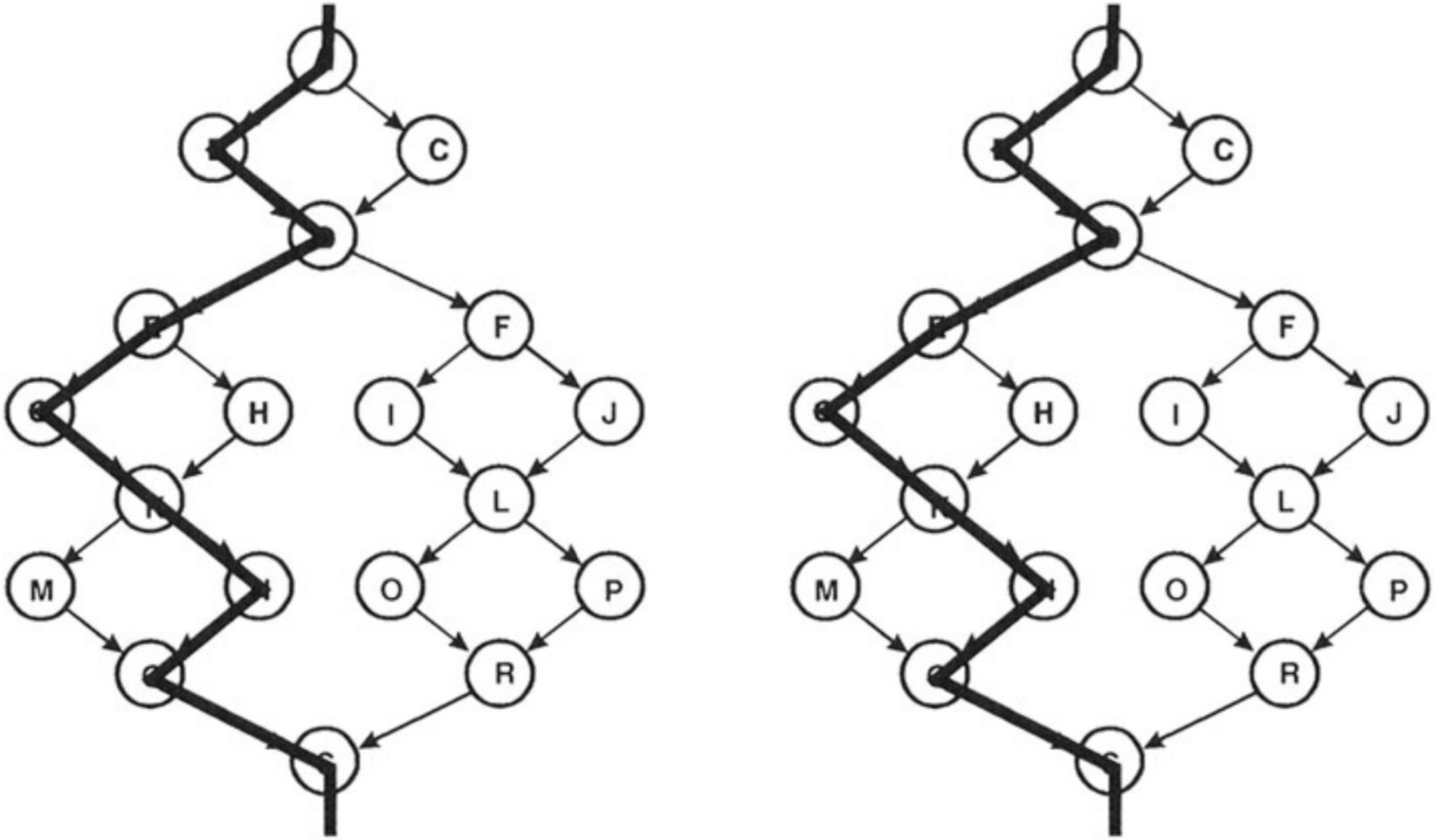


Generate the next paths

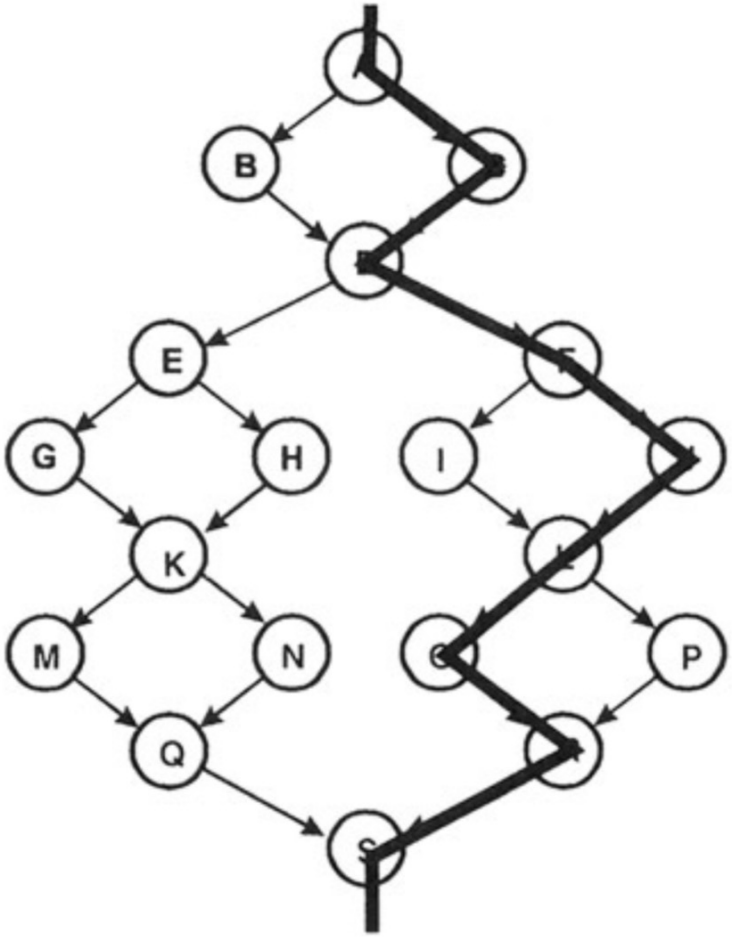
- Continue varying each decision, one by one, until the bottom of the graph is reached
- ABDEHKMQS



