BASIS PATH TESTING

Basis path testing is a white-box testing technique. The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

Flow Graph Notation

Before the basis path method can be introduced, a simple notation for the representation of control flow, called a flow graph (or program graph) must be introduced.

The flow graph depicts logical control flow using the notation illustrated in Figure below. Each structured construct has a corresponding flow graph symbol.

To illustrate the use of a flow graph, we consider the procedural design representation in Figure A. Here, a flowchart is used to depict program control structure.

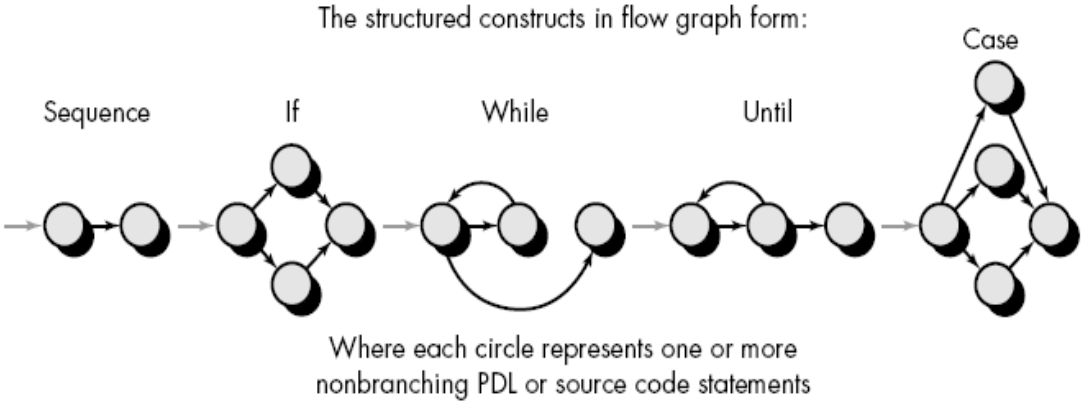


Figure B maps the flowchart into a corresponding flow graph (assuming that no compound conditions are contained in the decision diamonds of the flowchart). Each circle, called a flow graph node, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node. The arrows on the flow graph, called edges or links, represent flow of control and are analogous to flowchart arrows. An edge must terminate at a node, even if the node does not represent any procedural statements (e.g., see the symbol for the if-then-else construct). Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph as a region.

Cyclomatic Complexity

Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. When used in the context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

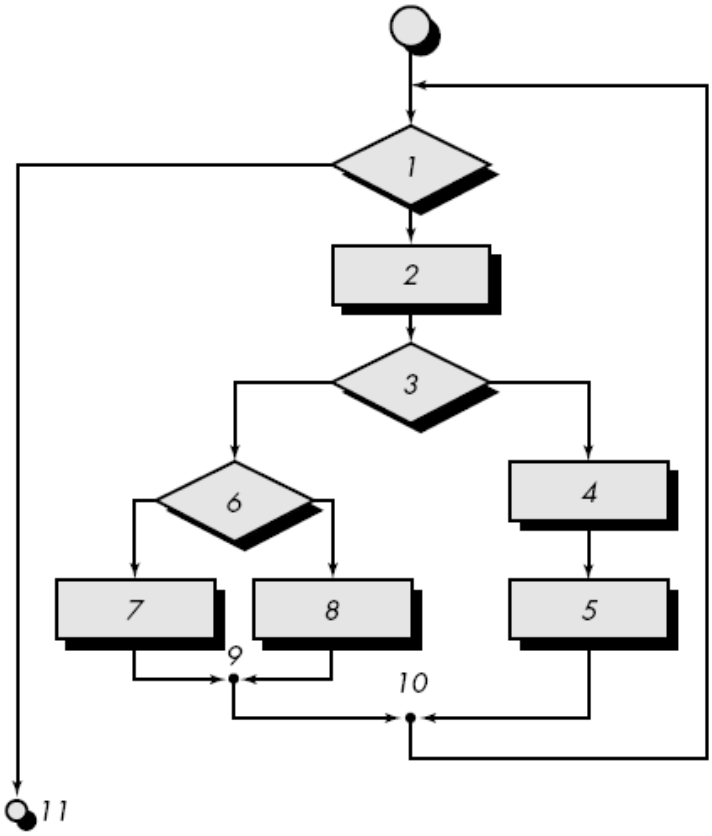
An ***independent path*** is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined. For example, a set of independent paths for the flow graph illustrated in Figure is:

