**Lab 7**

**Graph Testing Techniques**

**Control Flow Graph (CFG)**

A **Control Flow Graph (CFG)** is the graphical representation of control flow or [computation during the execution of programs](https://www.geeksforgeeks.org/cyclomatic-complexity/) or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside of a program unit

**Characteristics of Control Flow Graph:**

* Control flow graph is process oriented.
* Control flow graph shows all the paths that can be traversed during a program execution.
* Control flow graph is a directed graph.
* Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

There exist 2 designated blocks in Control Flow Graph:

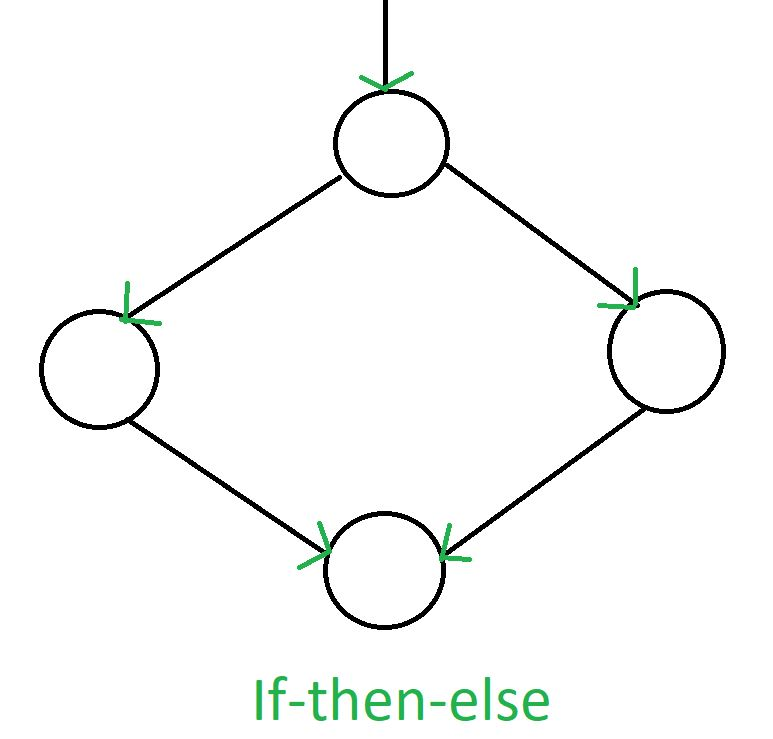
1. **Entry Block:** Entry block allows the control to enter into the control flow graph.
2. **Exit Block:** Control flow leaves through the exit block.

Hence, the control flow graph is composed of all the building blocks involved in a flow diagram such as the start node, end node and flows between the nodes.

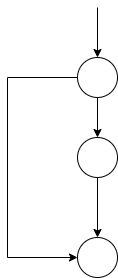
**Control Flow Graph (CFG) Generation:**

Control Flow Graph is represented differently for all statements and loops. Following images describe it:

