

# SOFTWARE ENGINEERING

CSE 470 –Control Flow Graph (Path  
Based Testing )

BRAC University



Inspiring Excellence

# Control Flow Graph: Introduction

---

- An abstract representation of a structured program/function/method.
- Consists of two major components:
  - ▮ *Node*:
    - Represents a stretch of sequential code statements with no branches.
  - ▮ *Directed Edge* (also called *arc*):
    - Represents a branch, alternative path in execution.
- **Path**:
  - ▮ A collection of *Nodes* linked with *Directed Edges*.

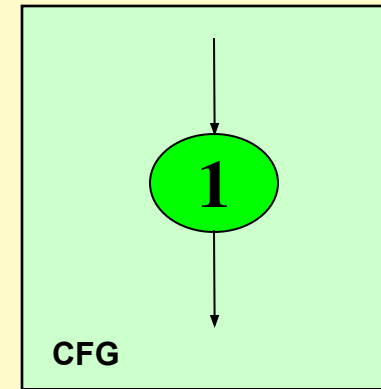
# Notation Guide for CFG

- A CFG should have:
  - 1 entry arc (known as a directed edge, too).
  - 1 exit arc.
- All nodes should have:
  - At least 1 entry arc.
  - At least 1 exit arc.
- **A Logical Node** that does not represent any actual statements can be added as a joining point for several incoming edges.
  - Represents a logical closure.
  - Example:
    - Node 4 in the `if-then-else` example in next slides

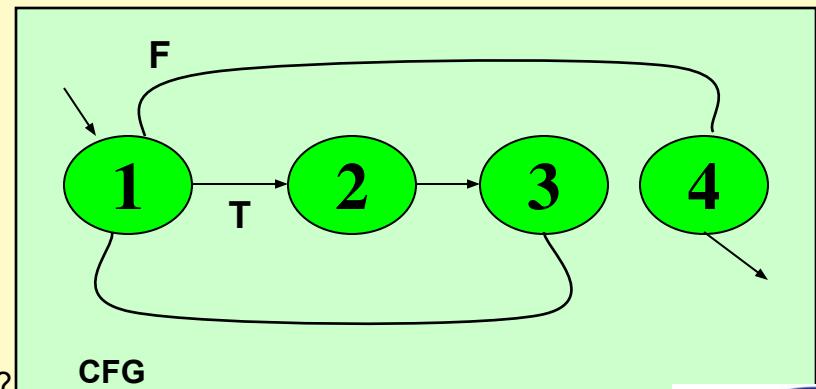
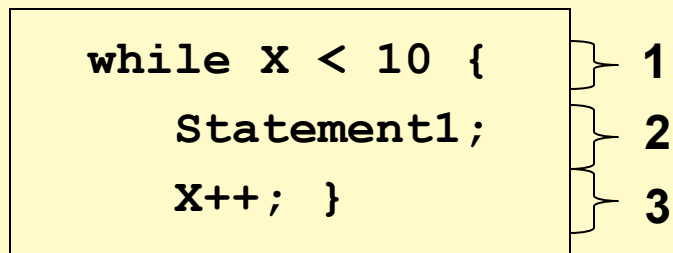
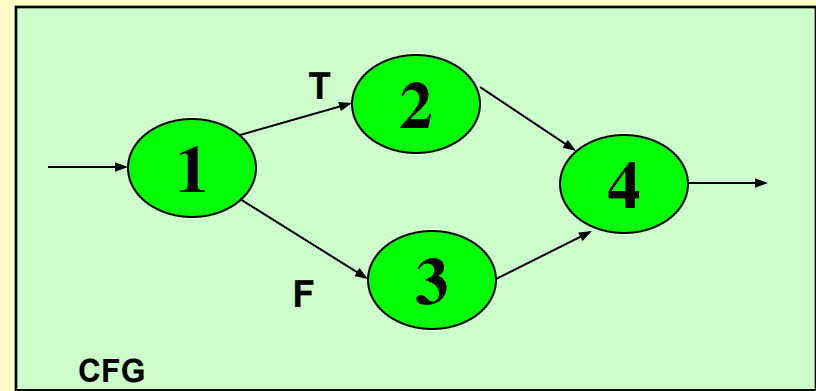
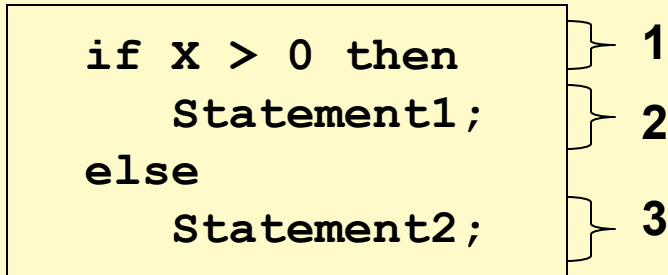
# Simple Examples

```
Statement1;  
Statement2;  
Statement3;  
Statement4;
```

Can be  
represented as  
**one** node as there  
is no branch.