Generate the next paths



Generate the next paths



ABDEGKMQS

ACDEGKMQS

ABDFILORS

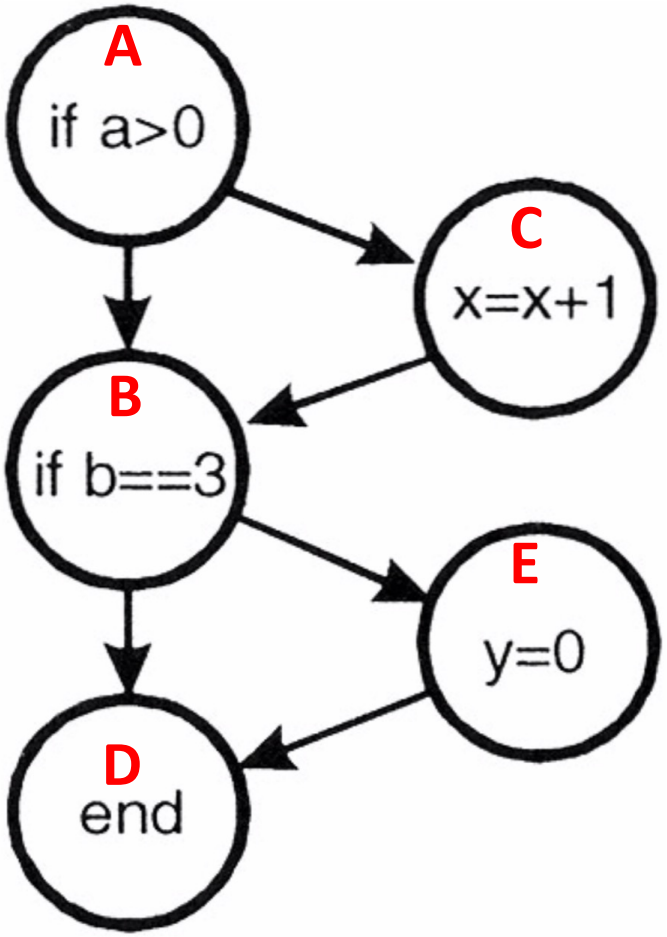
ABDEHKMQS

ABDEGKNQS

ACDFJLORS

ACDFILPRS

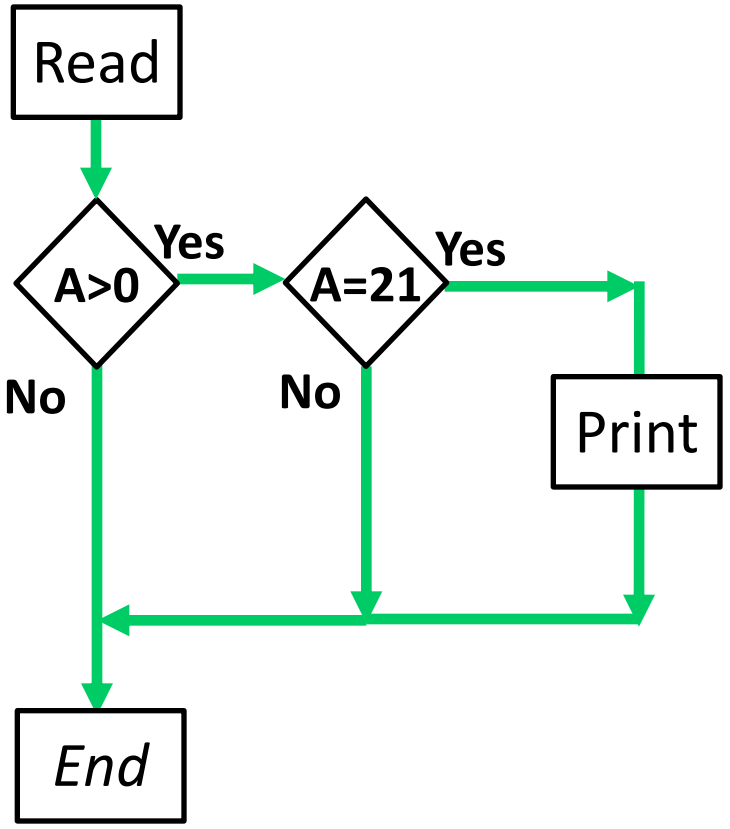
Example



- 6 edges, 5 nodes
- Cyclomatic Complexity
  - 6 – 5 + 2 = 3
- 3 basis paths
  - ABD
  - ACBD
  - ABED
- 3 Test cases
  - ABD: a=0, b=2
  - ACBD: a=1, b=2
  - ABED: a=0, b=3