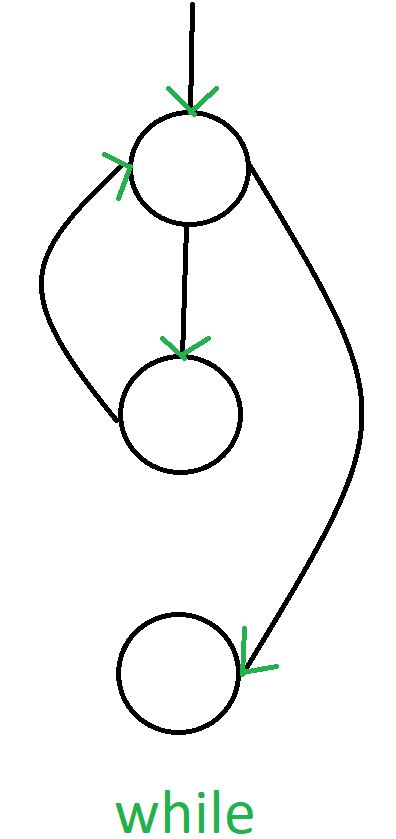
**1. If-else:**



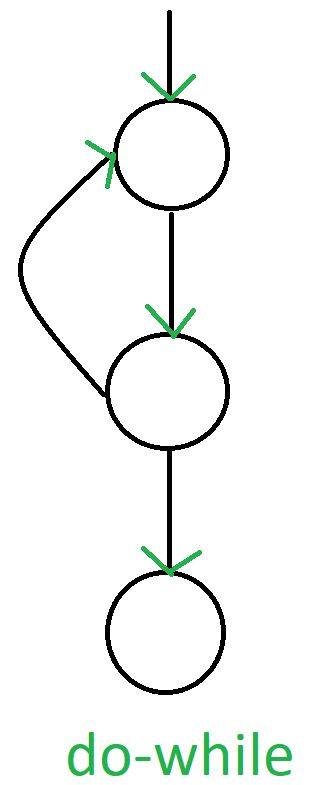
**2. If without else:**



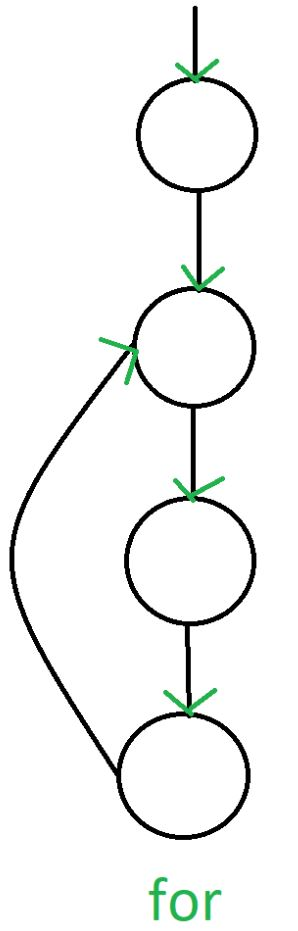
**2. While:**

****

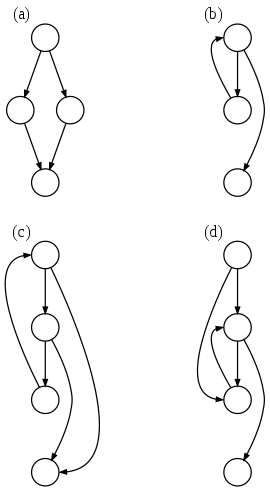
**3. Do-while:**

****

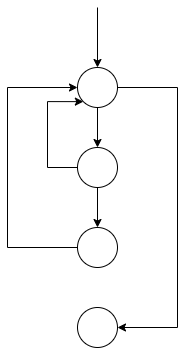
**4. For loop:**

****

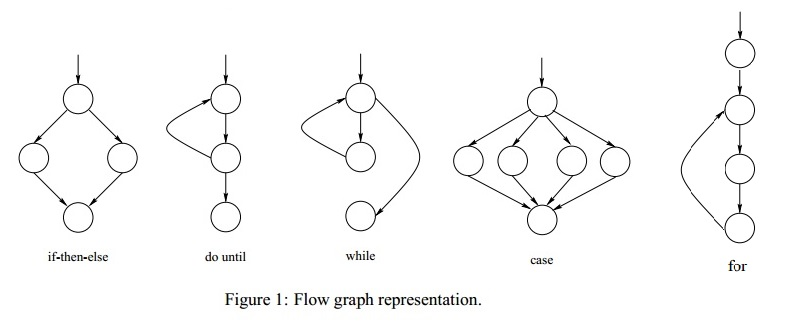
**5. Loop with break:**

****

**6. Loop with continue:**

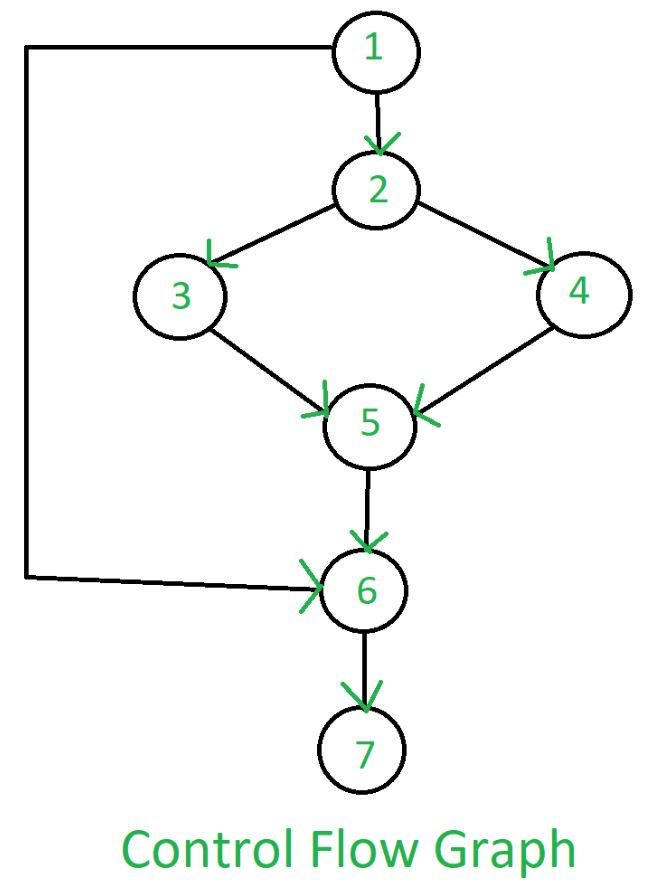
****

**6. Switch statement:**

****

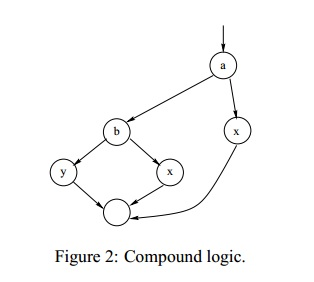
**Example 1:**

|  |
| --- |
| **if A = 10 then**  **if B > C**  **A = B**  **else**  **A = C**  **endif**  **endif**  **print A, B, C** |

****

**Example 2:**

|  |
| --- |
| **1. if a or b**  **2. then execute function x**  **3. else execute function y**  **4. endif** |

****

**Advantage of CFG:**

There are many advantages of a control flow graph. It can easily encapsulate the information per each basic block. It can easily locate inaccessible codes of a program and syntactic structures such as loops are easy to find in a control flow graph.

