Chapter 2 Test Cases for simple Programs

The test case design techniques are broadly classifies as:

1.Specification based Test case Design Techniques:

2.Structure based Test case Design Techniques:

3.Experience based Test case Design Techniques:

White –Box testing

- White-box testing is a way of testing the external functionality of the code by examining and testing the program code that realizes the external functionality.
- White-box testing takes into account the program code ,code structure and internal design flow.
- A number of defects come about because of incorrect translation of requirements and design into program code. Some other defects are created by programming errors.

Write Simple Programs Make Use Of Loops And Control Structure with Test Cases

Code Coverage Testing:

- The percentage of program statements that can be invoked during this phase of testing is called code coverage.
- Code coverage is white-box testing because it requires us to have full access to the code to view what parts of the software we pass through when we run the test and what parts of the software fail.
- The primary purpose of code coverage testing is to ensure that the testing activity covers as much code as possible.
- Code coverage testing is a technique that involves measuring the percentage of a system's code that is being tested by a test suite.
- Code coverage includes creating tests to satisfy some criteria of code coverage(e.g the test designer can create tests to cause all statements in the program to be executed at least once).
- Code coverage testing is determining how much code of the source is being tested. It can be calculated using the formula:
- ***Code Coverage = (Number of lines of code executed) / (Total Number of lines of code in a system component) * 100***
- Code coverage helps in determining how much code of the source is tested which helps in accessing quality of test suite and analysing how software is verified.
- Code coverage testing is a dynamic white-box testing where testing is done by executing the test and the tester has complete access to the code.

- A tester is expected to know and understand the low level design and design and identify the code-based approach and to apply the techniques in the code that is written.
- Commonly used code coverage testing techniques are

1. Statement coverage testing
2. Decision coverage testing
3. Branch coverage testing
4. Loop coverage testing.
5. Path coverage testing

1. Statement Coverage Testing

1. Statement coverage is one of the widely used software testing. It comes under white box testing.

2. Statement coverage technique is used to design white box test cases.
3. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.
4. Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.
5. $$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

6. In white box testing, concentration of the tester is on the working of internal source code and flow chart or flow graph of the code.
7. Generally, in the internal source code, there is a wide variety of elements like operators, methods, arrays, looping, control statements, exception handlers, etc. Based on the input given to the program, some code statements are executed and some may not be executed. The goal of statement coverage technique is to cover all the possible executing statements and path lines in the code.

Example 1:

Source code structure :

- Take input of two values like values for x and y.
- Find the sum of these two values.
- IF the sum is greater than 0, then print " This is the positive result".
- IF the sum is less than 0,then print "This is the negative result".

Source Code:

Example 1

```
1. input(int a, int b) {
2. sum  = a + b;
3. If (sum > 0)
```

```
4. Printf("Positive Result”)
5. Else
6. Printf("Negative result”)
7. }
```

Let us consider two different test case scenarios and find the percentage of statement coverage for given source code.

Test Case 1:

If a=6, b=2

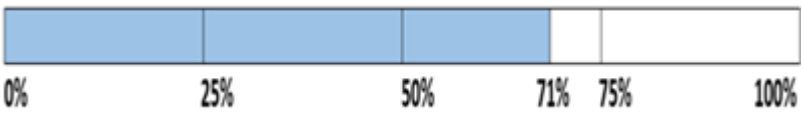
```
1. input(int a, int b) {
2. sum  = a + b;
3. If (sum > 0)
4. Printf("Positive Result”);
5. Else
6. Printf("Negative result”);
7. }
```

It is observed that the value of sum will be 8 which is greater than 0 and as per the condition result will be “This is a positive result”.
Number of executed statements = 5,
Total number of statements = 7

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

Statement coverage = 5/7 *100

=500/7



=71 %

Test Case 2:

If a= -4, y=-3

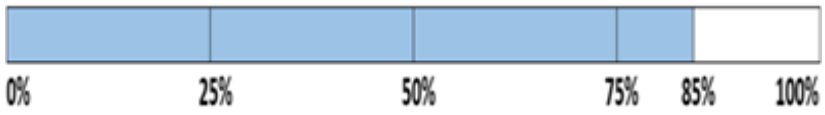
```
1. input(int a, int b) {
2. sum  = a + b;
3. If (sum > 0)

4. Printf("Positive Result”);

5.Else
6.Printf("Negative result”);
7. }
```

It is observed that the value of sum will be -7 which is greater than 0 and as per the condition result will be “This is a negative result”.
Number of executed statements = 6,

Total number of statements = 7

Statement coverage = 6/7 *100

=600/7



=85 %

Example 2:

Test case 1:

If X=100,Y=75

```
1. input(int X,int Y){
2. int z= ((X+Y)/200)*100;
3. if (z>50)
4.       printf (“Pass”);
5. else
6.       printf(“Fail”);
7. }
```

It is observe that the value of z will be 87.5 which is greater than 50 and as per the condition result will be “Pass”.