Example

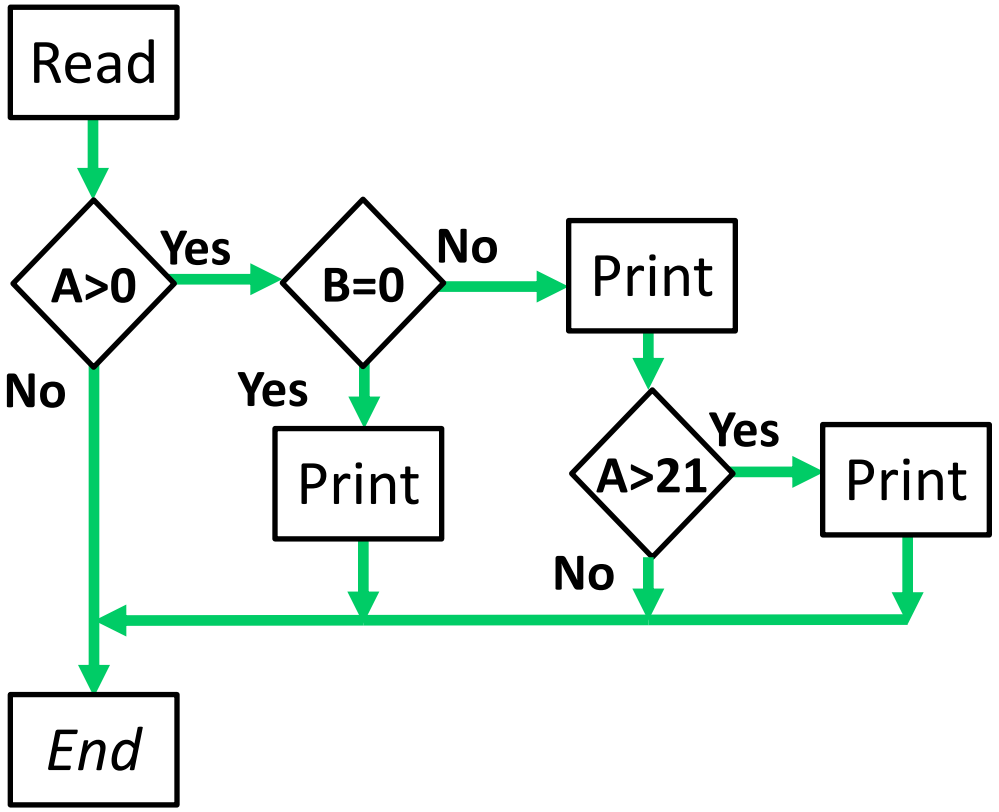
```
Read A
IF A > 0 THEN
    IF A  = 21 THEN
     Print “Key”
    ENDIF
ENDIF
```



- Cyclomatic complexity: _____
- Minimum tests to achieve:
  - Statement coverage: _____
  - Branch coverage: _____

Example

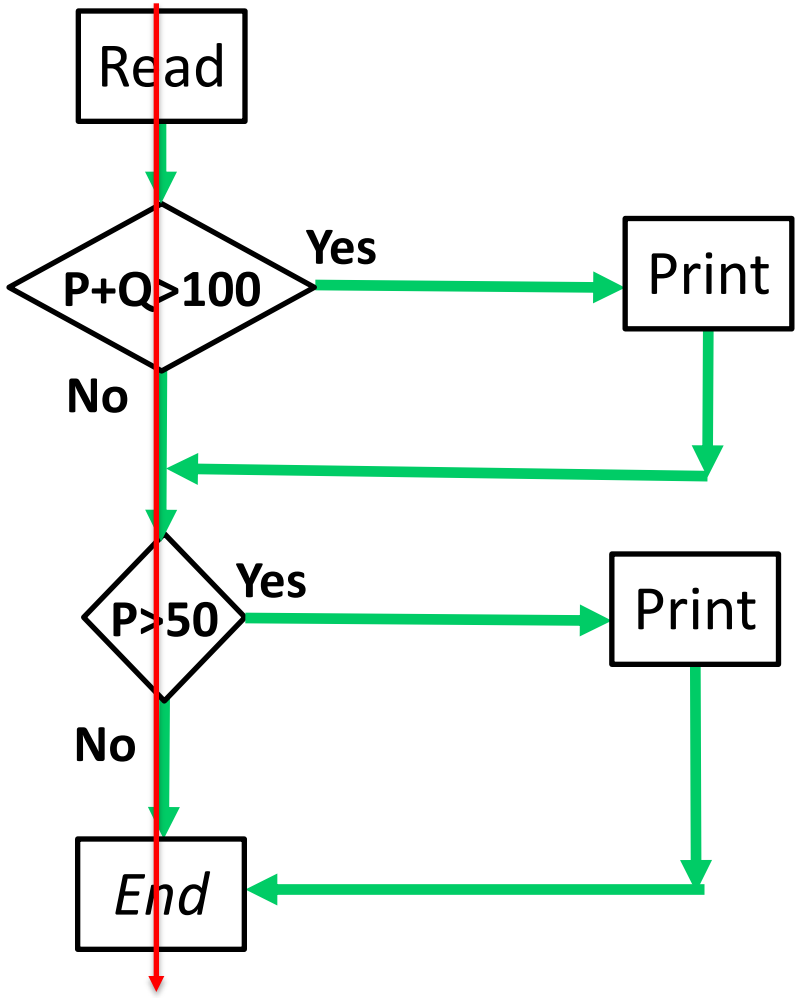
```
Read A
Read B
IF A > 0 THEN
    IF B  = 0 THEN
     Print “No values”
    ELSE
     Print B
     IF A > 21 THEN
        Print A
     ENDIF
    ENDIF
ENDIF
```



