K.T.S.P.Madal's

Hutatma Rajguru Mahavidyalaya, Rajgurunagar

Tal-Khed, Dist.-Pune 410505.

TY.BSc (Computer Science)

Semester-VI

Subject- Software Testing Tools

According to new CBCS syllabus w.e.f.2019-2020

Prof.S.V.Patole

Department of Computer Science

Hutatma Rajguru Mahavidyalaya,

Rajgurunagar.

Unit 2. Test Cases for Simple Programs

Introduction

Software testing is known as a process for validating and verifying the working of a software/application. It makes sure that the software is working without any errors, bugs, or any other issues and gives the expected output to the user. The software testing process isn't limited to finding faults in the present software but also finding measures to upgrade the software in various factors such as efficiency, usability, and accuracy. So, to test software the software testing provides a particular format called a **Test Case**.

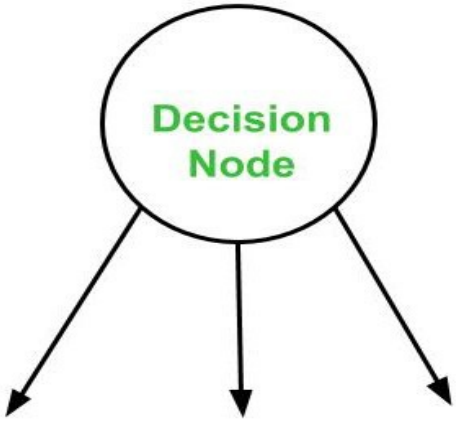
Basic Path Testing

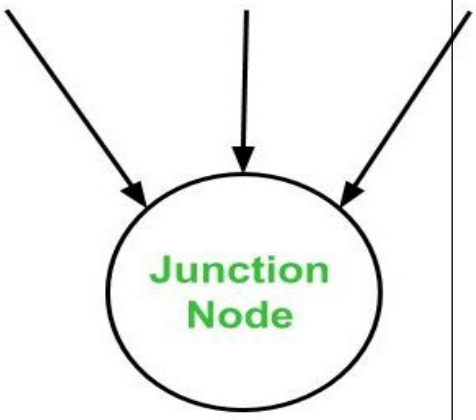
Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module. Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure. To design test cases using this technique, four steps are followed :

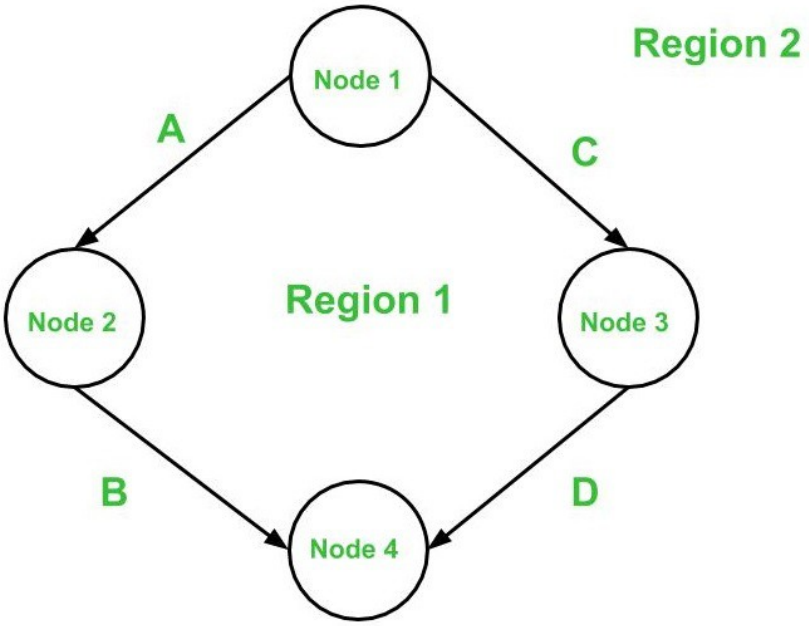
1. Construct the Control Flow Graph
2. Compute the Cyclomatic Complexity of the Graph
3. Identify the Independent Paths
4. Design Test cases from Independent Paths

Let's understand each step one by one. **1. Control Flow Graph –** A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module. A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it. A control graph can also have :

- **Junction Node –** a node with more than one arrow entering it.
- **Decision Node –** a node with more than one arrow leaving it.
- **Region –** area bounded by edges and nodes (area outside the graph is also counted as a region.).

