CSE 470
Software Engineering
Testing - CFG

Imran Zahid
Lecturer
Computer Science and Engineering, BRAC University

BRAC UNIVERSITY
Inspiring Excellence

Software Testing

Software Testing

- Testing is the process of exercising software with test cases
- The essence of software testing is to determine a set of test cases for the item to be tested
- The target of the test can vary:
  - A single module, a group of such modules (related by purpose, use, behavior, or structure), or an entire system

Software Testing - Stages

- Unit Testing
- Integration Testing
- System Testing

Software Testing - Observations

- It is impossible to completely test any nontrivial module or system
  - Practical Limitations: Complete testing is prohibitive in time and cost
  - Theoretical Limitations: e.g. Halting Problem
- “Testing can only show the presence of bugs, not their absence” - Dijkstra
- Testing is done best by independent testers
  - We often develop an attitude that the program should work in a certain way
  - Programmers stick to the data set that makes the program work
  - A program often doesn’t work when tried by someone else
    - “It works on my machine”

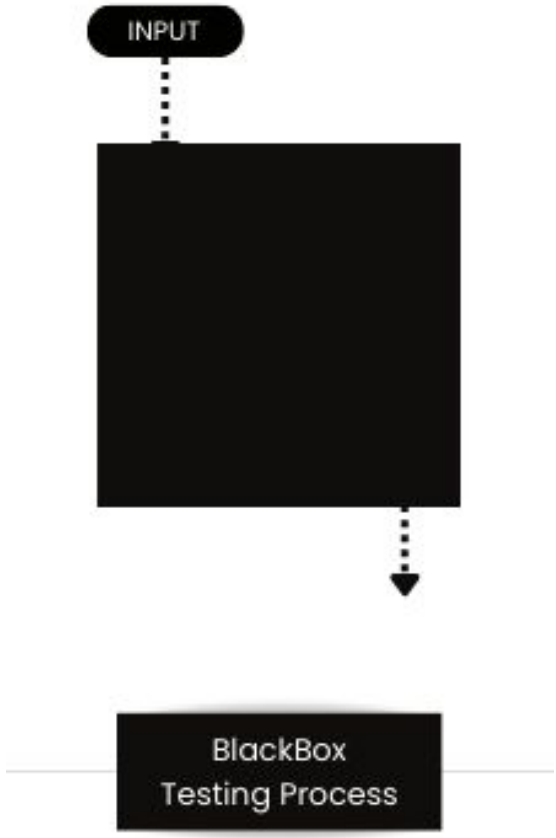
Black Box Testing

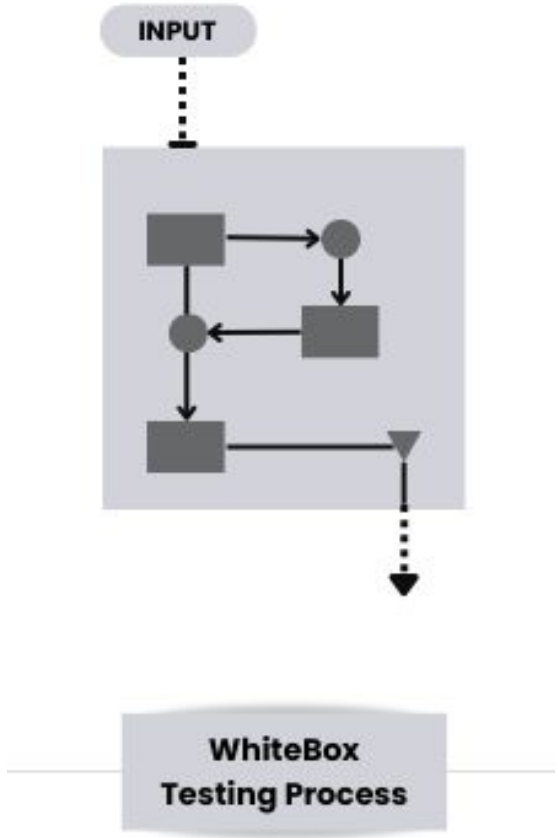
- Black-box testing is a type of testing in which it only focuses on the outer structure of the implementation that needs to be tested.
- In other words, we say that the tester does not know the internal functions of the code.
- Black box testing will be done on the external structure of the system. The input will go in the black box testing and it produces an Output as a Response and it will be tested.

White Box Testing

- White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing.
- It is also called glass box testing, clear box testing, structural testing, transparent testing or open box testing.

Black Box vs White Box Testing





Black Box vs White Box Test Cases

- In black box testing, test cases focus on whether the output matches the expected result based on specific inputs
- In white box testing, the entire internal structure and logic of the software are tested for correctness
- Black box test cases are created based on input-output specifications without knowing the internal code
- White box test cases are designed using knowledge of the code structure to ensure coverage of all paths, conditions and logic