Total number of statement=7

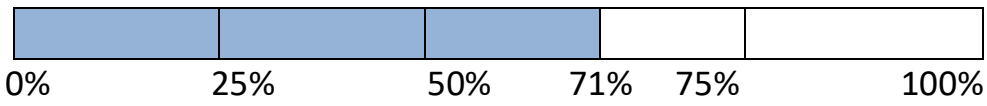
Number of executed statements =5

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

Statement coverage= 5/7 *100

= 500/7

=71%



Test case 2:

If X=20, Y=30

```
1. input(int X,int Y){
2. int z= ((X+Y)/200)*100;
3. if (z>50)
4.       printf (“Pass”);
5. else
6.       printf(“Fail”);
7. }
```

It is observe that the value of z will be 25 which is less than 50 and as per the condition result will be “Fail”.

Total number of statement=7

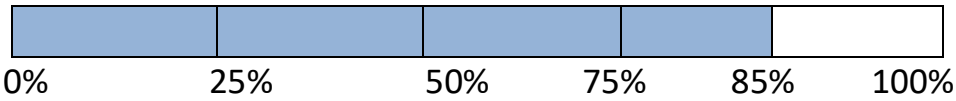
Number of executed statements =6

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

Statement coverage= 6/7 *100

= 600/7

=85%



Advantage of statement coverage:

- It verifies what the written code is expected to do and not to do
- It measures the quality of code written
- It checks the flow of different paths in the program and it also ensure that whether those path are tested or not.

Disadvantage of statement coverage:

- It cannot test the false conditions.
- It does not report that whether the loop reaches its termination condition.
- It does not understand the logical operators.

2.Decision Coverage Testing

1. Decision coverage technique comes under white box testing which gives decision coverage to Boolean values.

2. This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like **do while statement, if statement and case statement** (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

3. Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.

4. Generally, a decision point has two decision values one is true, and another is false that's why most of the times the total number of outcomes is two. The percent of decision coverage can be found by dividing the number of exercised outcome with the total number of outcomes and multiplied by 100.

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

5. In this technique, it is tough to get 100% coverage because sometimes expressions get complicated. Due to this, there are several different methods to report decision coverage. All these methods cover the most important combinations and very much similar to decision coverage. The benefit of these methods is enhancement of the sensitivity of control flow.

Example :

Consider the code to apply on decision coverage technique:

```
1. Test (int a)
2. {   If(a>4)
3. a=a*3
4. Print (a)
5. }
```

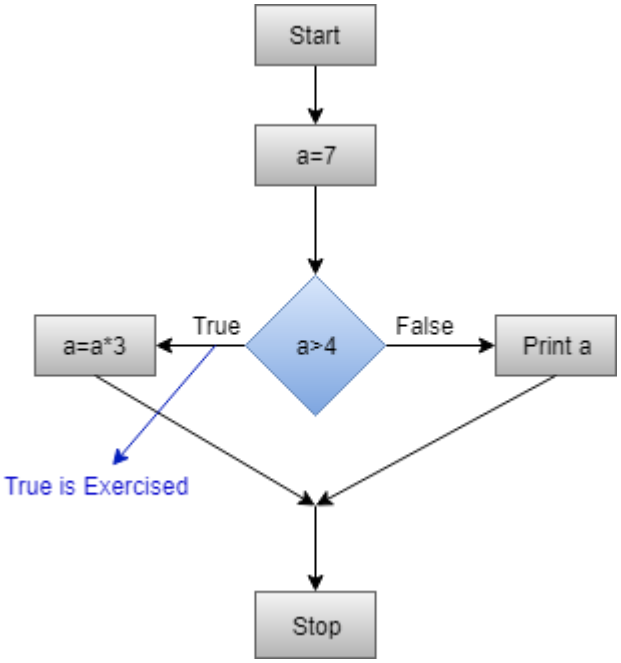
Scenario 1:

Value of a is 7 (a=7)

```
1. Test (int a=7)
2. { if (a>4)
3. a=a*3
4. print (a)
5. }
```

The outcome of this code is "True" if condition (a>4) is checked.

