White Box Testing

Ashima Tyagi
Assistant Professor
School of Computer Science & Engineering

Outline

- White box testing
- Control structure testing
- Path testing

White box testing

- It is also called glass box testing or clear box testing or structural testing.
- The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.
- The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.
- 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.

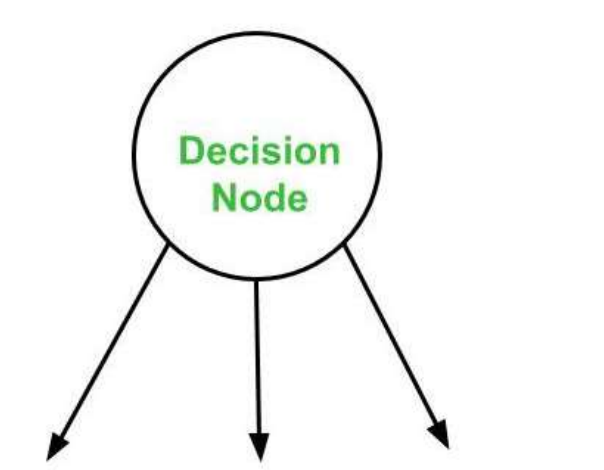
Control Structure Testing

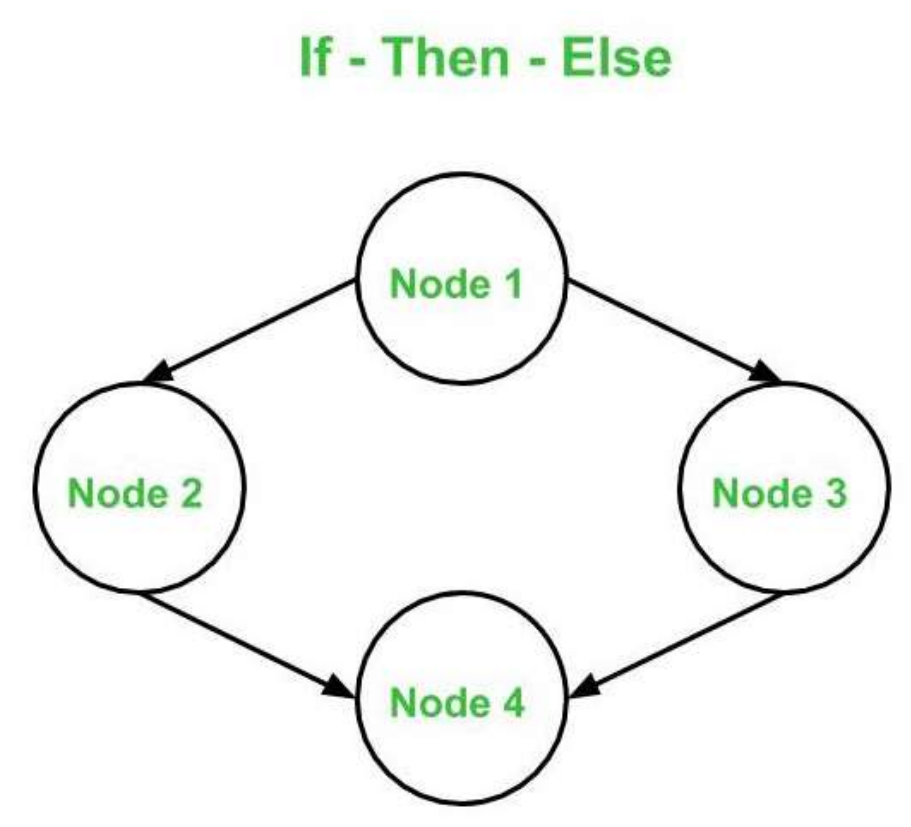
- Control structure testing is used *to increase the coverage area* by testing various control structures present in the program. The different types of testing performed under control structure testing are as follows-