BRAC UNIVERSITY
Inspiring Excellence

Software Quality Metrics

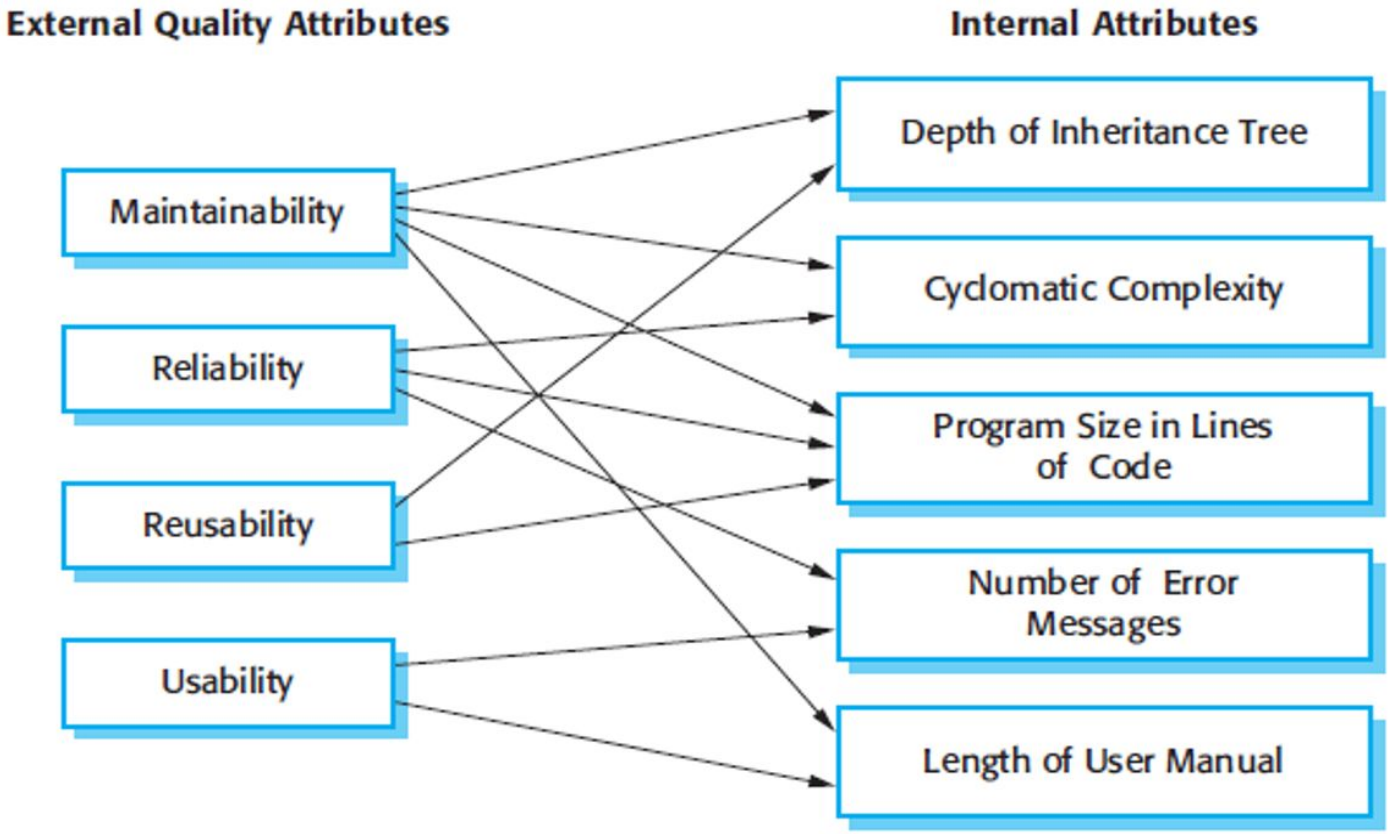
Software Measurement

- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don’t make systematic use of software measurement.
- There are few established standards in this area.

Software Metrics

- Any type of measurement which relates to a software system, process or related documentation.
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allows the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

Relationships between internal and external attributes



BRAC UNIVERSITY
Inspiring Excellence

Control Flow Graph

Control Flow Graph (Path Based Testing)

- 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.

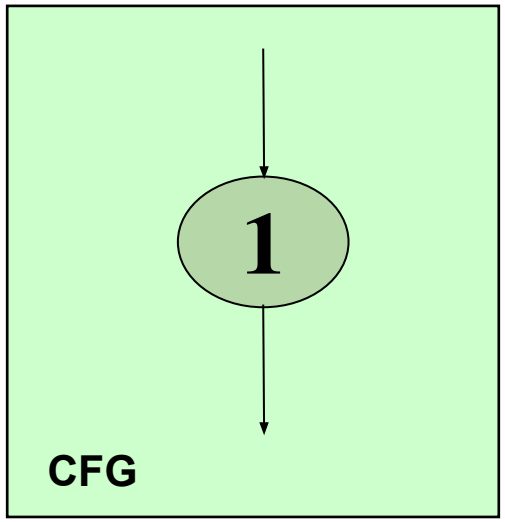
Notation Guide for CFG

- 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

Example - Code Block

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

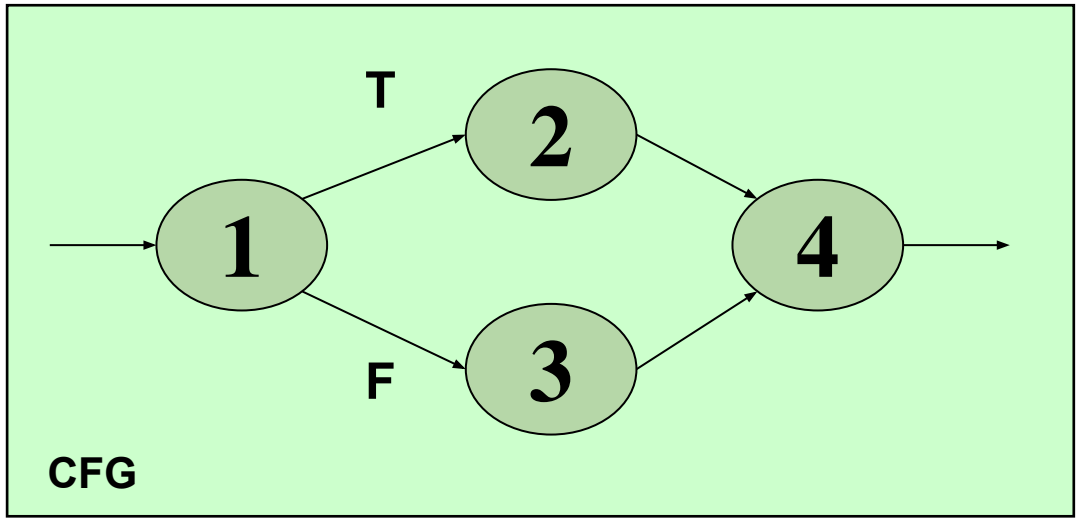
Can be represented as one node as there is no branch.