Control flow graph when the value of a is 7.



Calculation of Decision Coverage percent:

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

Decision Coverage = ½*100 (Only "True" is exercised)

=100/2

= 50

Decision Coverage is 50%

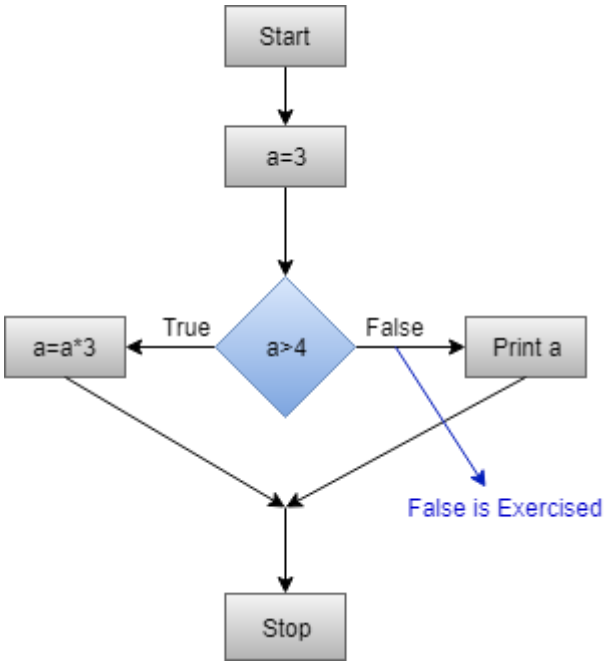
Scenario 2:

Value of a is 3 (a=3)

1. Test (**int** a=3)
2. { **if** (a>4)
3. a=a*3
4. print (a)
5. }

The outcome of this code is False if condition (a>4) is checked.

Control flow graph when the value of a is 3



Calculation of Decision Coverage percent:

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

Decision coverage= ½*100 (Only "False" is exercised)

=100/2

= 50

Decision Coverage = 50%

Result table of Decision Coverage:

Test Case	Value of A	Output	Decision Coverage
1	3	3	50%
2	7	21	50%

Advantages of decision coverage:

- To validate that all the branches in the code are reached.
- To ensure that no branches lead to any abnormality of the program's operation.
- It eliminate problems that occur with statement coverage testing.

Disadvantage of decision Coverage

1.This metric ignores branches within Boolean expression which occur due to short circuit operators

Branch coverage Testing:

1. Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
2. Branch coverage is also sometimes referred to as all edges coverage.
3. Branch coverage technique is a white box testing technique that ensures that every branch of each decision point must be executed.
4. However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two. Decision coverage technique covers all branches of each decision point whereas branch testing covers all branches of every decision point of the code.
5. In other words, branch coverage follows decision point and branch coverage edges. Many different metrics can be used to find branch coverage and decision coverage, but some of the most basic metrics are: finding the percentage of program and paths of execution during the execution of the program.

Branch Testing = (Number of executed branches / Total Number of branches) x 100 %