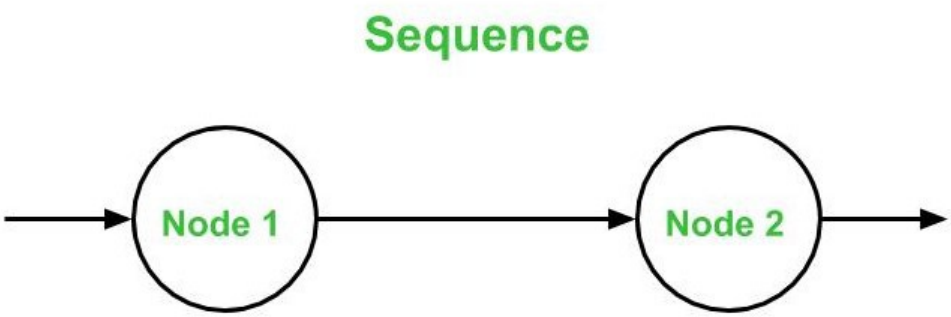

Decision
Node


Junction
Node

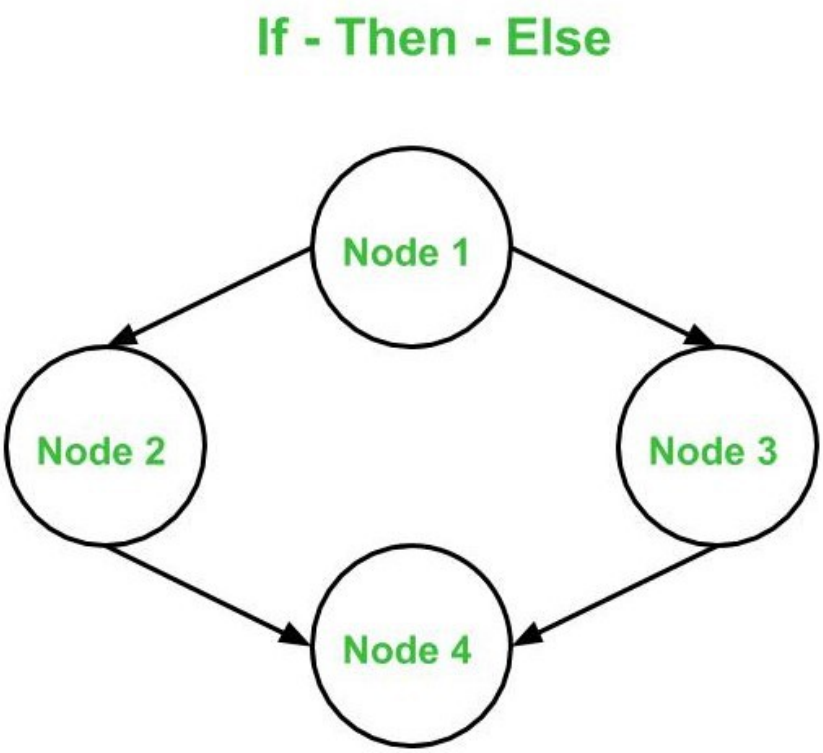

Region 2
Node 1
A
C
Node 2
Region 1
Node 3
B
D
Node 4

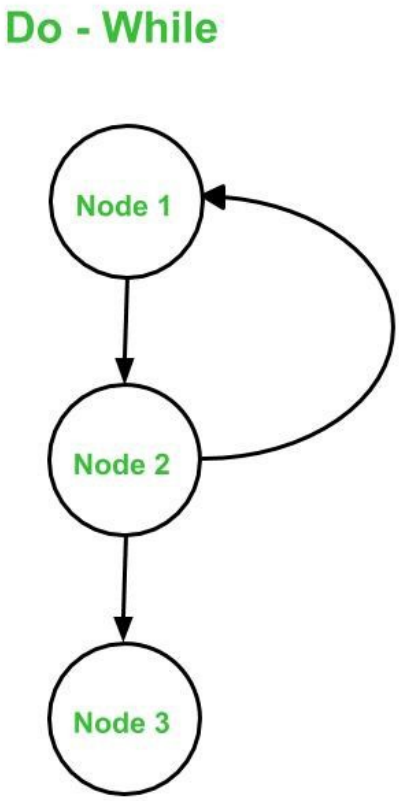
the **notations** used while constructing a flow graph :

- **Sequential Statements –**

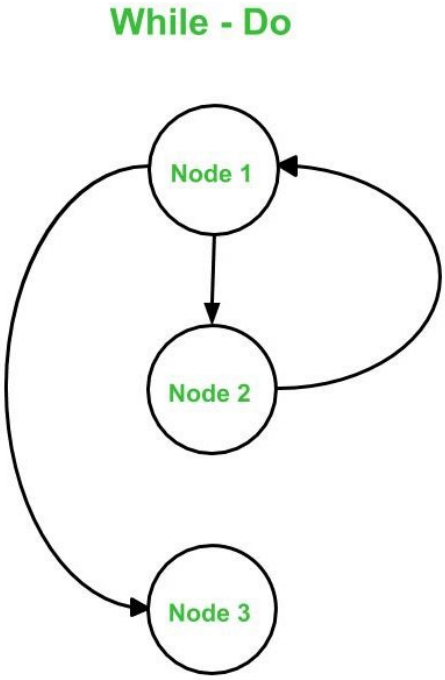


- **If – Then – Else –**