# More Examples

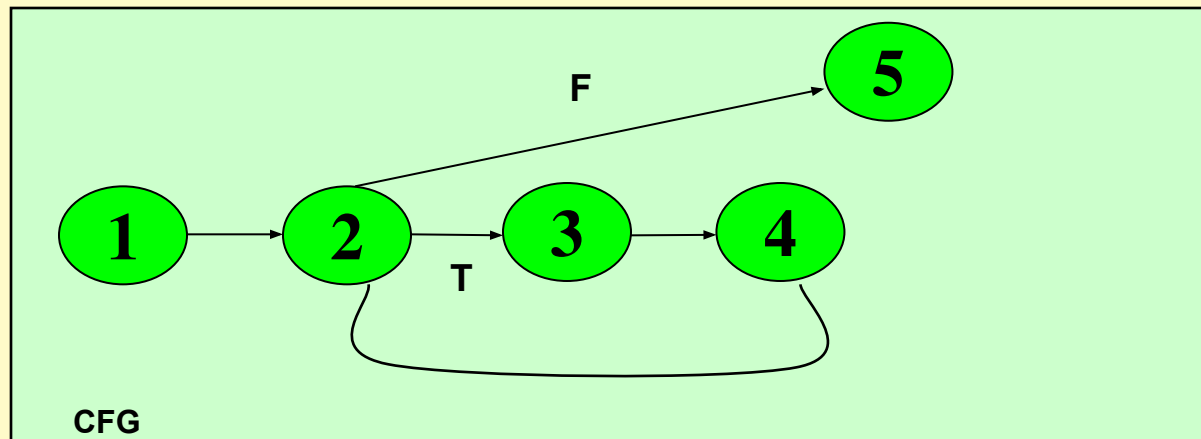


Question: Why is there a node 4 in both CFGs?

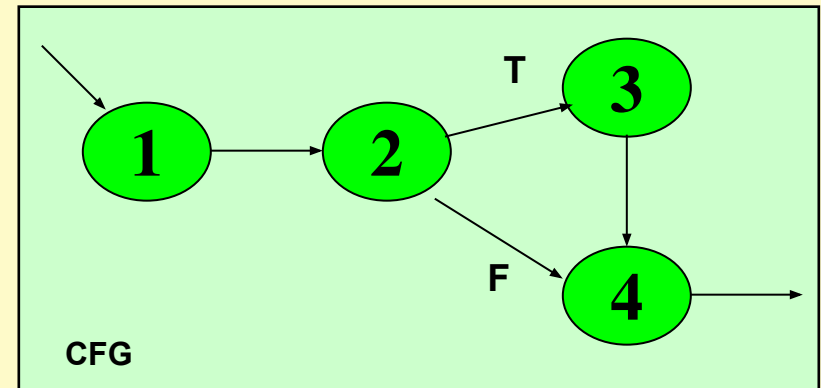
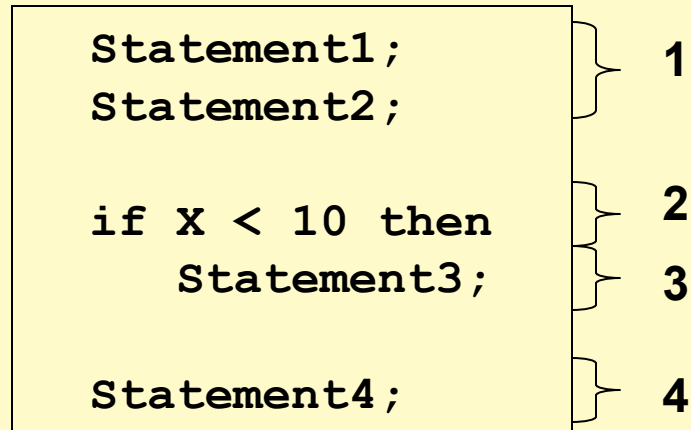
Answer: A logical node

# More Examples

```
      1      2      4
for (int I = 0; I < 10 ; I ++) {
  Statement1; }
  Statement2; } 3
  Statement3; }
}
Statement4; } 5
```