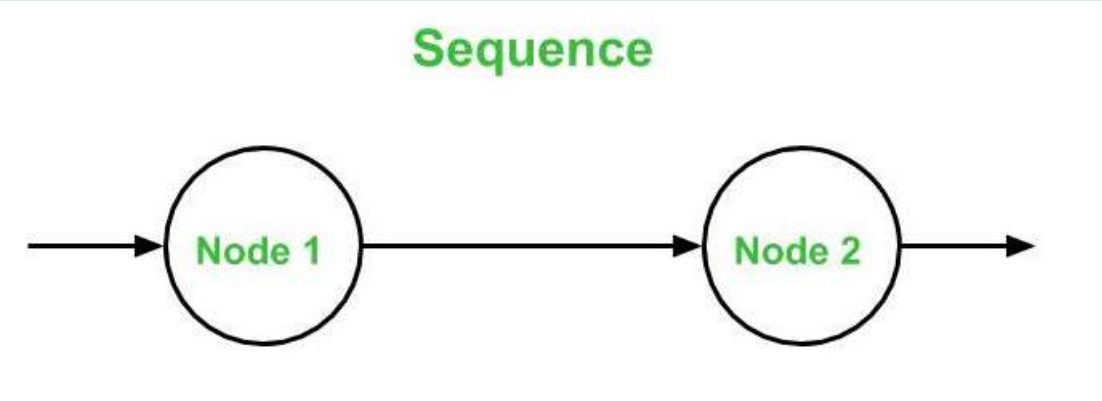
1. Condition Testing
2. Data Flow Testing
3. Loop Testing

1. Condition Testing : Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors. The common types of logical conditions that are tested using condition testing are-

- A relation expression, like E1 op E2 where 'E1' and 'E2' are arithmetic expressions and 'OP' is an operator.
- A simple condition like any relational expression preceded by a NOT (~) operator. For example, (~E1) where 'E1' is an arithmetic expression and 'a' denotes NOT operator.
- A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis. For example, (E1 & E2) | (E2 & E3) where E1, E2, E3 denote arithmetic expression and '&' and ' | ' denote AND or OR operators.
- A Boolean expression consists of operands and a Boolean operator like 'AND', OR, NOT. For example, 'A | B' is a Boolean expression where 'A' and 'B' denote operands and | denotes OR operator.