Example - If-Else

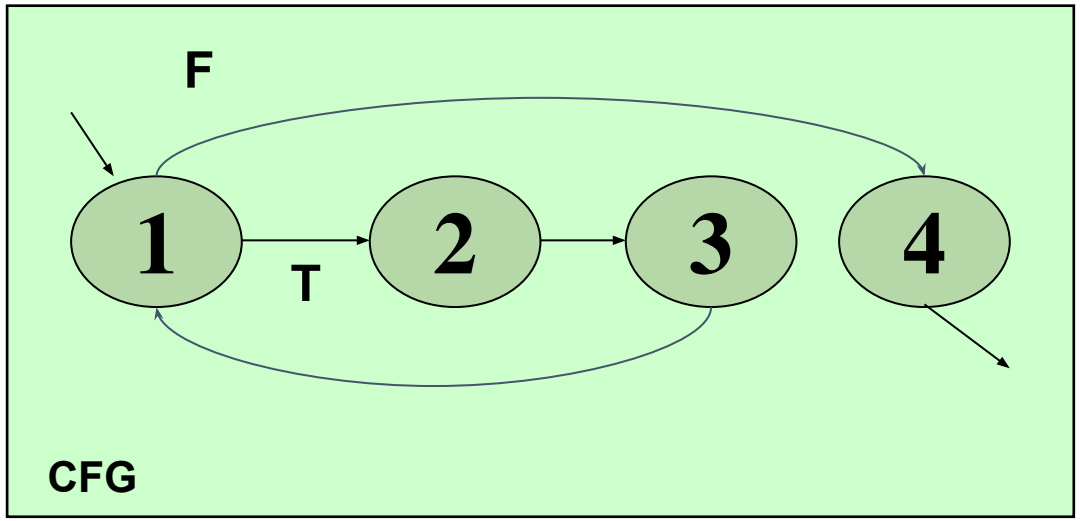
```
if X > 0 then      } 1
  Statement1;      } 2
else
  Statement2;      } 3
```

4



Example - While loop

```
while X < 10 {          1
    Statement1;         2
    X++;
}                       3
4
```



Logical Nodes

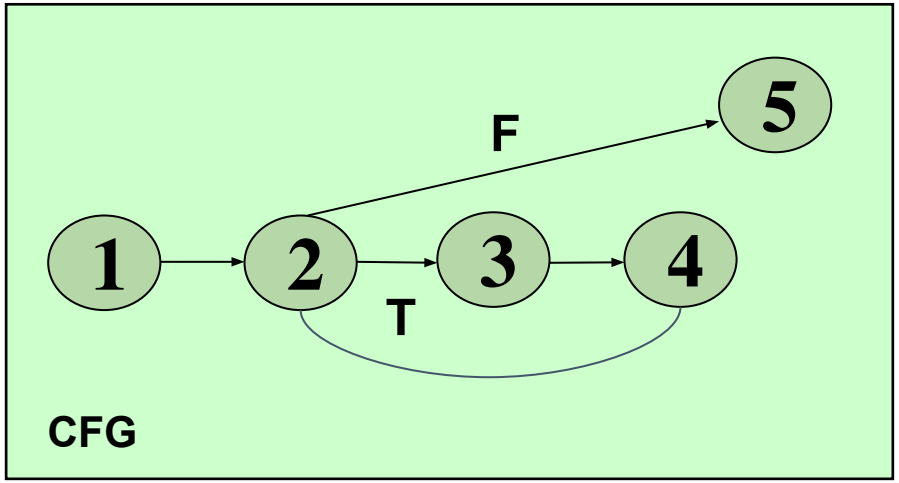
Question: Why is there a node 4 in both CFGs?

Answer: A logical node

- Wherever one or more different branches would merge together but there is no one statement that could represent that point of merging, introduce a logical node

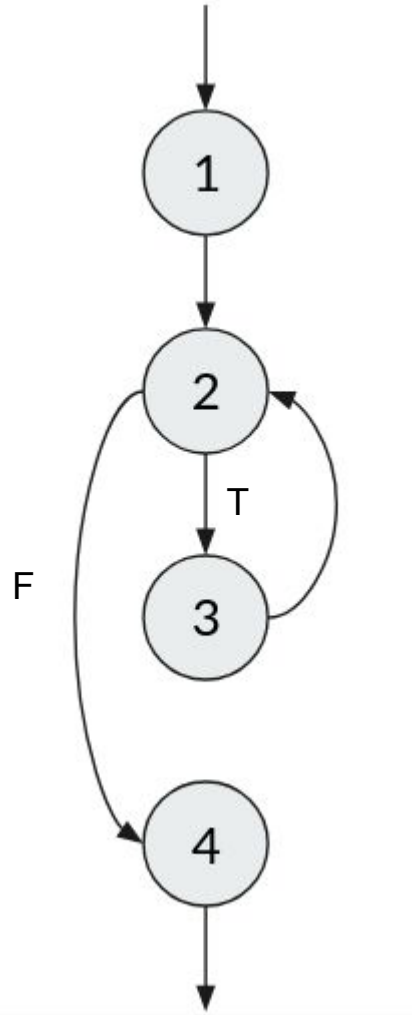
Example - For loop

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



Structure of the Python For Loop

```
      1   2
    for i in range(i):
3           print("hello")
4   print("world")
```



Structure of the Python For Loop

for _ in _:

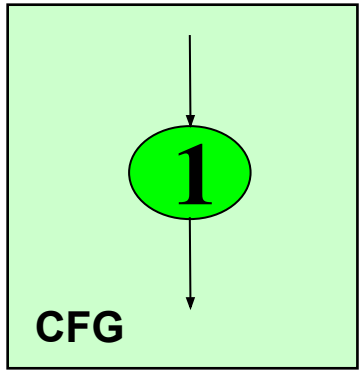
The above statement does two things:

- It initializes the iterator (creates one in some cases, as in the above example it creates an iterator over the range i). This occurs one time when the for statement is first executed.
- It checks if there is a next object in the iterator, i.e. it checks if the end has been reached. If there is a next object (returns True), then it executes the body, otherwise the loop terminates.