**Graph Coverage Using CFG**

**We use graphs in testing as follows: -**

- Develop a model of the software as a graph

- Require tests to visit or tour specific sets of nodes, edges, or subpaths

**Test requirements (TR):** Describe properties of test paths

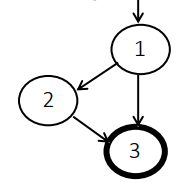
**Test Criterion:** Rules that define test requirements

**Satisfaction:** Given a set TR of test requirements for a criterion C, a set of tests T satisfies C on a graph if and only if for every test requirement in TR, there is a test path in path(T) that meets the test requirement tr.

**Node Coverage**

The first and simplest criteria. It requires that each node in a graph to be executed

* TR contains each reachable node in Graph.



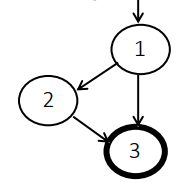
TR = {1 ,2 3}

Test Paths = [ 1, 2, 3 ]

**Edge (Branch) Coverage**

It requires that each edge in a graph to be executed

* TR contains each reachable path of length up to 1, inclusive, in Graph.
* Length up to 1 covers a graph with single node



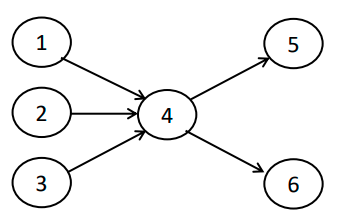
TR = { (1, 2), (1, 3), (2, 3) }

Test Paths = [ 1, 2, 3 ] [ 1, 3 ]