- Cyclomatic complexity: _____
- Minimum tests to achieve:
  - Statement coverage: _____
  - Branch coverage: _____

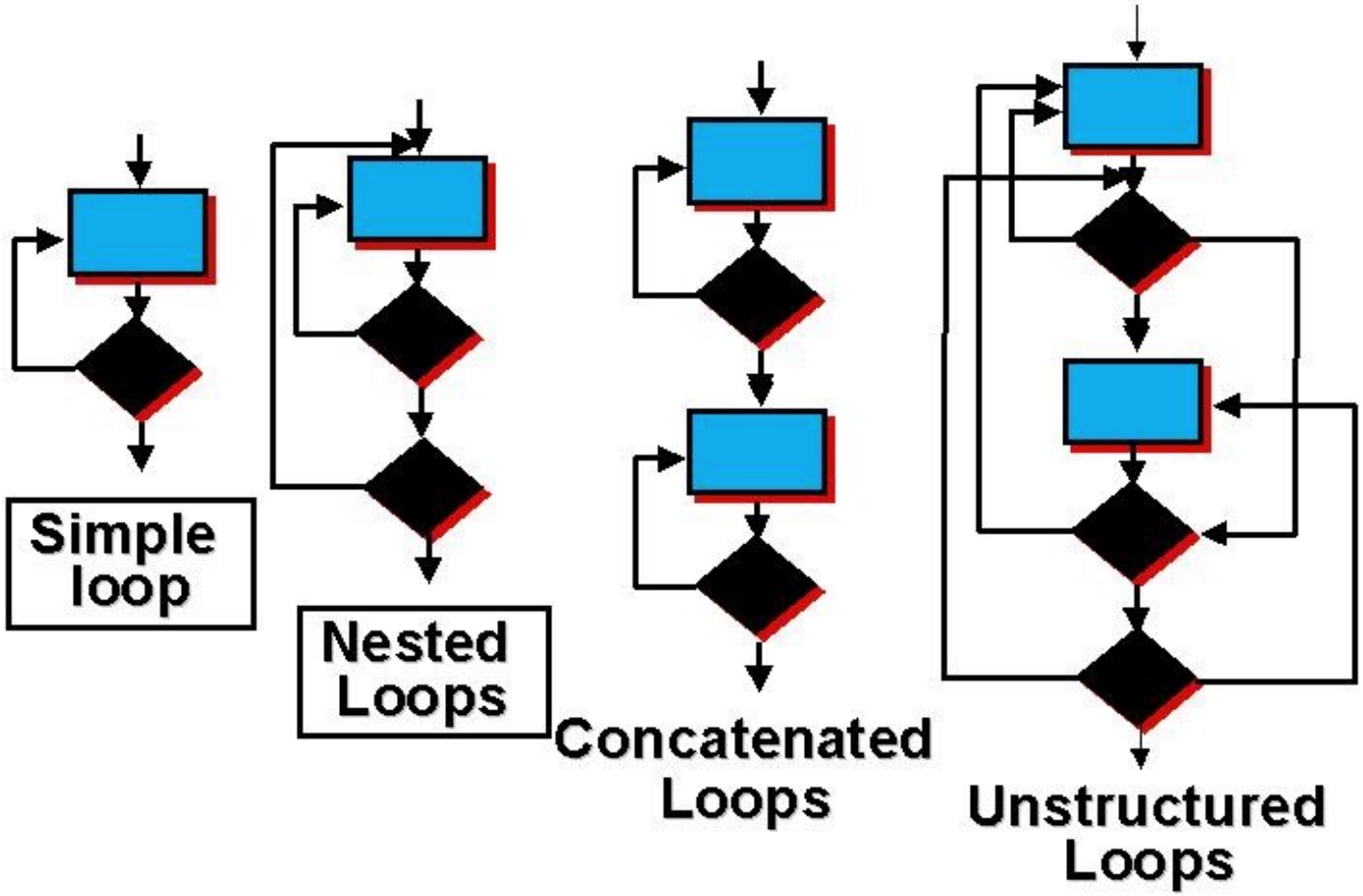
Example

```
Read P
Read Q
IF P+Q > 100 THEN
Print “Large”
ENDIF
If P > 50 THEN
Print “P Large”
ENDIF
```