6. There are several methods to calculate Branch coverage, but path finding is the most common method.

7. In this method, the number of paths of executed branches is used to calculate Branch coverage.

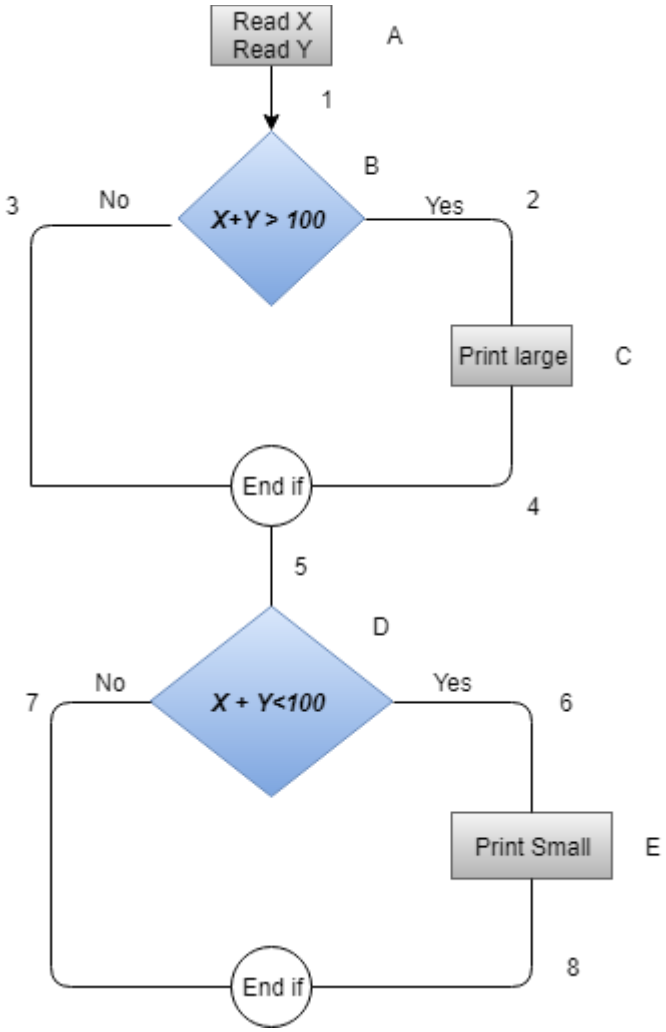
8. Branch coverage technique can be used as the alternative of decision coverage. Somewhere, it is not defined as an individual technique, but it is distinct from decision coverage and essential to test all branches of the control flow graph.

Example:

1. Read X
2. Read Y
3. IF X+Y > 100 THEN
4. Print "Large"
5. ENDIF
6. If X + Y<100 THEN
7. Print "Small"
8. ENDIF

This is the basic code structure where we took two variables X and Y and two conditions. If the first condition is true, then print "Large" and if it is false, then go to the next condition. If the second condition is true, then print "Small."

Control flow graph of code structure



In the above diagram, control flow graph of code is depicted. In the first case traversing through "Yes "decision, the path is **A1-B2-C4-5-D6-E8**, and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path. To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

Path 1 - A1-B2-C4-5-D6-E8

Path 2 - A1-B3-5-D7

Branch Coverage (BC) = Number of paths =2

To calculate percentage branch coverage,one has to find out the minimum number of paths which will ensure that all the edges are covered.

In this case there is no single path which will ensure coverage of all the edges at once.The aim is to cover all possible true/false decision.

$$\text{Branch coverage} = \frac{\text{Number of executed branches}}{\text{Total number of branches}} \times 100$$

Branch coverage for result Yes:

Branch coverage=5/8 * 100= 62.5%

Branch coverage for result No:

Branch coverage= 3/8 * 100= 37.5%

Case	Covered Branches	Path	Branch coverage
Yes	1, 2, 4, 5, 6, 8	A1-B2-C4-D6-E8	62.5%
No	3,7	A1-B3-5-D7	37.5%

Advantages of Branch coverage:

Branch coverage Testing offers the following advantages:

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation
- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

Disadvantage of Branch coverage

1. This metric ignores branches within Boolean expression which occur due to short – circuit operators .
2. It is costly.
3. It is take more time for performing this task.

Loop Coverage Testing

Loop Testing is a type of software testing type that is performed to validate the loops. It is one of the type of Control Structure Testing. Loop testing is a white box testing technique and is used to test loops in the program.

Objectives of Loop Testing:

The objective of Loop Testing is:

- To fix the infinite loop repetition problem.
- To know the performance.
- To identify the loop initialization problems.
- To determine the uninitialized variables.

Types of Loop Testing:

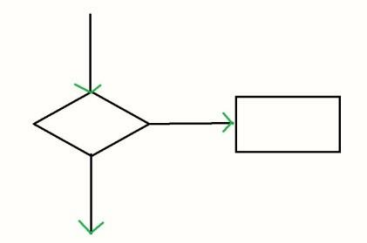
Loop testing is classified on the basis of the types of the loops:

1.Simple Loop Testing:

Testing performed in a simple loop is known as Simple loop testing. Simple loop is basically a normal “for”, “while” or “do-while” in which a condition is given and loop runs and terminates according to true and false occurrence of the condition respectively. This type of testing is performed basically to test the condition of the loop whether the condition is sufficient to terminate loop after some point of time.