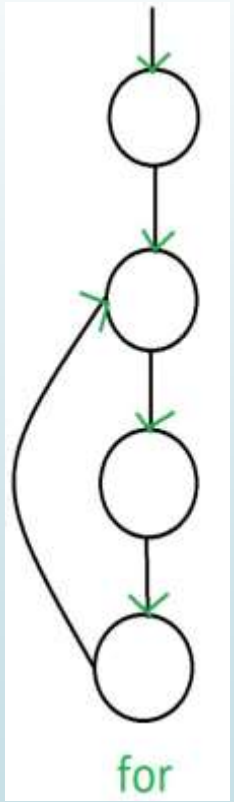

Decision
Node

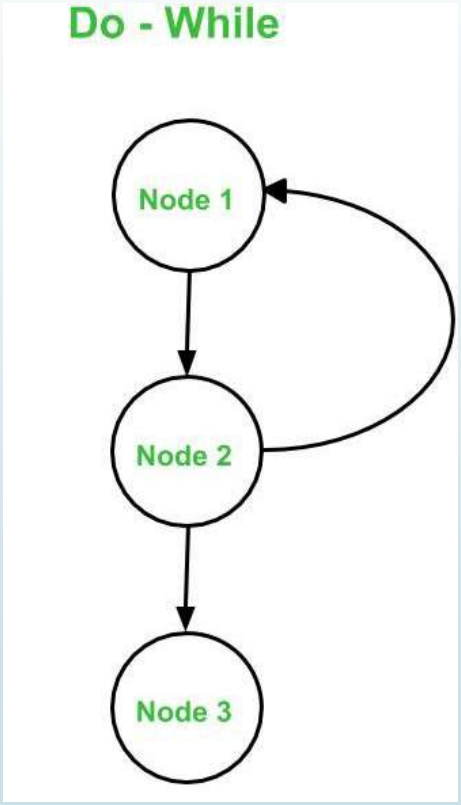

If - Then - Else
Node 1
Node 2
Node 3
Node 4

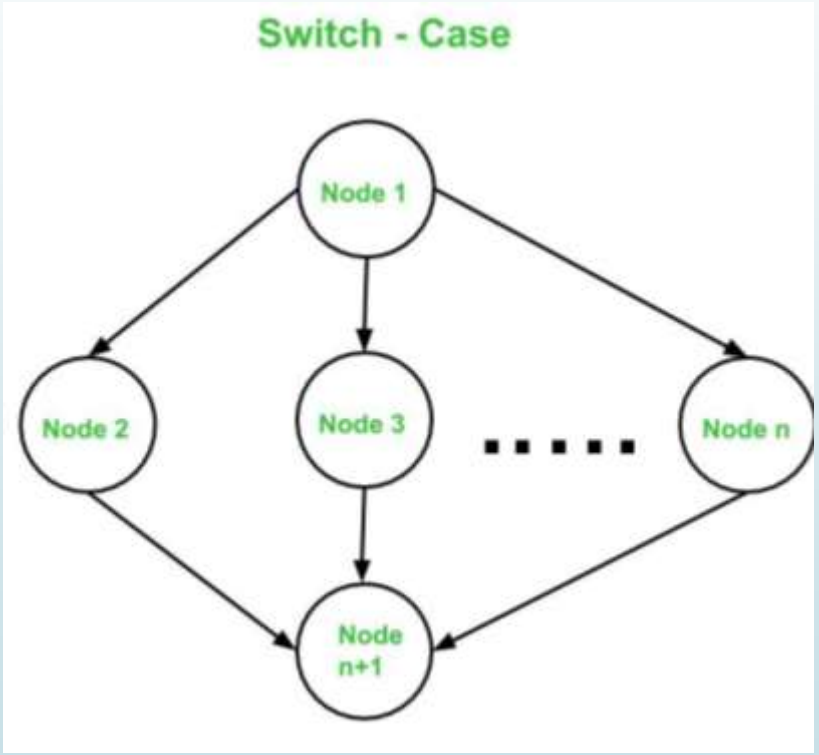

Sequence
Node 1
Node 2

2. Data Flow Testing : It focuses on tracking the flow of data through a program to identify anomalies, such as uninitialized variables, unused variables, and improper data modifications.

3. Loop Testing : Loop testing is actually a white box testing technique. It specifically focuses on the validity of loop construction.

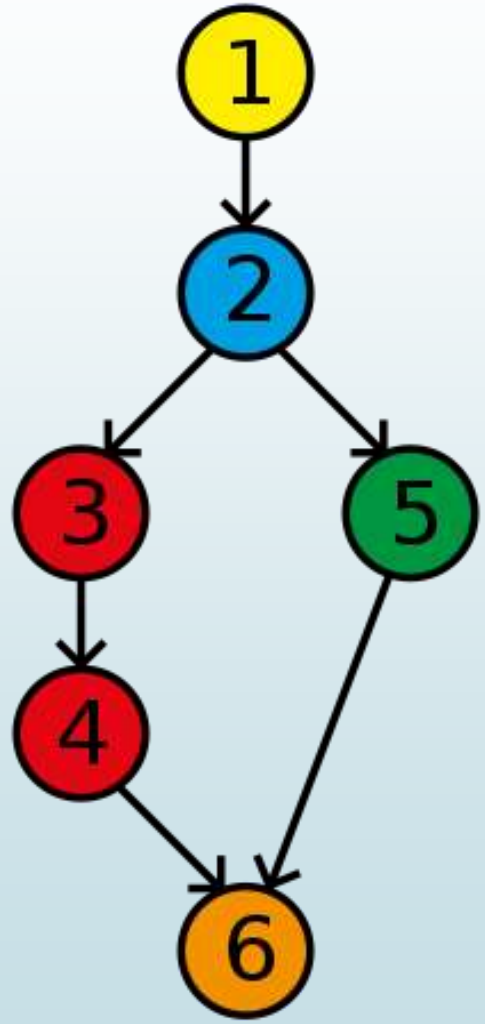






Decision-to-decision path

- A decision-to-decision path, or **DD-path**, is a path of execution (usually through a flow graph representing a program, such as a flow chart) between two decisions.
- DD-Paths are also known as segments
- So, in this flow graph, a DD path can be 1-2-5-6, or 3-4



A DD-path is a set of nodes in a program graph such that one of the following holds:

- It consists of a single node with in-degree = 0 (initial node)
- It consists of a single node with out-degree = 0 (terminal node)
- It consists of a single node with in-degree ≥ 2 or out-degree ≥ 2 (decision/merge points)
- It consists of a single node with in-degree = 1 and out-degree = 1
- It is a maximal chain of length ≥ 1.

Basis Path testing

- Path Testing aims to uncover errors, defects, and unexpected outcomes that might occur when the application takes different paths.
- By systematically exploring all possible routes, testers can identify vulnerabilities, logic errors, and discrepancies that could undermine the reliability of the software.
- Basis Path Testing is a white-box testing technique based on the control structure of a program or a module.
- Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing.

To design test cases using this technique, four steps are followed :

- Construct the Control Flow Graph
- Compute the Cyclomatic Complexity of the Graph
- Identify the Independent Paths
- Design Test cases from Independent Paths

Cyclomatic Complexity – The cyclomatic complexity V(G) is said to be a measure of the logical complexity of a program.