- Cyclomatic complexity: _____
- Minimum tests to achieve:
  - Statement coverage: _____
  - Branch coverage: _____

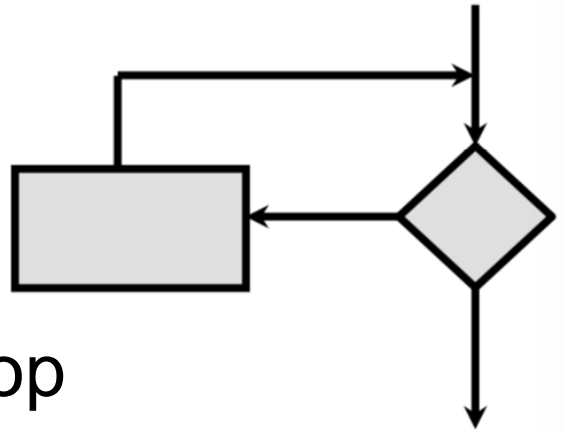
Loop Testing



Loop Testing: Simple Loops

- Minimum conditions – simple loops
  1. **Skip** the loop entirely
  2. Only **one pass** through the loop
  3. **Two passes** through the loop
  4. **m passes** through the loop m < n
  5. **(n-1), n, and (n+1) passes** through the loop

  Where n is the maximum number of allowable passes



Loop Testing

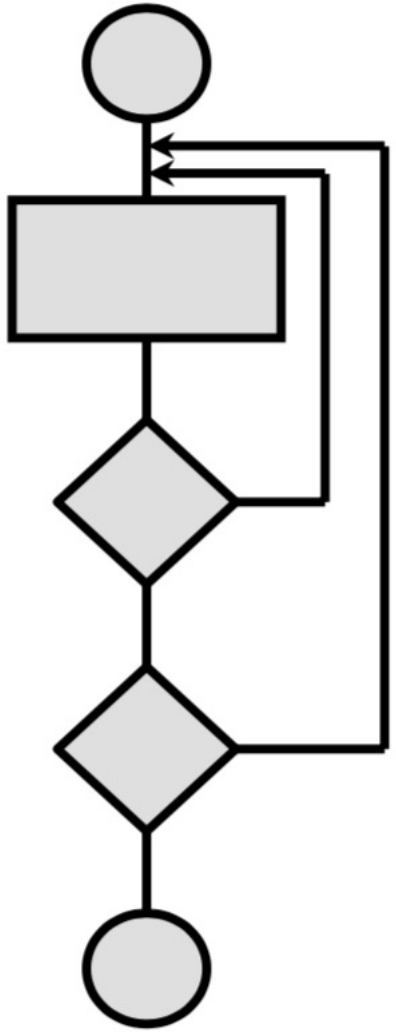
```
public class loopdemo
{
    private int[] numbers = {5,-3,8,-12,4,1,-20,6,2,10};

    /** Compute total of numItems positive numbers in the array
     *  @param numItems how many items to total, maximum of 10.
     */
    public int findTotal(int numItems)
    {
        int total = 0;
        if (numItems <= 10)
        {
            for (int count=0; count < numItems; count = count + 1)
            {
                if (numbers[count] > 0)
                {
                    total = total + numbers[count];
                }
            }
        }
        return total;
    }
}
```