**Edge-Pair Coverage**

Edge-pair coverage requires pairs of edges, or subpaths of length 2

* TR contains each reachable path of length up to 2, inclusive, in Graph
* Length up to 2 is used to include graphs that have less than 2 edges



TR = { [1,4,5], [1,4,6], [2,4,5], [2,4,6], [3,4,5], [3,4,6] }

Test Paths = { [1,4,5], [1,4,6], [2,4,5], [2,4,6], [3,4,5], [3,4,6]}

**Prime Path Coverage**

**Simple path:** A path is simple if no node appears more than once in the path, except that the first and last nodes may be the same.

Some properties of simple paths:

• no internal loops;

• can bound their length;

• can create any path by composing simple paths

• many simple paths exist [too many!]

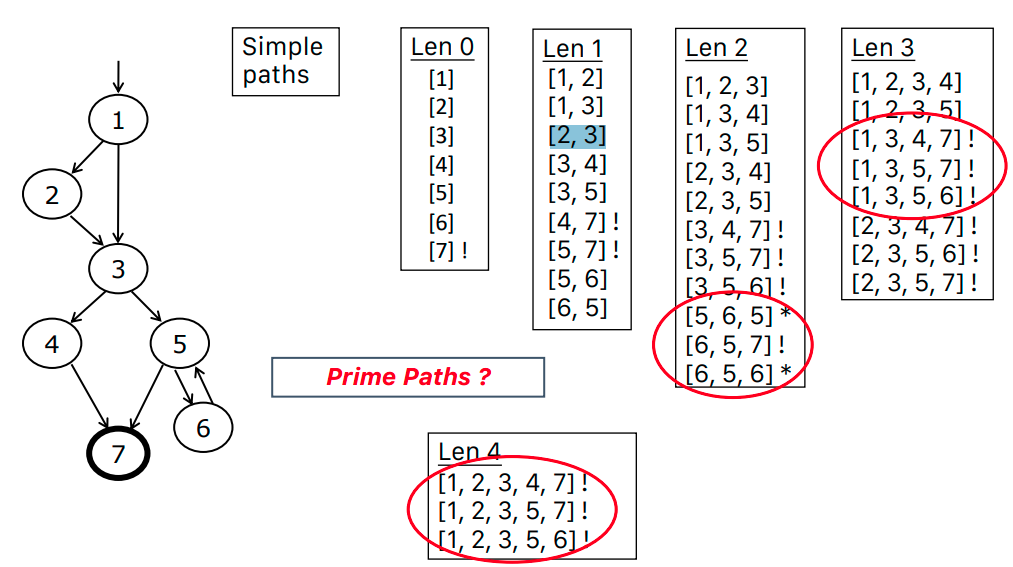
**Prime Path:** A simple path that does not appear as a proper subpath of any other simple path

* The symbol \* denotes a loop
* The symbol ! denotes reaching a final node

To find prime paths  
- generate simple paths level by level

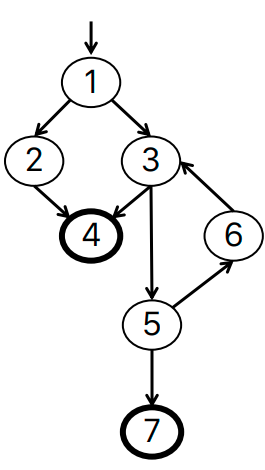
- remove simple paths that appear as a sub-paths in higher level paths

- remaining paths are the prime paths that defines TRs



**Finding paths that cover certain CFG test requirements can be generated using different tools such as** [**https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage**](https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage)

**Lab Task**

****

1. Write the set of test requirements for edge coverage.

2. List test paths that satisfy edge coverage.

3. Write the set of test requirements for edge-pair coverage.

4. List test paths that satisfy edge-pair coverage.

5. Write the set of test requirements for prime path coverage.

6. List test paths that satisfy prime path coverage.