It can be calculated using three different formulae :

1. Formula based on edges and nodes
2. Formula based on Decision Nodes
3. Formula based on number of regions

1. Formula based on edges and nodes :

$$V(G) = e - n + 2*P$$

Where, e is number of edges, n is number of vertices,

P is number of connected components (isolated sections of graph).

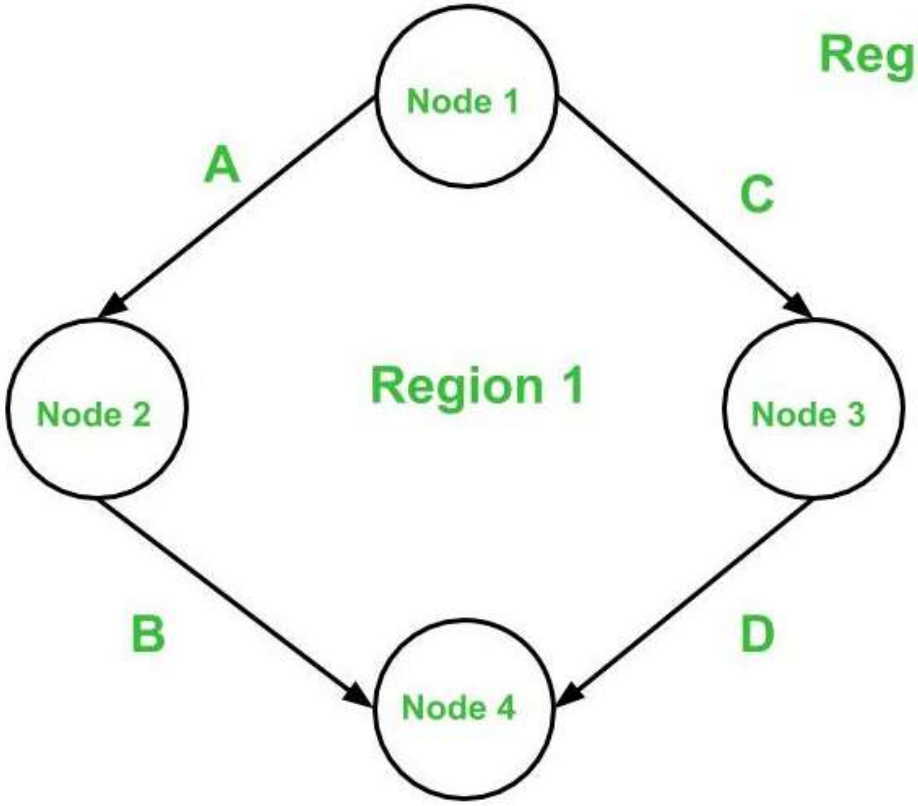
For example, consider this graph,

where, e = 4, n = 4 and p = 1

So,

Cyclomatic complexity V(G)