numItems
0
1
2
5
9
10
11

Nested Loops

- Extend simple loop testing
- Reduce the number of tests
  - start at the innermost loop; set all other loops to minimum values
  - conduct simple loop test; add out of range or excluded values
  - work outwards while keeping inner nested loops to typical values
  - continue until all loops have been tested



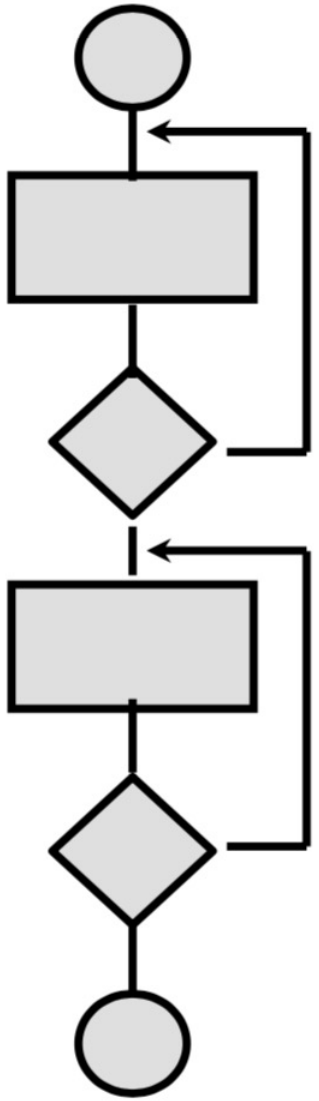
Concatenated Loops

- Independent Loops

=> Test as simple loops

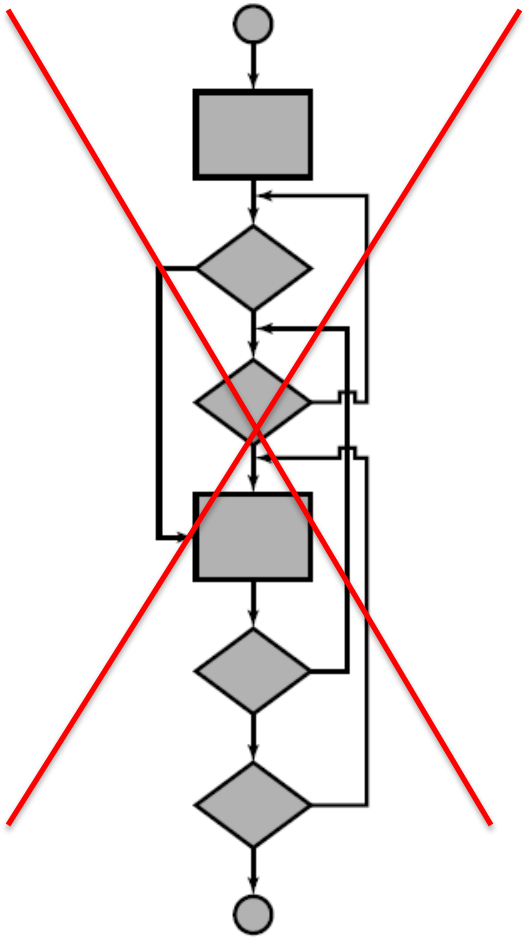
- Dependent Loops

=> Test as nested loops



Unstructured Loops

- DON'T test
- Re-design



Data Flow Testing: Definition

- Def – assigned or changed
- Uses – utilized (not changed)
  - C-use (Computation): right-hand side of an assignment, an index of an array, parameter of a function
  - P-use (Predicate): branching the execution flow (if, while, for statement)