Number of Paths through CFG

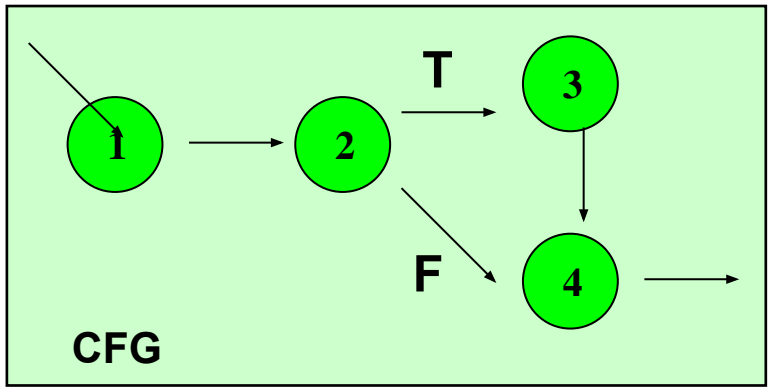
- Given a program, how do we exercise all statements and branches at least once?
- Translating the program into a CFG, an equivalent question is:
  - Given a CFG, how do we cover all arcs and nodes at least once?
- Since a path is a trail of nodes linked by arcs, this is similar to ask:
  - Given a CFG,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

Example



Only **one** path is needed:

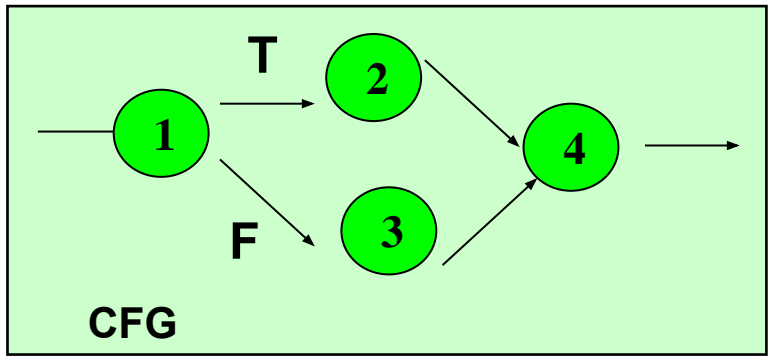
[1]



Two paths are needed:

[1 – 2 – 4]

[1 – 2 – 3 – 4]

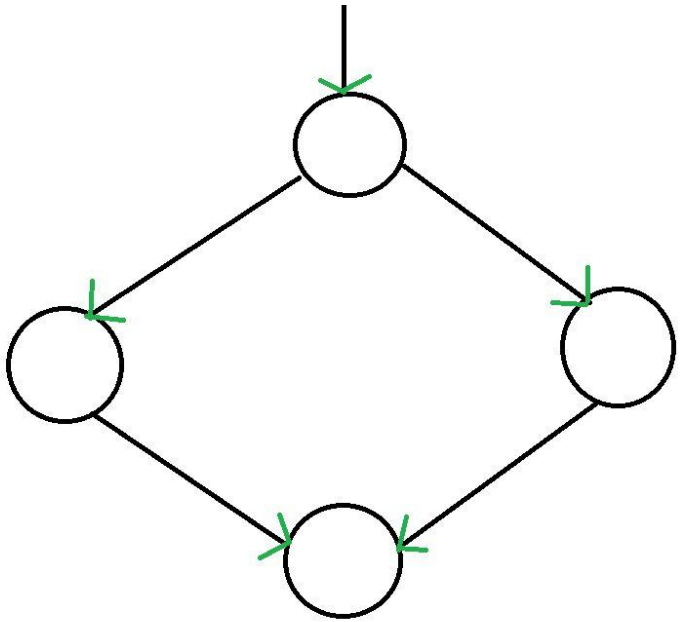


Two paths are needed:

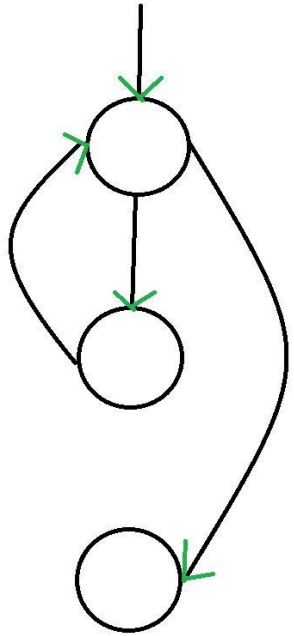
[1 – 2 – 4]

[1 – 3 – 4]

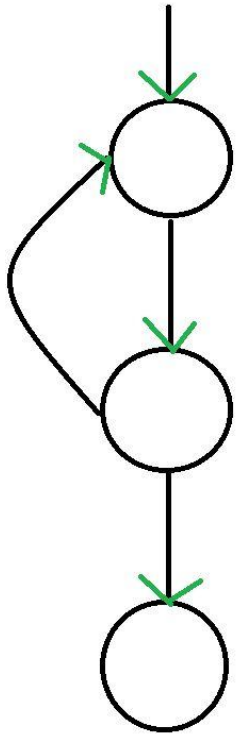
Control Flow Graph Types



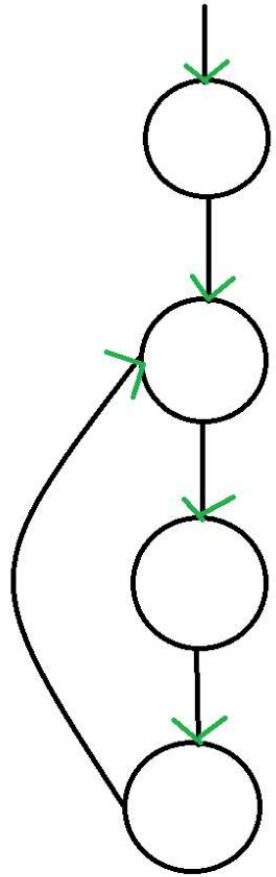
If-then-else



while



do-while



for

BRAC UNIVERSITY
Inspiring Excellence

Cyclomatic Complexity

CFG - Goals

- Provide guidelines for white box testing, the different approaches to exhaustively test all blocks of code
- Provide an objective metric for software quality assurance, in terms of its complexity (the Cyclomatic Complexity) which we aim to minimize