= 4 - 4 + 2 * 1 = 2



2. Formula based on Decision Nodes :

$$V(G) = D + 1$$

where, d is number of decision nodes

For example, consider this graph,

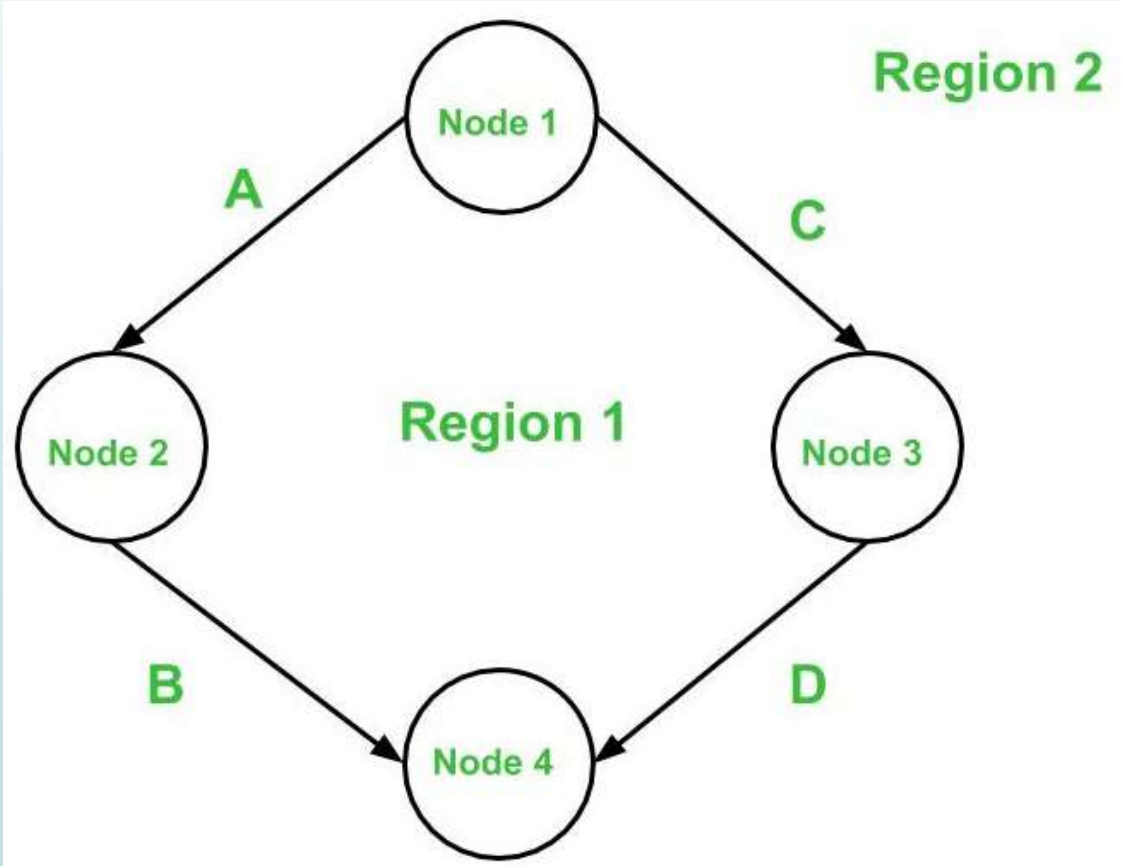
where, D = 1

So,

Cyclomatic Complexity V(G)

= 1 + 1

= 2



2. Formula based on Regions :

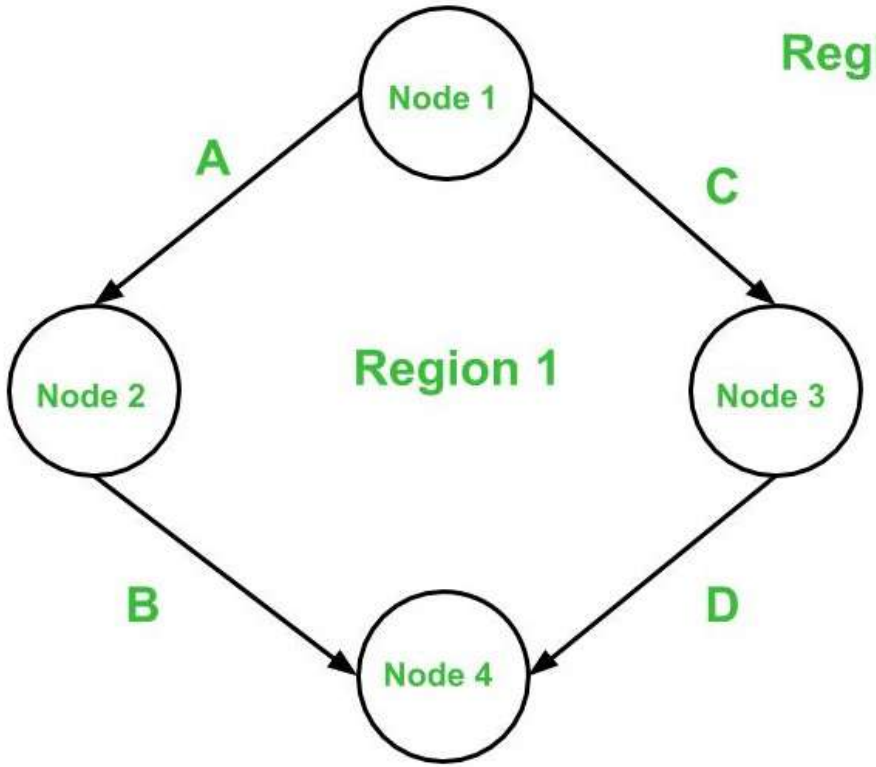
V(G) = number of regions in the graph

For example, consider first graph given above,

Cyclomatic complexity V(G)

= 1 (for Region 1) + 1 (for Region 2)

= 2



Identify Independent paths:

- An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined.
- **The cyclomatic complexity gives the number of independent paths present in a flow graph.**
- Consider first graph given above here the independent paths would be 2 because number of independent paths is equal to the cyclomatic complexity. So, the independent paths in above first given graph :
- Path 1:

A -> B

- Path 2:

C -> D

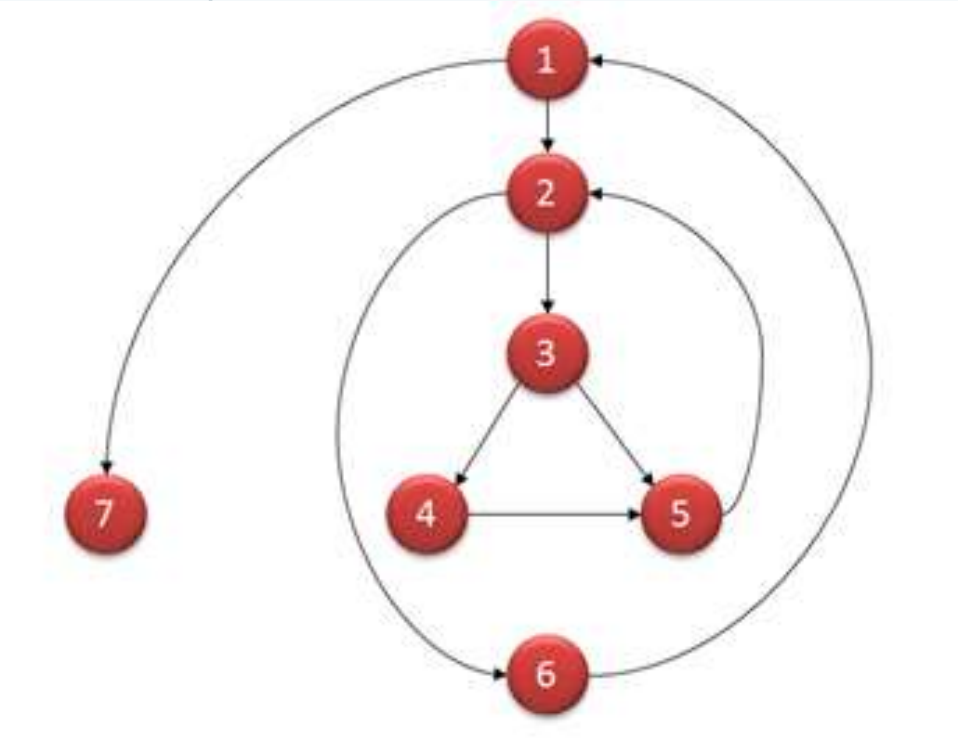
Properties of Cyclomatic complexity

Following are the properties of Cyclomatic complexity:

- V (G) is the maximum number of independent paths in the graph
- V (G) >=1
- G will have one path if V (G) = 1

Numericals

Numerical 1



Cyclomatic Complexity:

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
- V(G) = regions = 4

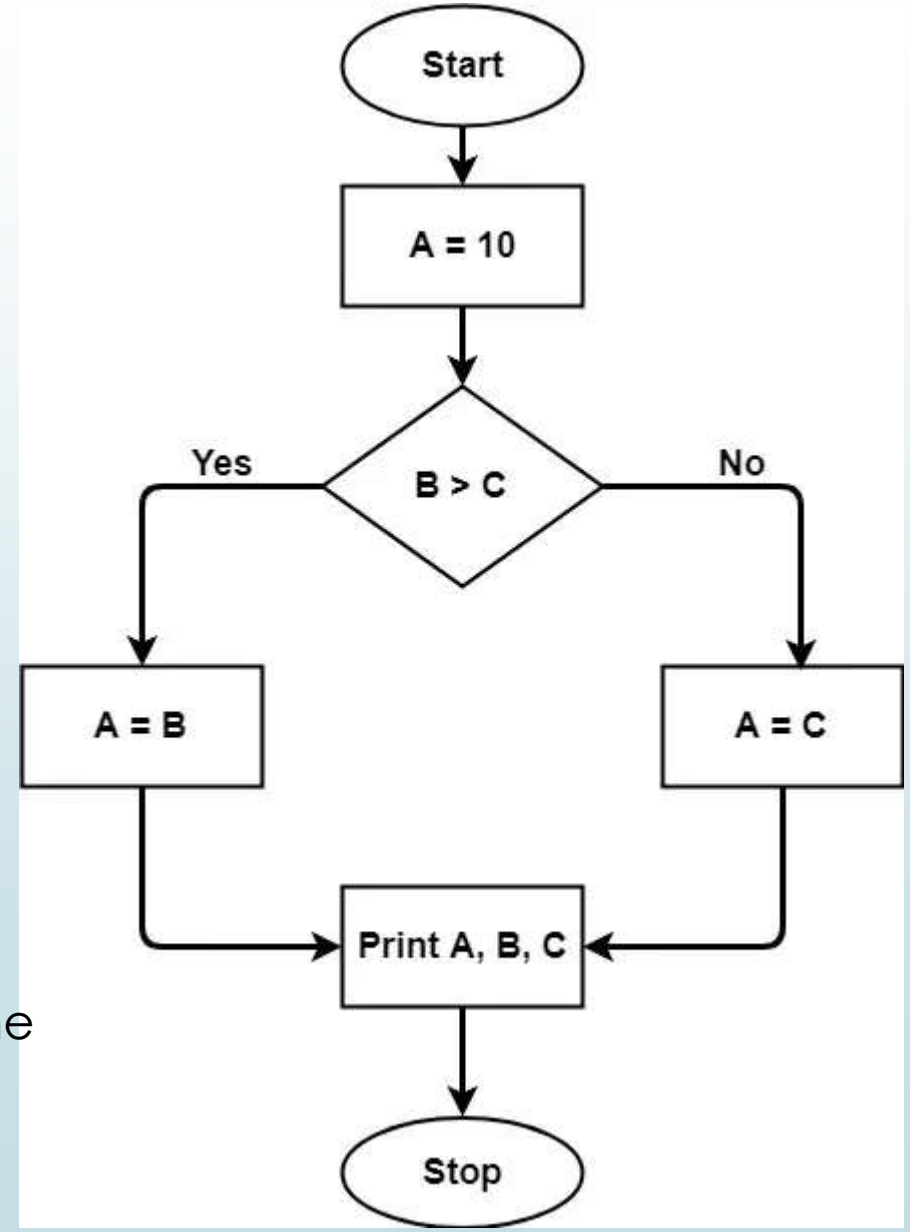
Basis Set – A set of possible execution path of a program

