White box testing

Outline

- White box testing
  - Why to derive test cases?
  - White box testing techniques
- Basic path testing
  - Flow graph notation
  - Independent program path
  - Deriving test cases
  - Graph matrices
- Control structure testing
  - Condition testing
  - Data flow testing
  - Loop testing : simple loops, concatenated loops, nested loops, and unstructured loops.

White box testing

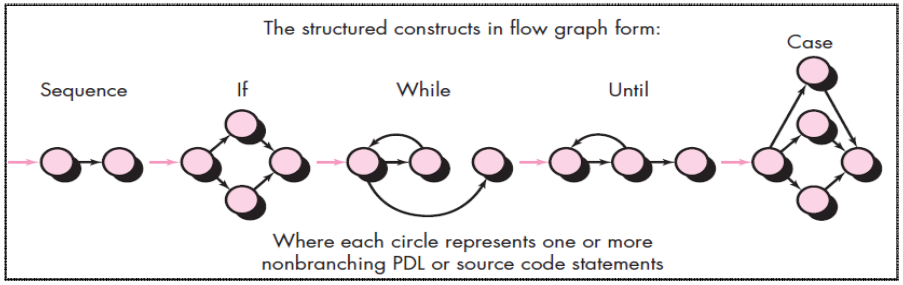
- White-box testing is a test-case design philosophy that uses the control structure to derive test cases.
- Using white-box testing methods, derive test cases that
  - (1) guarantee that all independent paths within a module have been exercised at least once
  - (2) exercise all logical decisions on their true and false sides
  - (3) execute all loops at their boundaries and within their operational bounds
  - (4) exercise internal data structures to ensure their validity.
- White box testing techniques :
  - Basic path testing
  - Control structure testing

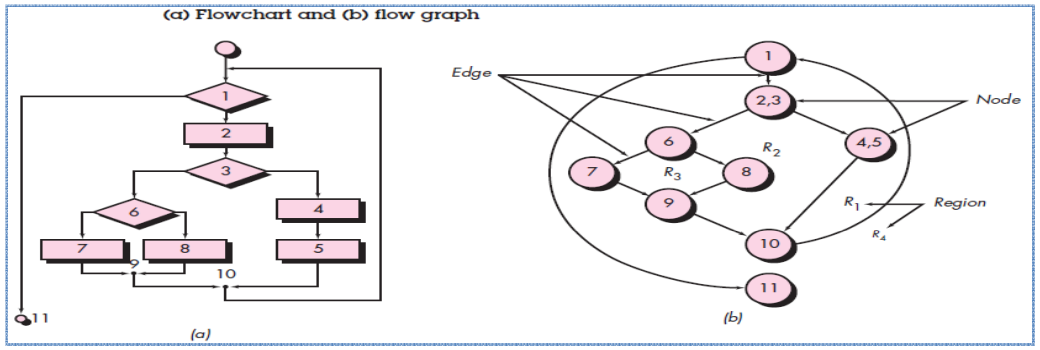
Basis path testing

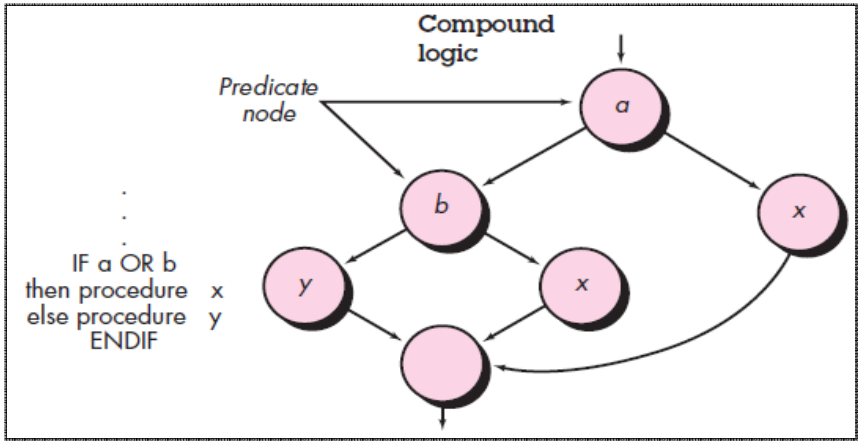
- Basis path testing is a white-box testing technique first proposed by Tom McCabe.
- Test cases derived are guaranteed to execute every statement in the program at least one time during testing.
- Various techniques explained in this testing are as follows.
  - Flow graph notation
  - Independent program path
  - Deriving test cases
  - Graph matrices

Basis path testing (Contd..)