White Box Testing: Path Based

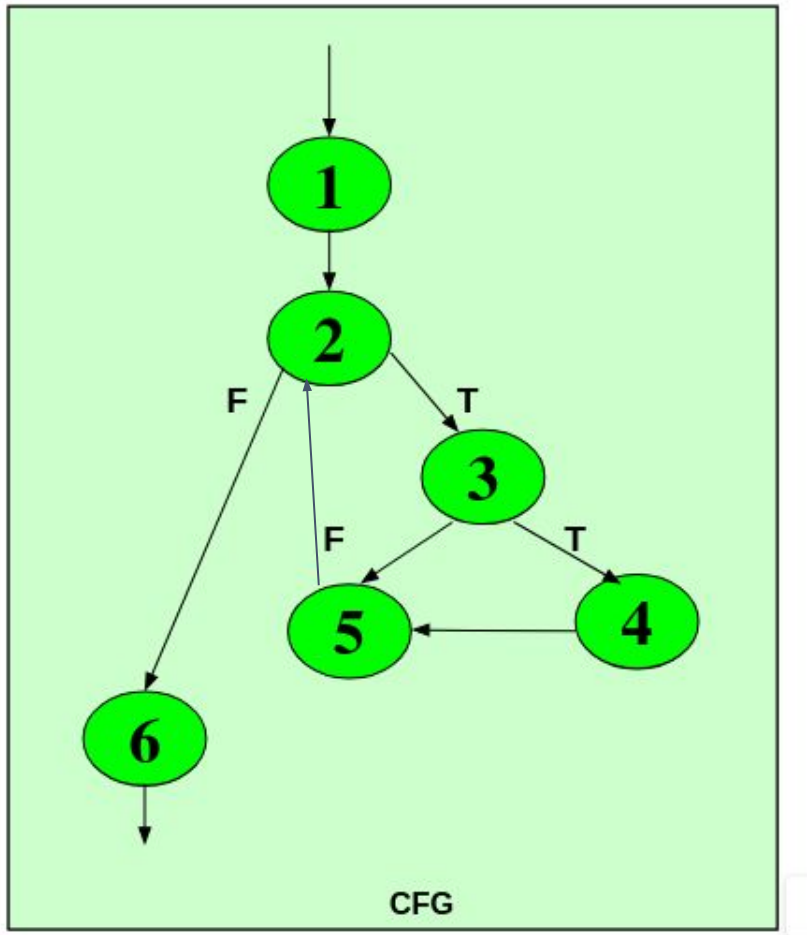
- A generalized technique to find out the number of paths needed (known as cyclomatic complexity) to cover all arcs and nodes in CFG.
- Steps:
  a. Draw the CFG for the code fragment.
  b. Compute the cyclomatic complexity number M, for the CFG.
  c. Find at most M paths that cover the nodes and arcs in a CFG, also known as Basic Paths Set;
  d. 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

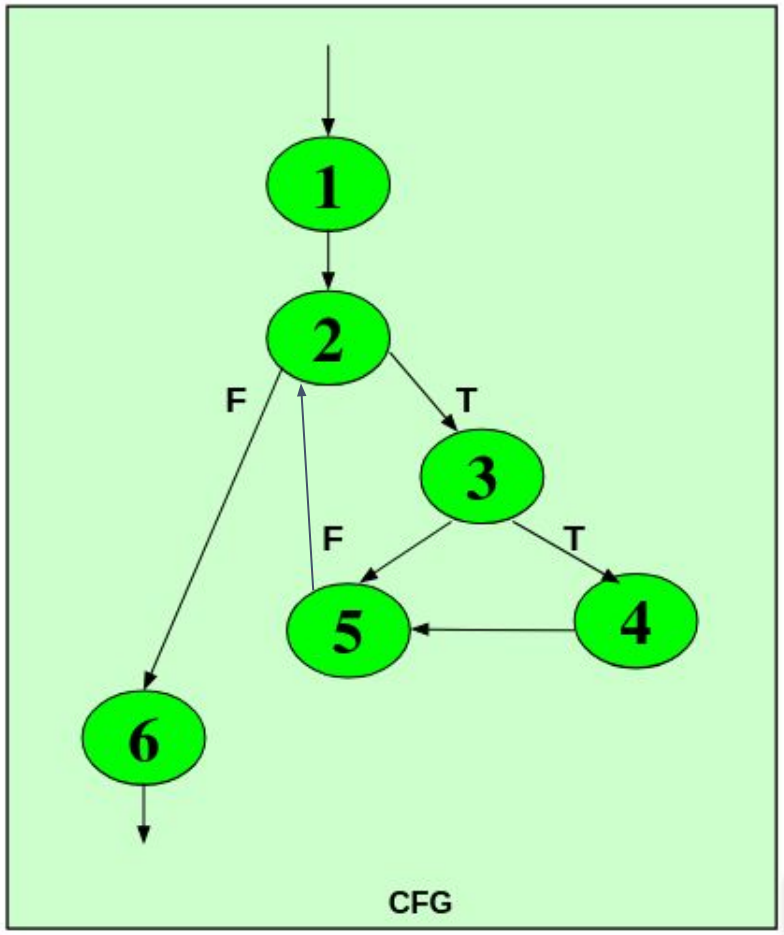


Path Based Testing: Step 2

There are 3 equations for defining the Cyclomatic Complexity, M

1. $\mathbf{M = R + 1}$,
   where R = the number of regions in the graph.
2. $\mathbf{M = P + 1}$,
   where P = the number of predicate nodes in the graph.
3. $\mathbf{M = E - N + 2P}$, where
   E = the number of edges of the graph.
   N = the number of nodes of the graph.
   P = the number of connected components.

Path Based Testing: Step 2



Region

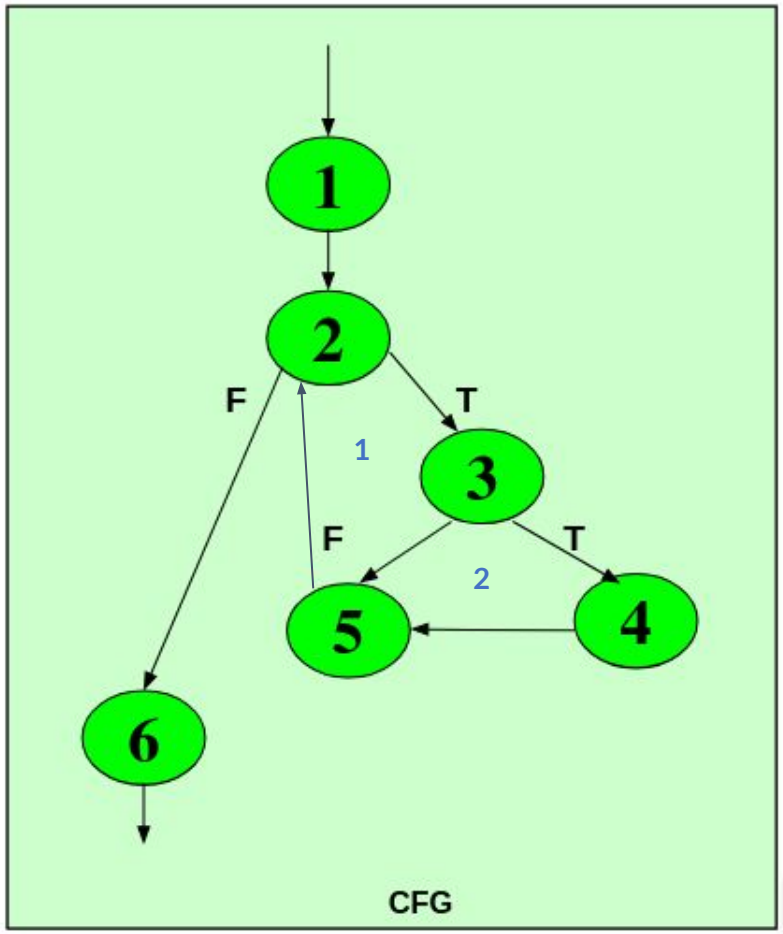
In a Control Flow Graph (CFG), a region is defined as a closed area that is bounded by edges in the graph, such that you cannot traverse between two points in the region without crossing an edge.