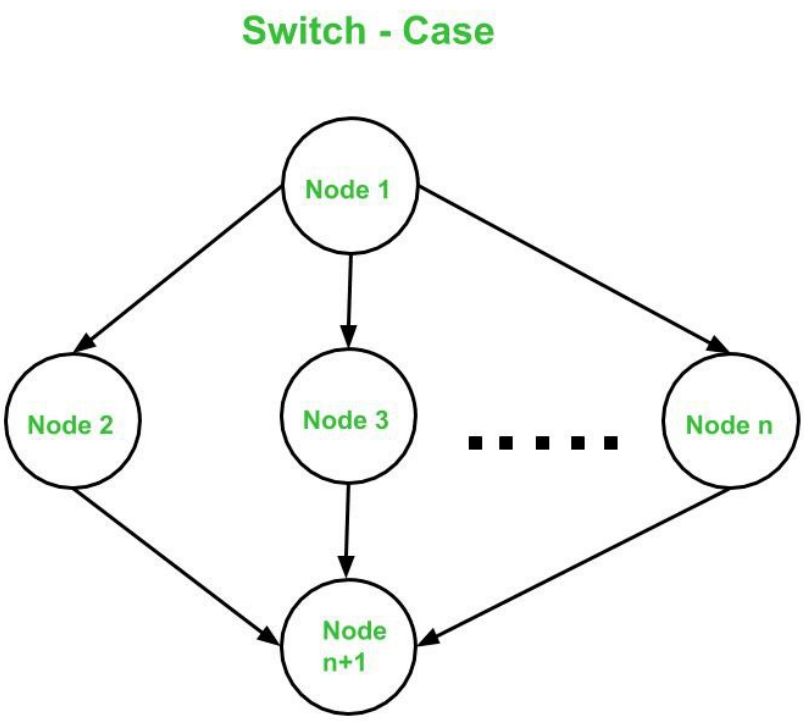


- **Do – While –**



- **While – Do –**

- **Switch – Case –**



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 :**

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

1. Where, e is number of edges, n is number of vertices, P is number of connected components. For example, consider first graph given above,