# Combined Examples



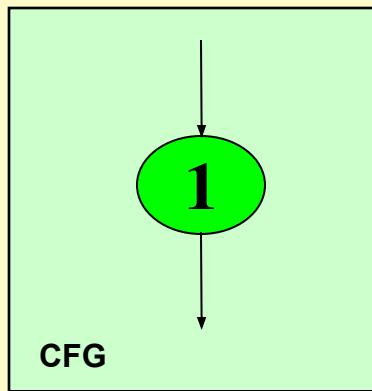
# Number of Paths through CFG

---

- ❑ Given a program, how do we exercise all statements and branches at least once?
- ❑ Translating the program into a CFG, an equivalent question is:
  - ❑ Given a CFG, how do we cover all arcs and nodes at least once?
- ❑ Since a path is a trail of nodes linked by arcs, this is similar to ask:
  - ❑ Given a CFG, what is the set of paths that can cover all arcs and nodes?

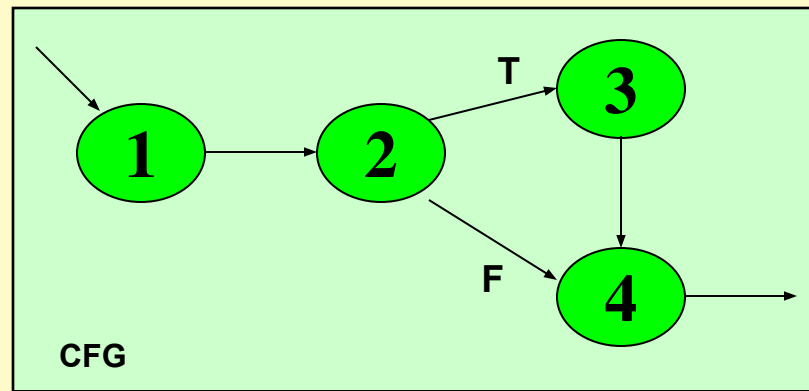


# Example



Only **one** path is needed:

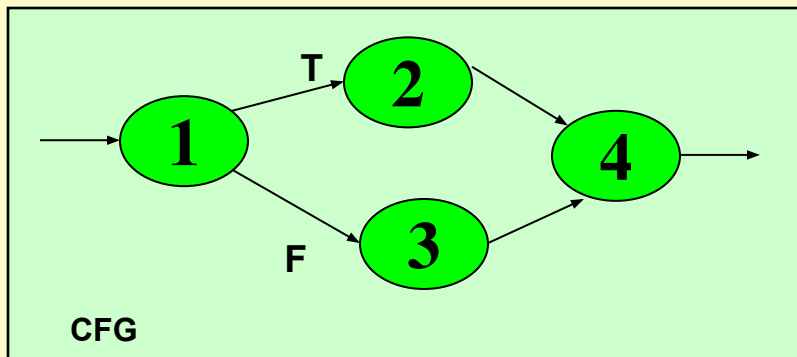
[ 1 ]



Two paths are needed:

[ 1 - 2 - 4 ]

[ 1 - 2 - 3 - 4 ]



Two paths are needed:

[ 1 - 2 - 4 ]

[ 1 - 3 - 4 ]

# White Box Testing: Path Based

---

- A generalized technique to find out the number of paths needed (known as *cyclomatic complexity*) to cover all arcs and nodes in CFG.
- Steps:
  1. Draw the CFG for the code fragment.
  2. Compute the *cyclomatic complexity number* **C**, for the CFG.
  3. Find at most **C** paths that cover the nodes and arcs in a CFG, also known as **Basic Paths Set**;
  4. Design test cases to force execution along paths in the **Basic Paths Set**.

# Path Based Testing: Step 1

```
min = A[0];
```

```
I = 1;
```

```
while (I < N) {
```

```
    if (A[I] < min)
```

```
        min = A[I];
```

```
        I = I + 1;
```

```
}
```

```
print min
```