- Each region is distinct and enclosed.
- Regions do not merge to form a larger region.

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

$$M = R + 1$$

Path Based Testing: Step 2



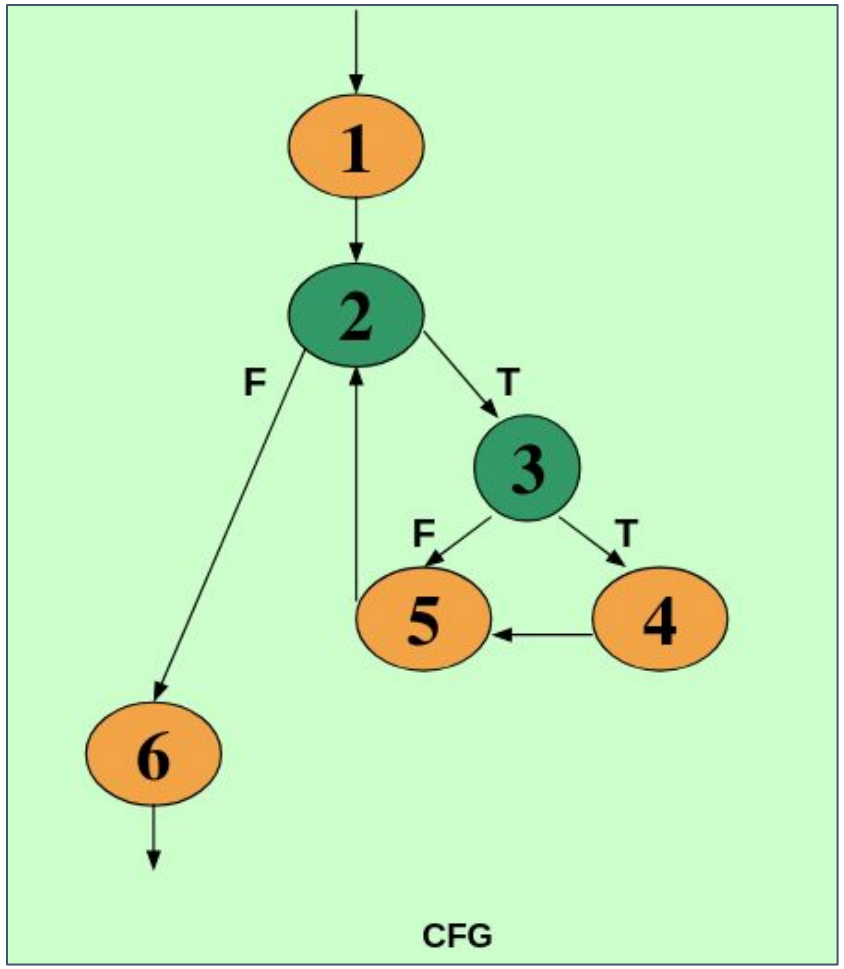
Cyclomatic complexity =

The number of 'regions' in the graph (R) + 1

$= 2 + 1$

$= 3$

Path Based Testing: Step 2



Predicate Node:

A predicate node in a Control Flow Graph (CFG) is a node that represents a decision point in the program, typically where the flow of control can branch based on a condition.

$$M = \text{Number of predicate node } (P) + 1$$

Here, Predicates, $P = 2$ (Node 2 and 3)