- Flow graph notation : A flow graph should be drawn only when the logical structure of a component is complex.
- The flow graph allows to trace program paths more readily.







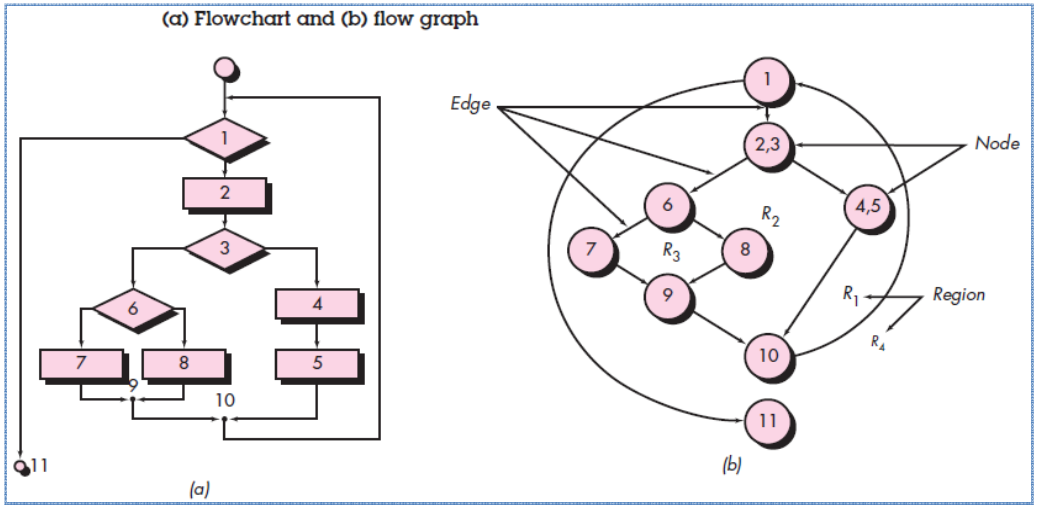
- When compound conditions are encountered in a procedural design, the generation of a flow graph becomes slightly more complicated.
- A compound condition occurs when one or more Boolean operators (logical OR, AND, NAND, NOR) is present in a conditional statement.

Basis path testing (Contd..)

- Independent program path : An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
- Cyclomatic complexity is a quantitative measure of the logical complexity of a program.
- The value for cyclomatic complexity defines the number of independent paths in a program.
- It provides the number of tests that must be conducted to ensure that all statements have been executed at least once.
- Cyclomatic complexity *V(G) for a flow graph G is defined as*

  V(G) = E - N + 2

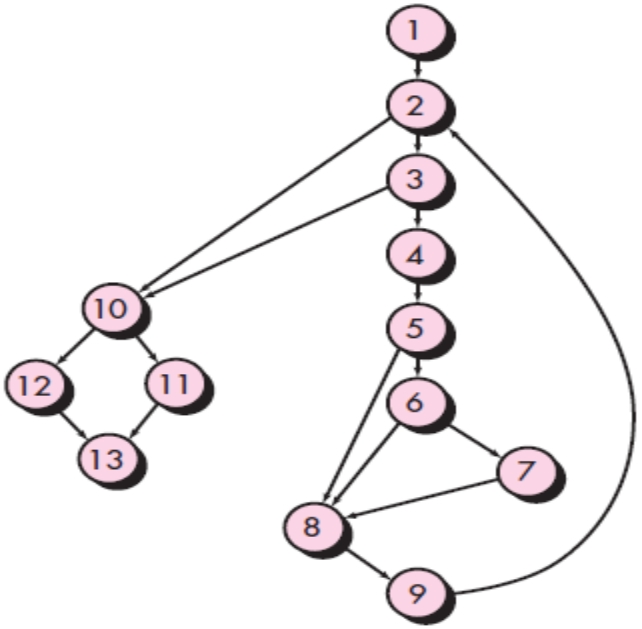
  where *E is the number of flow graph edges and N is the number of flow* graph nodes.



- *V(G) = 11 edges - 9 nodes + 2 = 4.*
- *4* number of tests that must be designed and executed to guarantee coverage of all program statements.