1

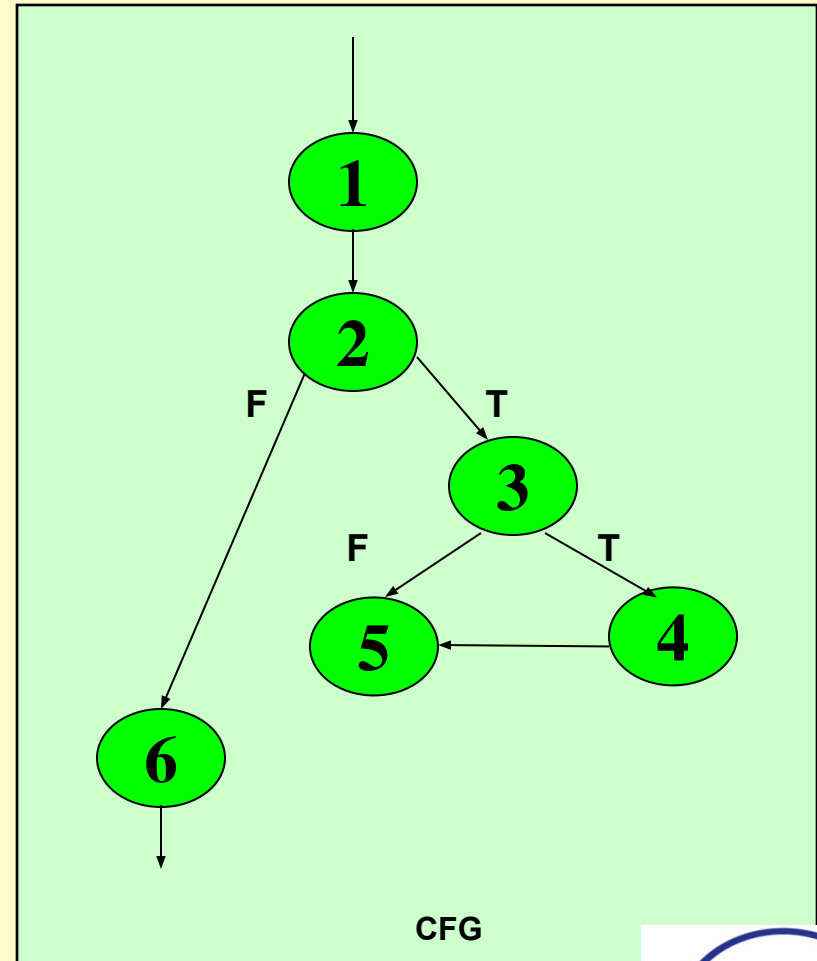
2

3

4

5

6



# Path Base Testing: Step 2

---

1. The complexity  $M$  is then defined as

$$M = R + I,$$

where  $R$  = the number of regions in the graph.

2. The complexity  $M$  is then defined as

$$M = P + I,$$

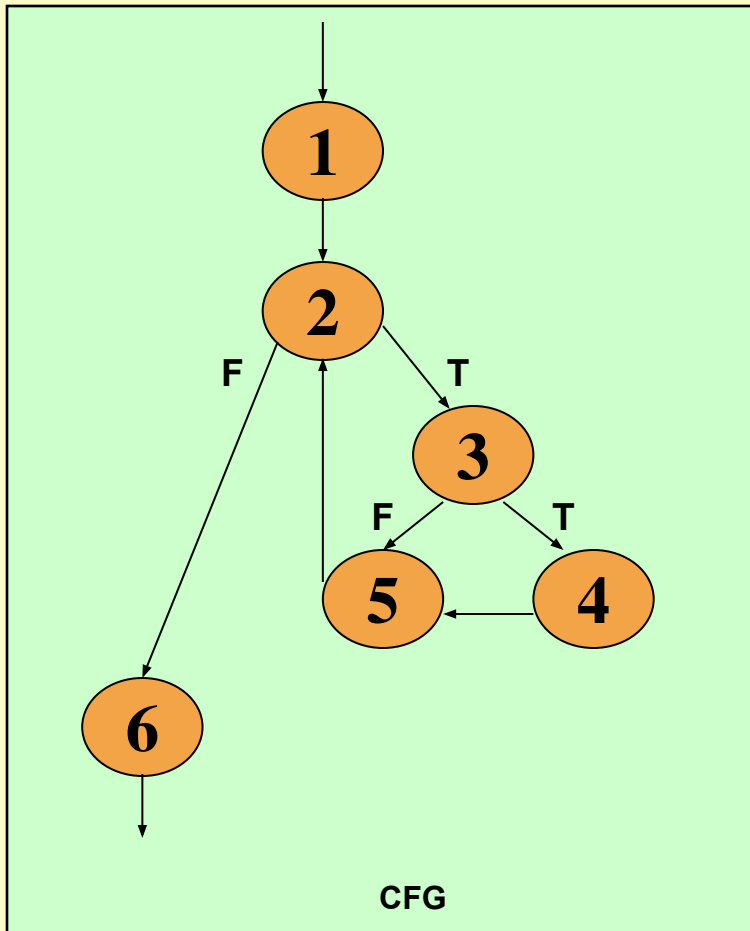
where  $P$  = the number of predicate nodes in the graph.