Cyclomatic Complexity, M

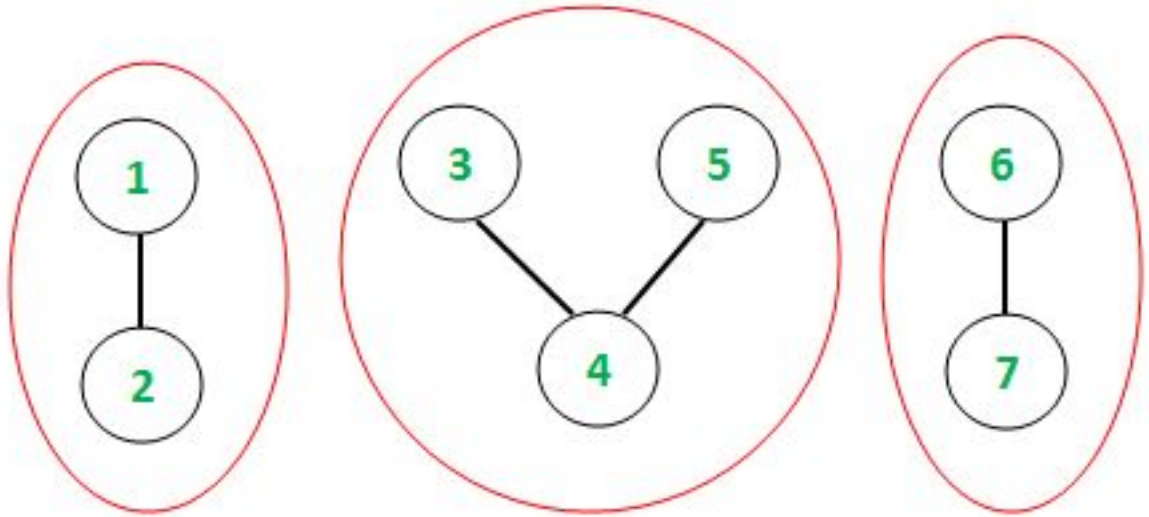
$$= 2 + 1$$

$$= 3$$

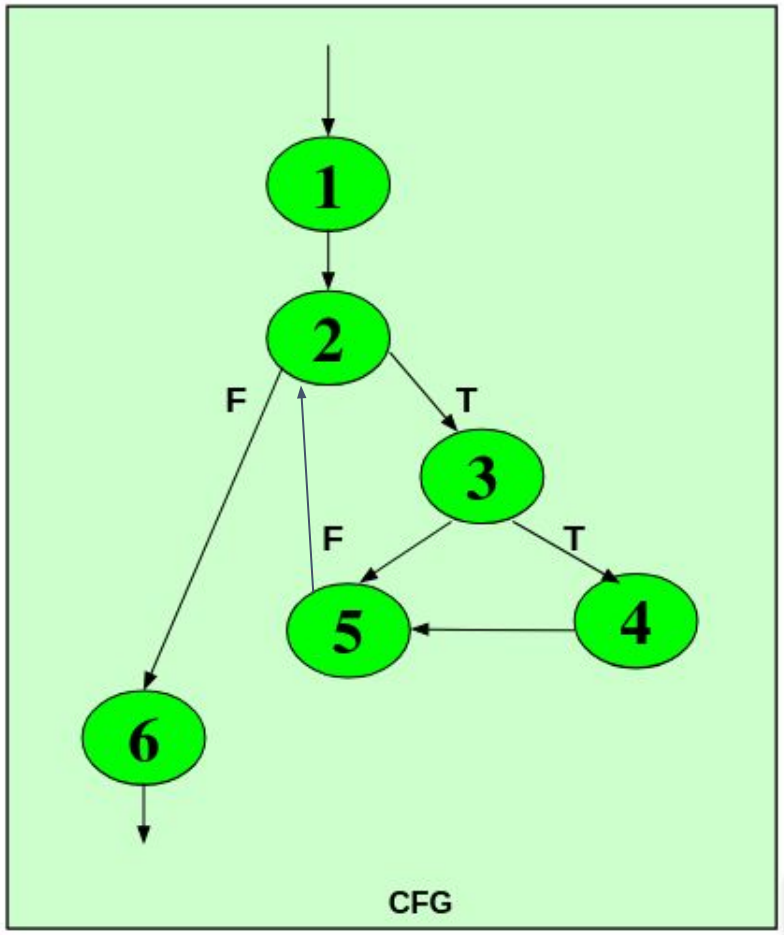
Path Based Testing: Step 2

Connected Components in a Graph

A connected component in a graph is a maximal subgraph in which any two vertices are connected to each other by a path, and there are no connections (edges) between vertices in this subgraph and any vertices outside it.



Path Based Testing: Step 2



Cyclomatic complexity, $M = E - N + 2P$

E, edges (excluding entry and exit) = 7

N, nodes = 6

P, connected components = 1

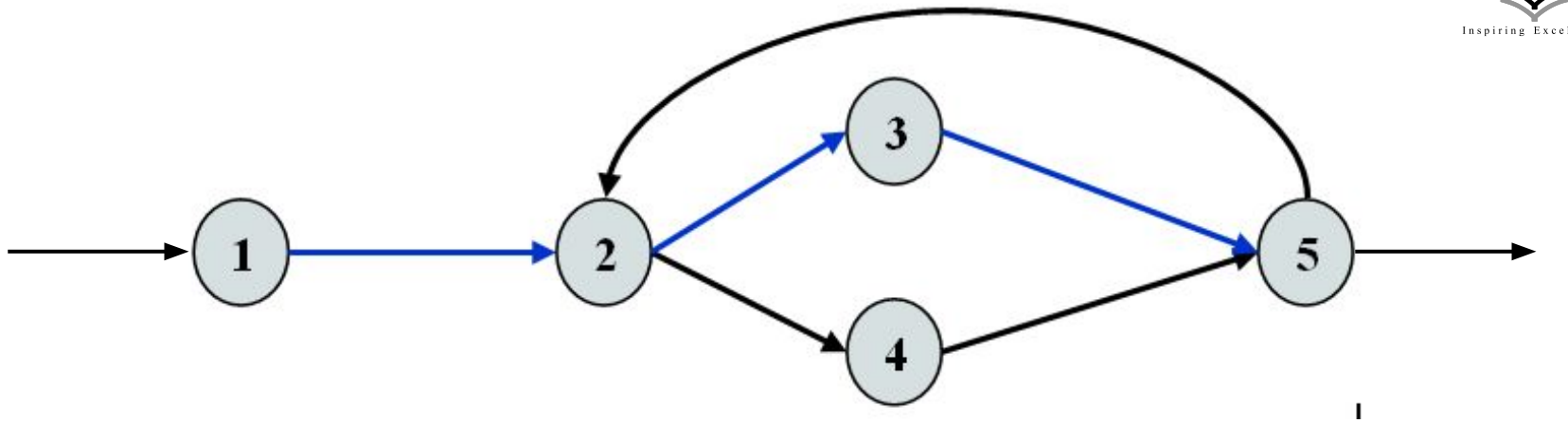
= 7 – 6 + (2x1)

= 3

Path Based 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 <= Cyclomatic complexity number, M
- The set of Independent paths is called Basic Path Set