Data Flow Testing

- Def-Use testing
  - All navigation paths from every definition of a variable to every use of it is exercised
- All-Use testing
  - At least one navigation path from every definition of a variable to every use of it is exercised

Data Flow Testing

1	sum = 0	*sum, def*
2	read (n),	*n, def*
3	i = 1	*i, def*
4	while (i <= n)	*i, n p-sue*
5	read (number)	*number, def*
6.	sum = sum + number	*sum, def, sum, number, c-use*
7	i = i + 1	*i, def, c-use*
8	end while	
9	print (sum)	*sum, c-use*

Def-Use Testing

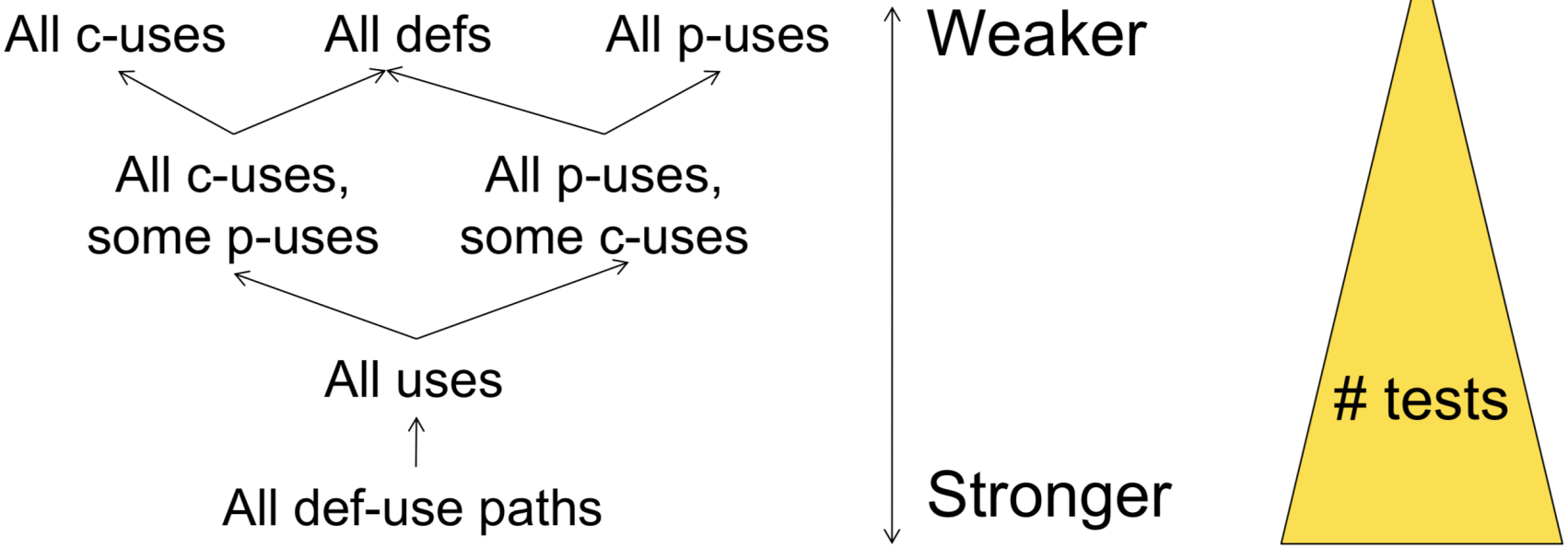
Table for sum

pair id	def	use
1	1	6
2	1	9
3	6	6
4	6	9

Table for i

pair id	def	use
1	3	4
2	3	7
3	7	7
4	7	4

Data Flow Criteria



White box Testing Disadvantages

1. The number of execution paths may be so large that they cannot all be tested
2. The chosen test cases may not detect data sensitivity errors
   - p = q / r
   - May execute correctly except when r=0
3. The tests are based on the existing paths, nonexistent paths cannot be discovered
4. Testers must have the programming skills