3. The complexity  $M$  is then defined as

$$M = E - N + 2P, \text{ where}$$

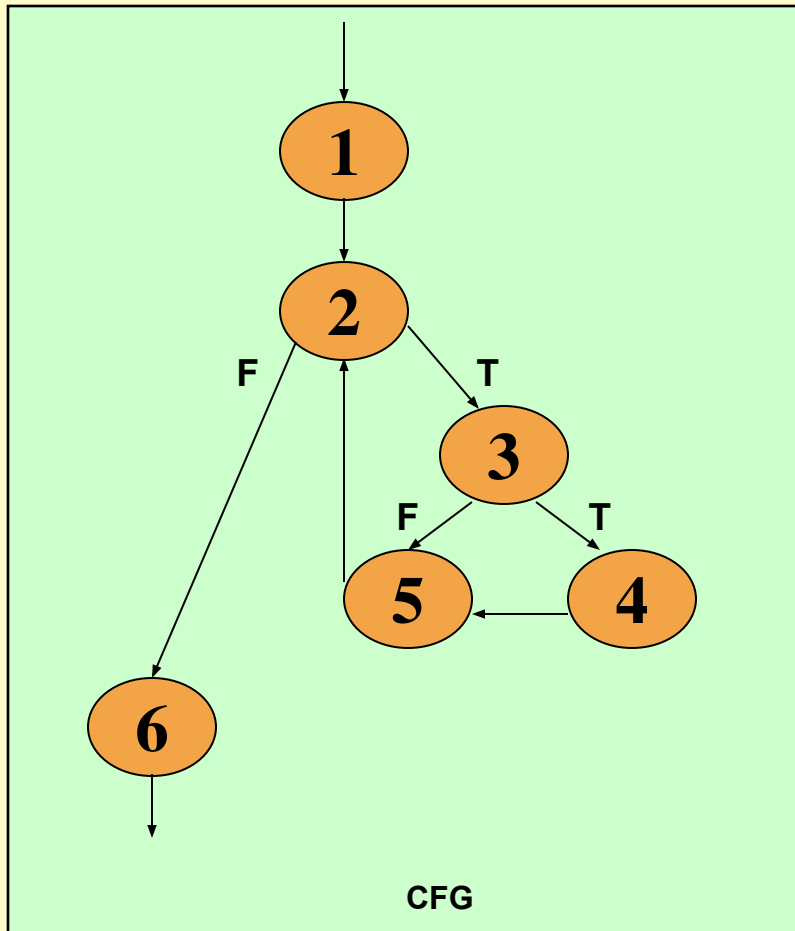
- ▮  $E$  = the number of edges of the graph.
- ▮  $N$  = the number of nodes of the graph.
- ▮  $P$  = the number of connected components.

# Path Base Testing: Step 2



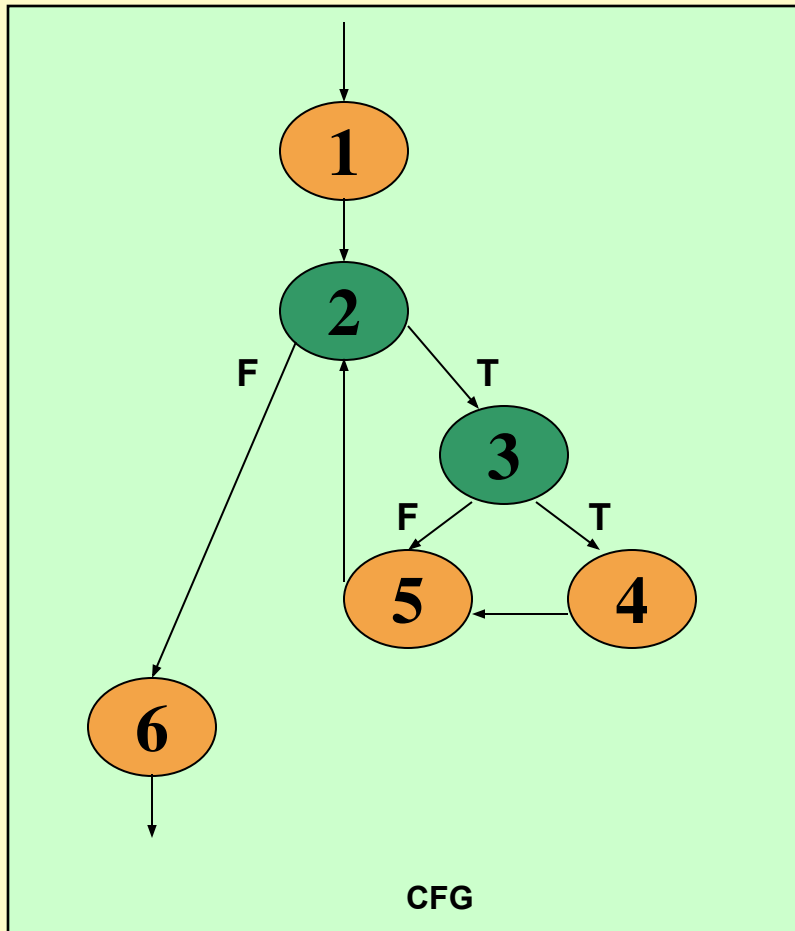
- Cyclomatic complexity =
- The number of 'regions' in the graph( R ) + 1
-