- 1, 7
- 1, 2, 6, 1, 7
- 1, 2, 3, 4, 5, 2, 6, 1, 7
- 1, 2, 3, 5, 2, 6, 1, 7

Numerical 2

The cyclomatic complexity calculated for the above code will be from the control flow graph. The graph shows seven shapes(nodes), and seven lines(edges), hence cyclomatic complexity is e-n+2*p = 7-7+2 = 2.

```
A = 10
  IF B > C THEN
    A = B
  ELSE
    A = C
  ENDIF
Print A
Print B
Print C
```



Control Flow Graph of the above code

Numerical 3

Solve cyclomatic complexity of a simple program to add two numbers together:

```
int a = 1;
int b = 2;
cout<<"a + b = "<<a+b;
```

Numerical 3

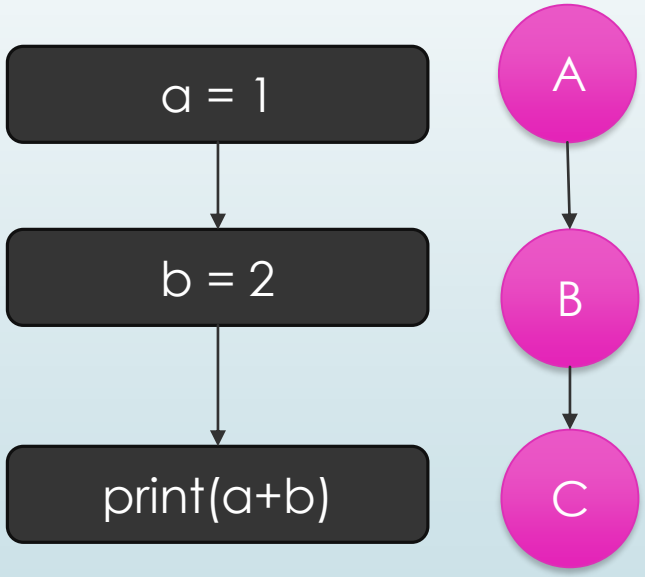
Solve cyclomatic complexity of a simple program to add two numbers together:

```
int a = 1;
int b = 2;
cout<<"a + b = "<<a+b;
```

Solution: There is only 1 connected component(P) that is from A – B – C
So, Cyclomatic complexity = CC = D+1 = 0 + 1