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

So,

Cyclomatic complexity V(G)

= 4 - 4 + 2 * 1

= 2

1. **Formula based on Decision Nodes :**

V(G) = d + P

1. where, d is number of decision nodes, P is number of connected nodes. For example, consider first graph given above,

where, d = 1 and p = 1

So,

Cyclomatic Complexity V(G)

= 1 + 1

= 2

1. **Formula based on Regions :**

V(G) = number of regions in the graph

1. For example, consider first graph given above,

Cyclomatic complexity V(G)

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

= 2

Hence, using all the three above formulae, the cyclomatic complexity obtained remains same. All these three formulae can be used to compute and verify the cyclomatic complexity of the flow graph. **Note –**

1. For one function [e.g. Main( ) or Factorial( )], only one flow graph is constructed. If in a program, there are multiple functions, then a separate flow graph is constructed for each one of them. Also, in the cyclomatic complexity formula, the value of 'p' is set depending of the number of graphs present in total.
2. If a decision node has exactly two arrows leaving it, then it is counted as one decision node. However, if there are more than 2 arrows leaving a decision node, it is computed using this formula :

d = k - 1

1. Here, k is number of arrows leaving the decision node.

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. This is because the cyclomatic complexity is used as an upper-bound for the number of tests that should be executed in order to make sure that all the statements in the program have been executed at least once. 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

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 errors present in logical conditions can be incorrect boolean operators, missing parenthesis in a booleans expression, error in relational operators, arithmetic expressions, and so on. The common types of logical conditions that are tested using condition testing are-

1. A relation expression, like E1 op E2 where 'E1' and 'E2' are arithmetic expressions and 'OP' is an operator.
2. 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.
3. 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.
4. 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.

2. Data Flow Testing : The data flow test method chooses the test path of a program based on the locations of the definitions and uses all the variables in the program. The data flow test approach is depicted as follows suppose each statement in a program is assigned a unique statement number and that theme function cannot modify its parameters or global variables. For example, with S as its statement number.

DEF (S) = {X | Statement S has a definition of X}

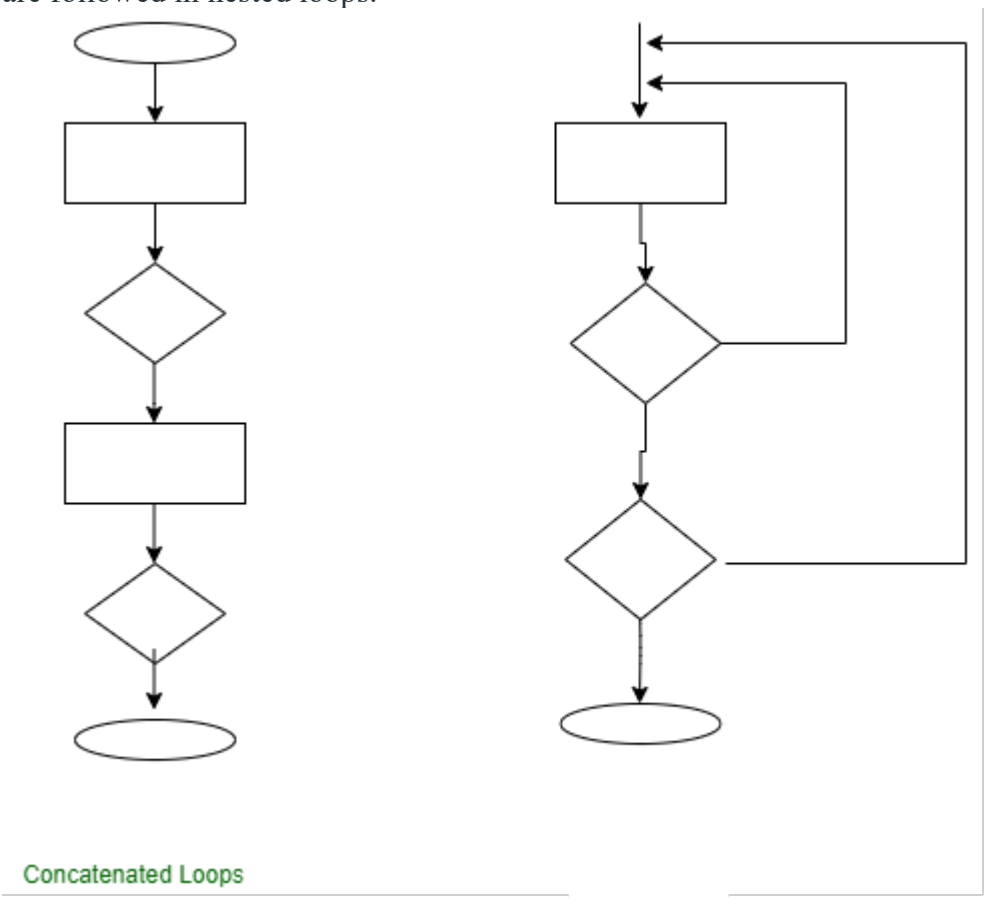
USE (S) = {X | Statement S has a use of X}