# Path Base Testing: Step 2



- Cyclomatic complexity,  $M =$ 
  - The number of 'regions' in the graph(  $R$  ) + 1
  - $= 2 + 1 = 3$

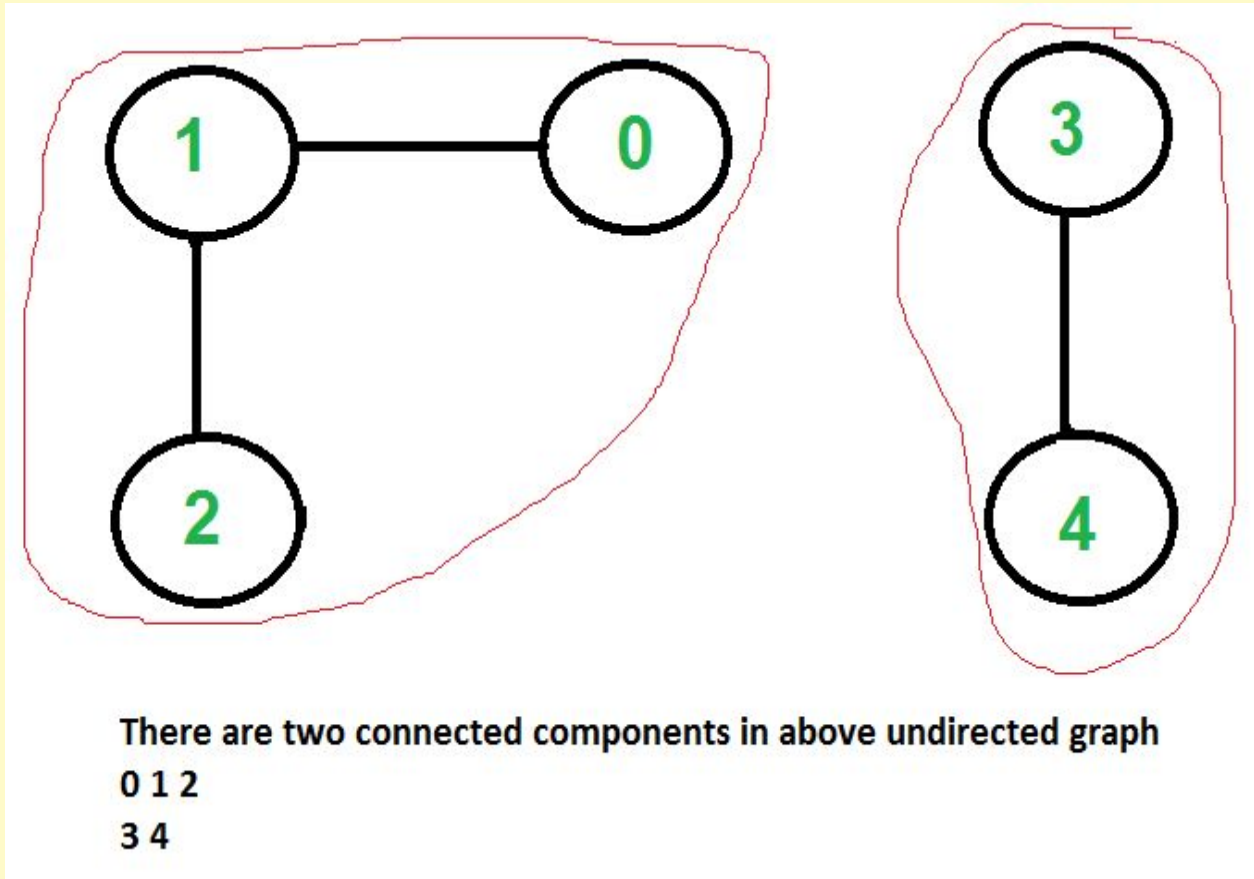
# Path Base Testing: Step 2



- $M = \text{Number of 'predicate' node (P)} + 1$
- In this example:
  - Predicates,  $P = 2$ 
    - (Node 2 and 3)
  - Cyclomatic Complexity,  $M$   
 $= 2 + 1$   
 $= 3$