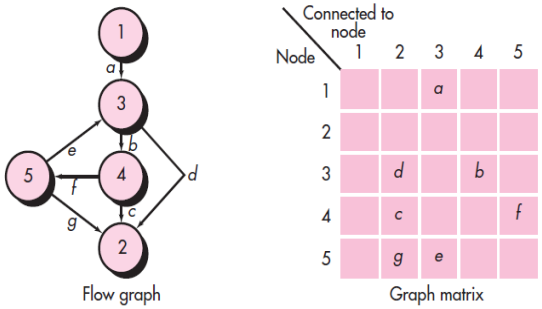
Basis path testing (Contd..)

- Deriving test cases : The following steps can be applied to derive the basis set:
- 1. Using the design or code as a foundation, draw a corresponding flow graph.
- 2. Determine the cyclomatic complexity of the resultant flow graph.
- 3. Determine a basis set of linearly independent paths.
- 4. Prepare test cases that will force execution of each path in the basis set.



3. Determine a basis set of linearly independent paths. Six paths identified are :
Path 1: 1-2-10-11-13
Path 2: 1-2-10-12-13
Path 3: 1-2-3-10-11-13
Path 4: 1-2-3-4-5-8-9-2-. . .
Path 5: 1-2-3-4-5-6-8-9-2-. . .
Path 6: 1-2-3-4-5-6-7-8-9-2-. . .
In this case, nodes 2, 3, 5, 6, and 10 are predicate nodes.

Basis path testing (Contd..)



- Graph matrices : A data structure, called a *graph matrix, can be quite* useful for developing a software tool that assists in basis path testing.
- The graph matrix is nothing more than a tabular representation of a flow graph.
- A powerful tool for evaluating program control structure during testing.
- Using these techniques, the analysis required to design test cases can be partially or fully automated.

Control structure testing

- Control structure testing broaden test coverage and improve the quality of white-box testing.
- Condition testing : tests the logical conditions contained in a program module.

  E_1 <relational operator> E_2

  where E_1 and E_2 are arithmetic expression.
- This testing tests the boolean condition, compound condition, boolean expression, relational expression, arithmetic expression.
- This test ensures all the following errors are tested :
  - Boolean operator errors, Boolean variable errors, Boolean parenthesis errors, relational operator errors, and arithmetic expression errors.

Control structure testing (contd..)

- Data flow testing : The data flow testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.
- For a statement with S as its statement number,

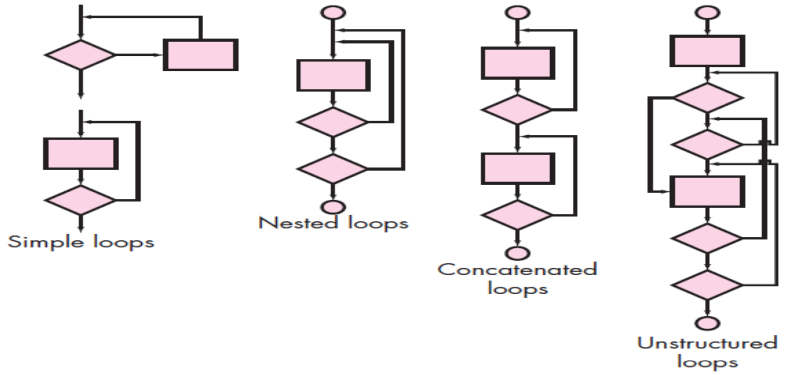
  DEF(S) {X | statement S contains a definition of X}

  USE(S) {X | statement S contains a use of X}
- If statement S is an if or loop statement, its DEF set is empty and its USE set is based on the condition of statement S.
- A definition-use (DU) chain of variable X is of the form [X, S, S'], where S and S' are statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is live at statement S'.
- Data flow testing strategy is that every DU chain be covered at least once.

Control flow structure(contd..)

- Loop testing : Loop testing is a white-box testing technique that focuses exclusively on the four different classes of loops, simple loops, concatenated loops, nested loops, and unstructured loops.

Simple loops The following set of tests can be applied to simple loops, where *n* is the maximum number of allowable passes through the loop.
1. Skip the loop entirely.
2. Only one pass through the loop.
3. Two passes through the loop.
4. m passes through the loop where m < n.
5. *n - 1, n, n + 1 passes through the loop.*



Concatenated loops : concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other.
However, if two loops are concatenated, the approach applied to nested loops is recommended.

Nested loop : The number of possible tests would grow geometrically as the level of nesting increases. The following set of tests can be applied:
1. Start at the innermost loop. Set all other loops to minimum values.
2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.
3. Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to “typical” values.
4. Continue until all loops have been tested.