If statement S is an if loop statement, them its DEF set is empty and its USE set depends on the state of statement S. The definition of the variable X at statement S is called the line of statement S' if the statement is any way from S to statement S' then there is no other definition of X. A definition use (DU) chain of variable X has the form [X, S, S'], where S and S' denote statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is line at statement S'. A simple data flow test approach requires that each DU chain be covered at least once. This approach is known as the DU test approach. The DU testing does not ensure coverage of all branches of a program. However, a branch is not guaranteed to be covered by DU testing only in rare cases such as then in which the other construct does not have any certainty of any variable in its later part and the other part is not present. Data flow testing strategies are appropriate for choosing test paths of a program containing nested if and loop statements.

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

1. **Simple Loop –** The following set of test can be applied to simple loops, where the maximum allowable number through the loop is n.
    1. Skip the entire loop.
    2. Traverse the loop only once.
    3. Traverse the loop two times.
    4. Make p passes through the loop where p<n.
    5. Traverse the loop n-1, n, n+1 times.
    6.

2. **Concatenated Loops –** If loops are not dependent on each other, contact loops can be tested using the approach used in simple loops. if the loops are interdependent, the steps are followed in nested loops.

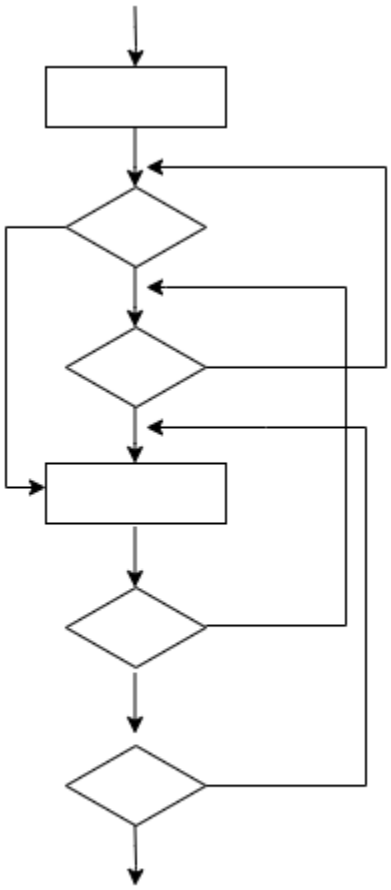


Concatenated Loops

3. **Nested Loops –** Loops within loops are called as nested loops. when testing nested loops, the number of tested increases as level nesting increases. The following steps for testing nested loops are as follows-

1. Start with inner loop. set all other loops to minimum values.
2. Conduct simple loop testing on inner loop.
3. Work outwards.
4. Continue until all loops tested.

4. **Unstructured loops –** This type of loops should be redesigned, whenever possible, to reflect the use of unstructured the structured programming constructs.



Unstructured Loops