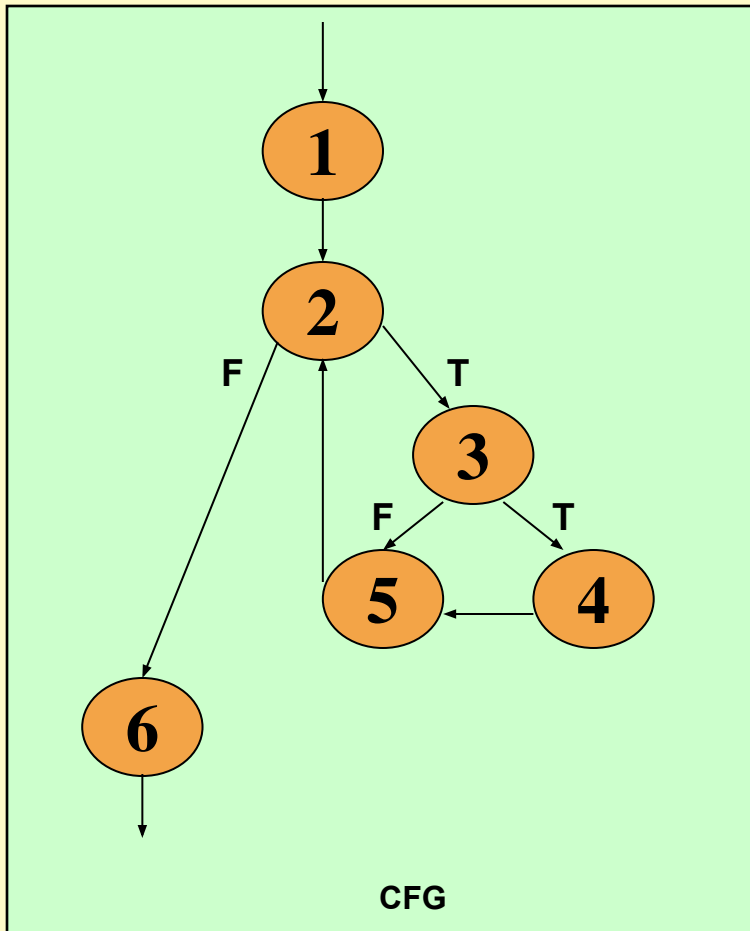
# Path Base Testing: Step 2

## Connected Components in a Graph





# Path Base Testing: Step 2



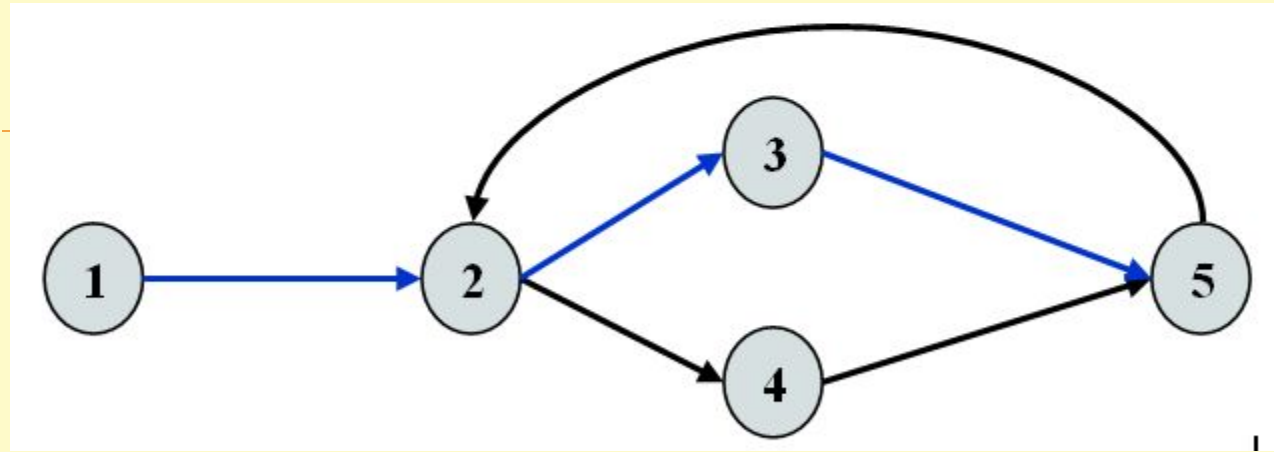
- ❑ Cyclomatic complexity,  $M = E - N + 2P$
- ❑ E, edges = 7
- ❑ N, nodes = 6
- ❑ P, connected components = 1
- ❑  $= 7 - 6 + (2 \times 1)$
- ❑  $= 3$