Example:

```
while(condition)
  {
   statement(s);
  }
```

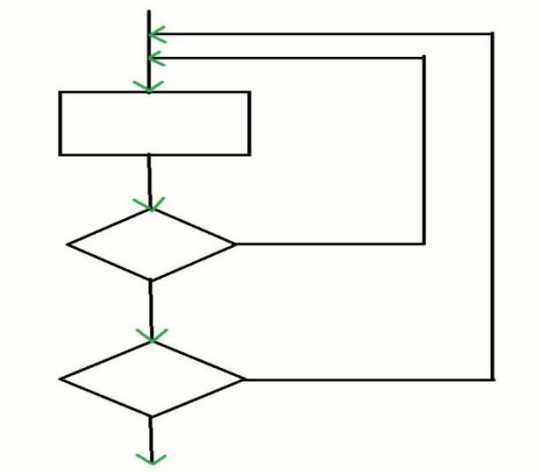


2.Nested Loop Testing:

Testing performed in a nested loop in known as Nested loop testing. Nested loop is basically one loop inside the another loop. In nested loop there can be finite number of loops inside a loop and there a nest is made. It may be either of any of three loops i.e., for, while or do-while.

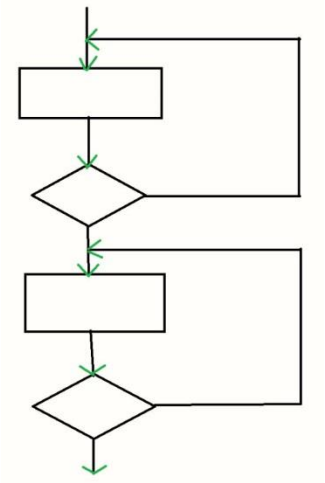
Example:

```
while(condition 1)
  {
   while(condition 2)
    {
     statement(s);
    }
  }
```



3.Concatenated Loop Testing:
Testing performed in a concatenated loop is known as Concatenated loop testing. It is performed on the concatenated loops. Concatenated loops are loops after the loop. It is a series of loops. Difference between nested and concatenated is that in nested loop is inside the loop but here loop is after the loop.
Example:

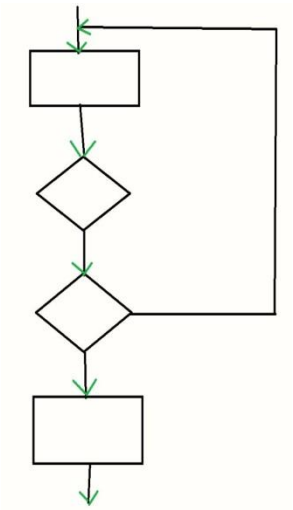
```
while(condition 1)
  {
   statement(s);
  }
 while(condition 2)
 {
  statement(s);
 }
```



4.Unstructured Loop Testing:
Testing performed in an unstructured loop is known as Unstructured loop testing. Unstructured loop is the combination of nested and concatenated loops. It is basically a group of loops that are in no order.

Example:

```
while()
  {
   for()
    {}
   while()
    {}
  }
```



Advantages of Loop Testing:

Loop testing limits the number of iterations of loop.

- Loop testing ensures that program doesn't go into infinite loop process.
- Loop testing endures initialization of every used variable inside the loop.
- Loop testing helps in identification of different problems inside the loop.
- Loop testing helps in determination of capacity.

Disadvantages of Loop Testing:

- Loop testing is mostly effective in bug detection in low-level software.
- Loop testing is not useful in bug detection.

Example 1: In order to have loop coverage,testers should exercise the tests given as follows:

These are loops in which their loop body contains no other loops (the innermost loop in case of nested).

In order to have loop coverage, testers should exercise the tests given below.