OR CC = Number of regions = 1

OR CC = E - N + 2P
CC = 2 - 3 + 2(1)
CC = 1

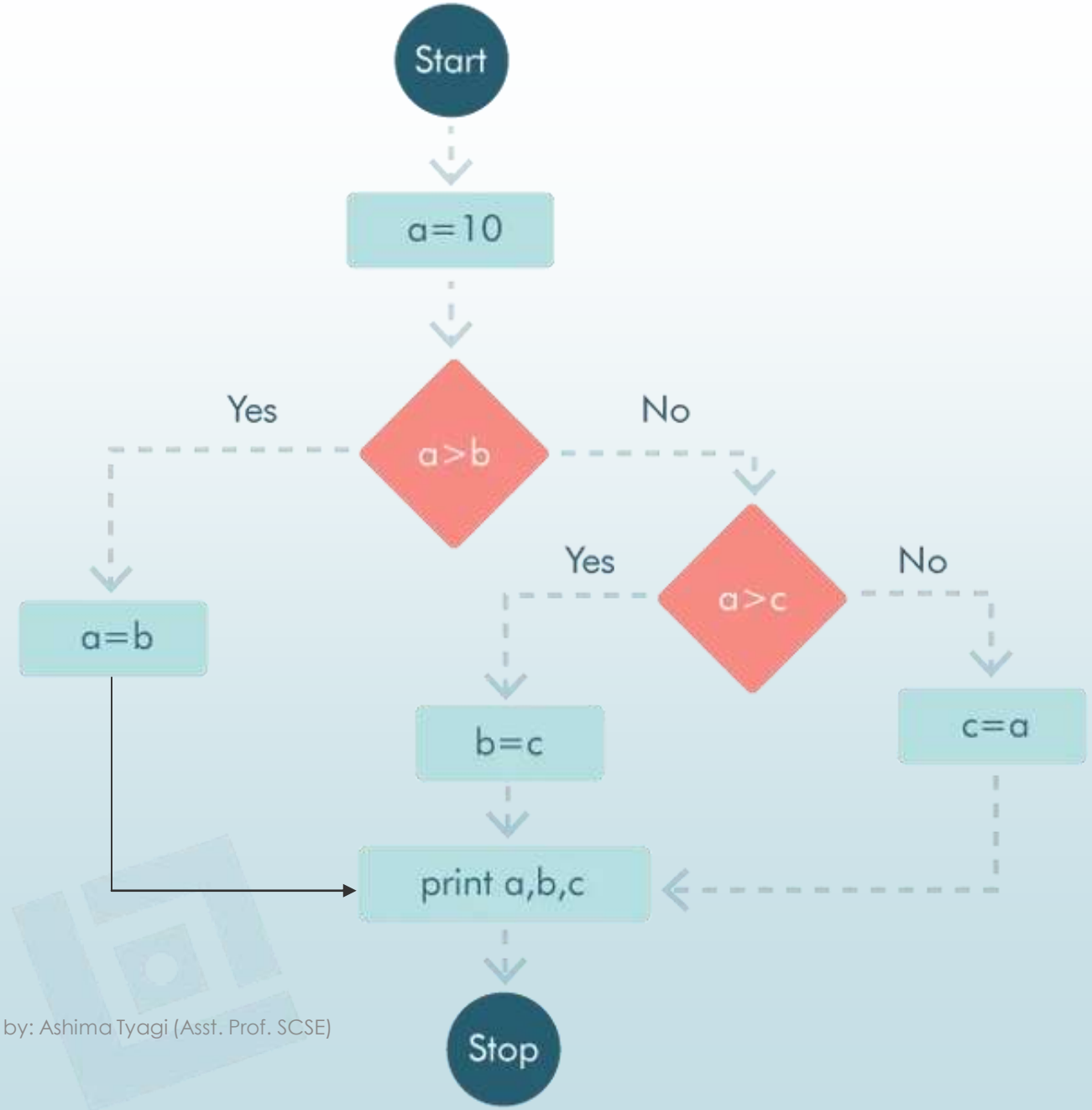


This shows that the graph G will have one path if V (G) = 1
And that one path is A – B – C

Numerical 4

Evaluate the cyclomatic complexity of the given code: a = 10 ;

```
if ( a > b ) then
        a = b ;
else
{
        if ( a > c ) then
                b = c ;
        else
                c = a ;
        end if;
}
end if;
print a
print b
print c
```


Start
a=10
a>b
Yes
No
a=b
a>c
Yes
No
b=c
c=a
print a,b,c
Stop

Using Method 1:

E = 10, N = 9, P = 1
Hence, V(G) = E - N + 2*P = 10 - 9 + 2 * 1
V(G) = 3

Using Method 2:

D = 2
Hence, V(G) = D + 1 = 2 + 1
V(G) = 3

Using Method 3:

Number of regions in the control flow graph = 3
Hence, V(G) = 3

Numerical 5

Deduct the cyclomatic complexity of A program consists of two if statements and one loop.

Numerical 6

Evaluate program cyclomatic complexity with no decision points (e.g., if, while, for, case statements).

Thank You