# Path Base Testing: Step 3

---

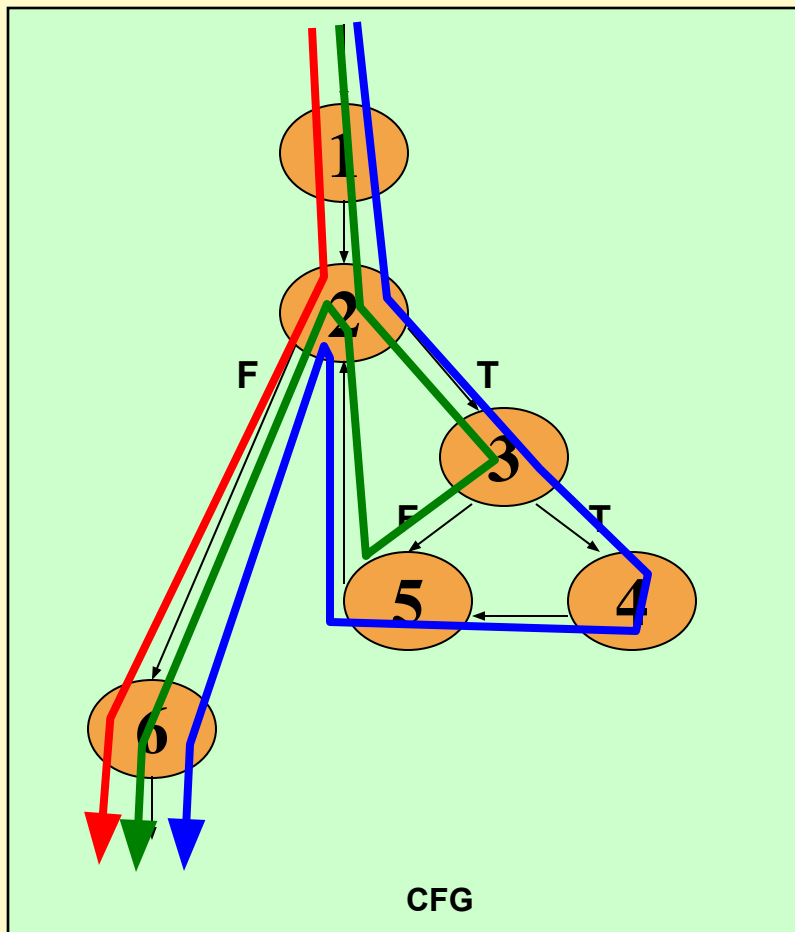
- Independent path:
  - An **executable** or **realizable path** through the graph from the start node to the end node that has not been traversed before.
  - **Must** move along **at least one arc** that has not been yet traversed (an unvisited arc).
  - The objective is to cover all statements in a program by independent paths.
- The number of independent paths to discover  $\leq$  Cyclomatic complexity number, M
- The set of Independent paths is called **Basic Path Set**

# Example



- $M = \text{Regions} + 1 = 2 + 1 = 3$
- 1-2-3-5 can be the first independent path; 1-2-4-5 is another; 1-2-3-5-2-4-5 is one more.
- Alternatively, if we had identified 1-2-3-5-2-4-5 as the first independent path, there would be no more independent paths.
- The number of independent paths therefore can vary according to the order we identify them.

# Path Base Testing: Step 3



- Cyclomatic complexity = 3.
- Need at most **3** independent paths to cover the CFG.
- In this example:
  - [ 1 – 2 – 6 ]
  - [ 1 – 2 – 3 – 5 – 2 – 6 ]
  - [ 1 – 2 – 3 – 4 – 5 – 2 – 6 ]

# Path Base Testing: Step 4

- Prepare a test case for each independent path.

- In this example:

- Path: [ 1 – 2 – 6 ]

- Test Case:  $A = \{ 5, \dots \}, N = 1$

- Expected Output: 5

```
min = A[0];  
I = 1;  
  
while (I < N) {  
    if (A[I] < min)  
        min = A[I];  
    I = I + 1;  
}  
print min
```

1  
2  
3  
4  
5  
6

Try to verify that the test cases actually force execution along a desired path.