Example



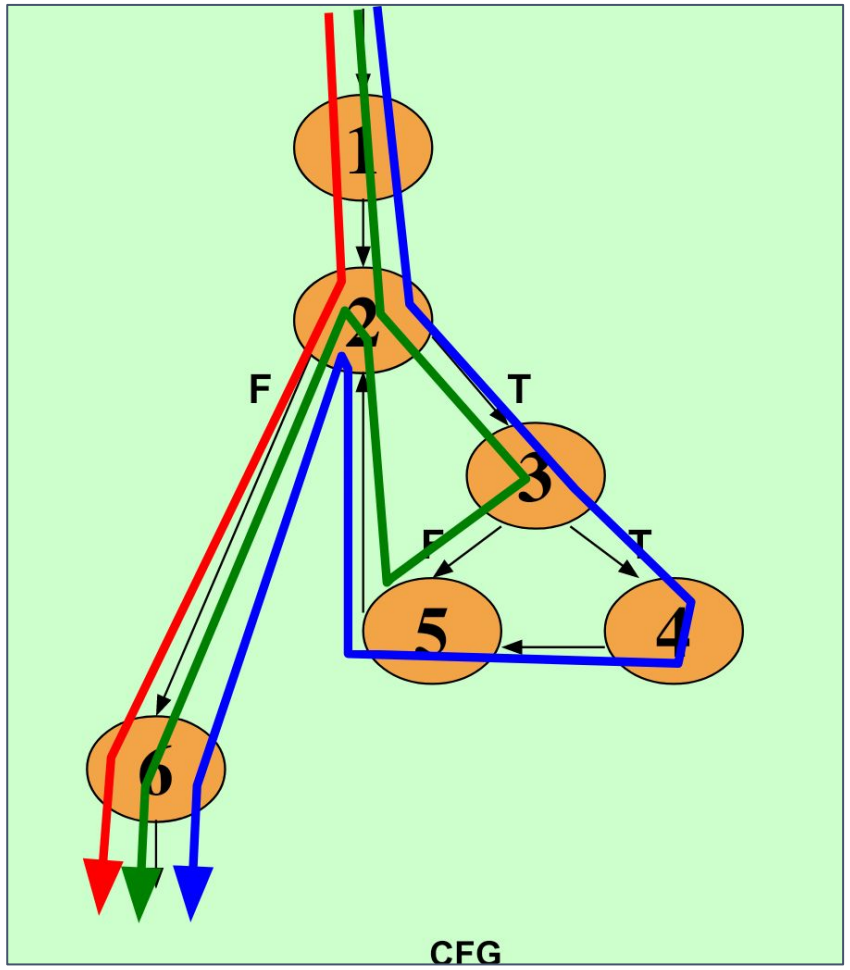
M = Regions + 1 = 2 + 1 = 3

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 Based 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 Based Testing: Step 3

Prepare a test case for each independent path.

In this example:

Path: [1 – 2 – 6]

Test Case: A = { 5, …}, N = 1

Expected Output: 5

```
min = A[0];                    1
I = 1;

while (I < N) {                2
  if (A[I] < min)              3
          min = A[I];          4
   I = I + 1;                  5
}
print min                      6
```

Python Example

if - elif- else

```
1
if condition:
    print(“hello”)
    print(“world”)
elif condition:
    print(“hello”)
else:
    print(“world”)
```

2

3

4

5

for

```
1  2
for i in range(i):
    print(“hello”)
print(“world”)
```

3

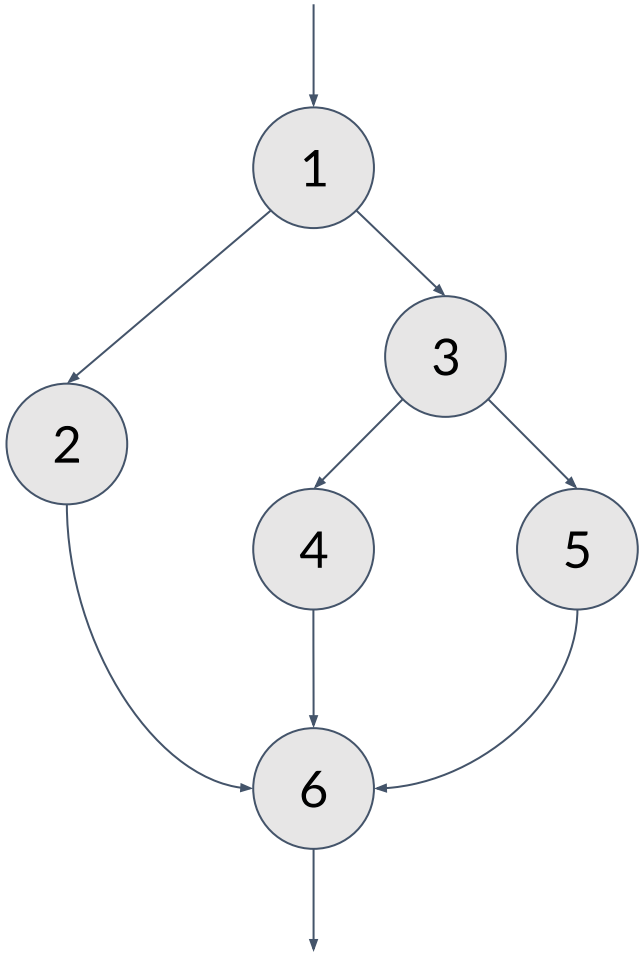
4

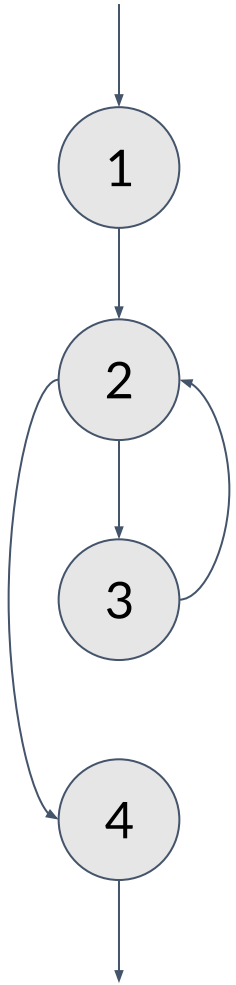
while

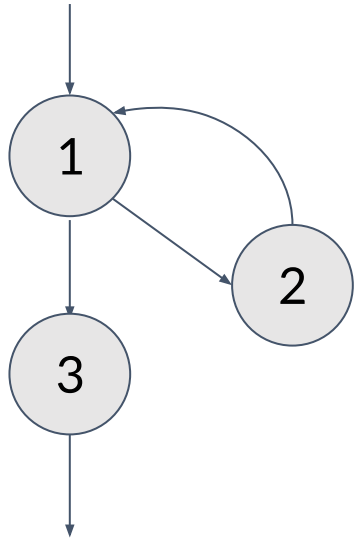
```
    1
while condition:
    print(“hello”)
```

2

Python Example







Thank you