Test 1 :Design a test in which loop body shouldn't execute at all (i.e. zero iterations)

Test 2 :Design a test in which loop–control variable be negative (Negative number of iterations)

Test 3 :Design a test in which loop iterates only once

Test 4 :Design a test in which loop iterates twice

Test 5 :Design a test in which loop iterates certain number of times , say m where m < maximum number of iterations possible

Test 6 :Design a test in which loop iterates one less than the maximum number of iterations

Test 7 :Design a test in which loop iterates the maximum number of iterations

Test 8 :Design a test in which loop iterates one more than the maximum number of iterations

Consider the below code example which applies all the conditions specified.

public class SimpleLoopTest {

private int[] numbers = {5,-77,8,-11,4,1,-20,6,2,10};

```
/** Compute total of  positive numbers in the array
 *  @param numItems number of items to total.
 */
public int findSum(int numItems)
{
    int total = 0;
    if (numItems <= 10)
    {
        for (int count=0; count < numItems; count = count + 1)
        {
          if (numbers[count] > 0)
             {
                total = total + numbers[count];
             }
        }
    }
    return total;
}
}
```

public class TestPass extends TestCase {

```
public void testname() throws Exception {

    SimpleLoopTest s = new SimpleLoopTest();
    assertEquals(0, s.findSum(0));    //Test 1
    assertEquals(0, s.findSum(-1));   //Test 2
    assertEquals(5, s.findSum(1));    //Test 3
    assertEquals(5, s.findSum(2));    //Test 4
    assertEquals(17, s.findSum(5));   //Test 5
    assertEquals(26, s.findSum(9));   //Test 6
    assertEquals(36, s.findSum(10));  //Test 7
    assertEquals(0, s.findSum(11));   //Test 8
}
}
```

Example 2. Consider the below C code example to be tested,it display first n numbers.

```
# include<stdio.h>
int main()
{
  int i=1,n;
printf("Enter n:");
scanf("%d",&n);
while(i<=n)
{
printf("%d \n",i);
i++;
}
return 0;
}
```

Some test cases with test data:

Loop Coverage	Test case	n	Output
While(i<=n)	1	4	1 2 3 4
	2	6	1 2 3 4 5 6
	3	-2	No output
	4	0	No output

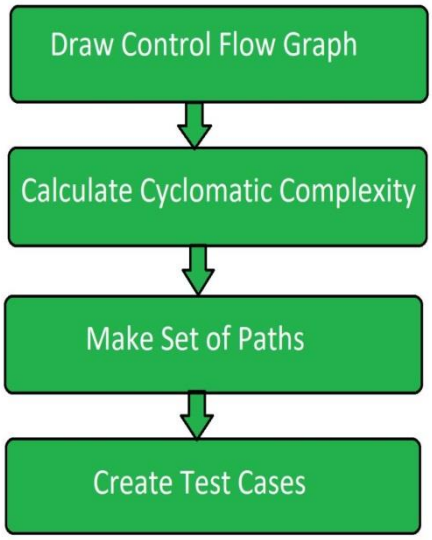
Path coverage Testing:

Path Testing is a method that is used to design the test cases. In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution. In this method Cyclomatic Complexity is used to determine the number of linearly independent paths and then test cases are generated for each path.

It give complete branch coverage but achieves that without covering all possible paths of the control flow graph. McCabe's Cyclomatic Complexity is used in path testing. It is a structural testing method that uses the source code of a program to find every possible executable path.

$$\text{Path coverage} = \frac{\text{Total path exercised}}{\text{Total number of paths in the program}} \times 100$$

Path Testing Process:



- **Control Flow Graph:**
  Draw the corresponding control flow graph of the program in which all the executable paths are to be discovered.
- **Cyclomatic Complexity:**
  After the generation of the control flow graph, calculate the cyclomatic complexity of the program using the following formula.
- `McCabe's Cyclomatic Complexity = E - N + 2P`
- `Where,`
- `E = Number of edges in control flow graph`
- `N = Number of vertices in control floe graph`

  `P = Program factor`
- **Make Set:**
  Make a set of all the path according to the control floe graph and calculated cyclomatic complexity. The cardinality of set is equal to the calculated cyclomatic complexity.
- **Create Test Cases:**
  Create test case for each path of the set obtained in above step.

Path Testing Techniques:

- **Control Flow Graph:**
  The program is converted into control flow graph by representing the code into nodes and edges.
- **Decision to Decision path:**
  The control flow graph can be broken into various Decision to Decision paths and then collapsed into individual nodes.

- **Independent paths:**
  Independent path is a path through a Decision to Decision path graph which cannot be reproduced from other paths by other methods.

  Advantages of Path Testing:

  1. Path testing method reduces the redundant tests.
  2. Path testing focuses on the logic of the programs.
  3. Path testing is used in test case design.