path 1: 1-11

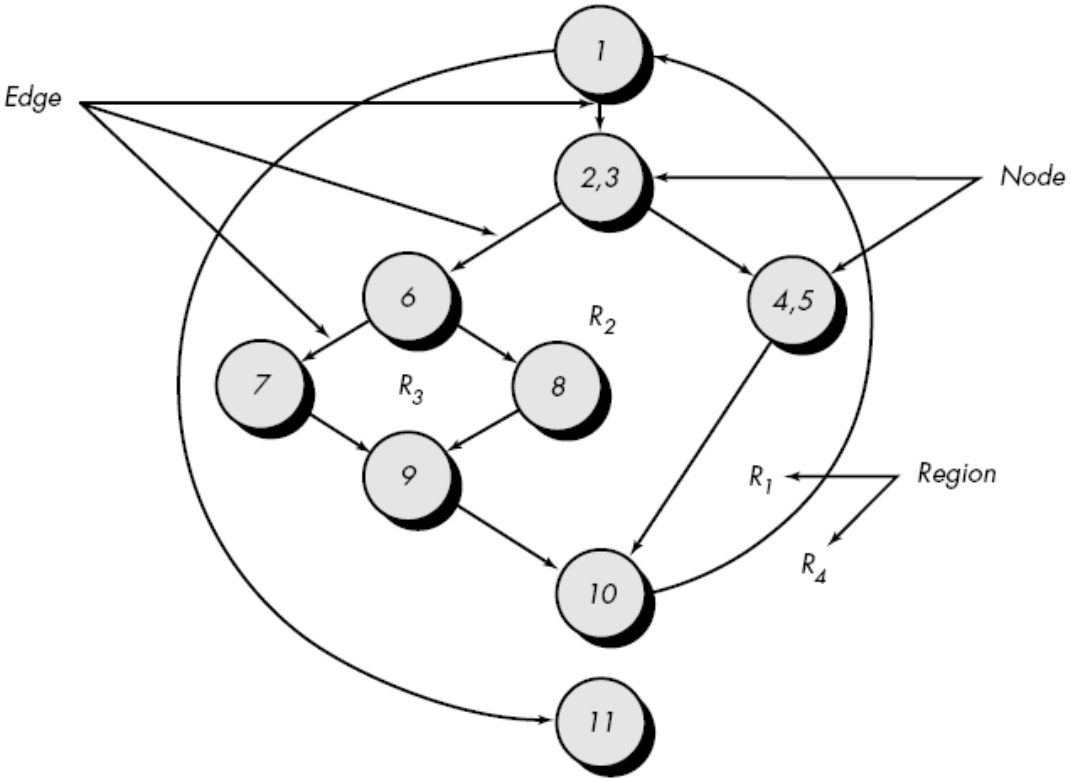
path 2: 1-2-3-4-5-10-1-11

path 3: 1-2-3-6-8-9-10-1-11

path 4: 1-2-3-6-7-9-10-1-11



(A)



(B)

How do we know how many paths to look for? The computation of cyclomatic complexity provides the answer.

Cyclomatic complexity has a foundation in graph theory and provides us with an extremely useful software metric. Complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. 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, N is the number of flow graph nodes.

3. Cyclomatic complexity, V(G), for a flow graph, G, is also defined as

V(G) = P + 1

where P is the number of predicate nodes contained in the flow graph G.

Referring once more to the flow graph in Figure B, the cyclomatic complexity

can be computed using each of the algorithms just noted:

1. The flow graph has four regions.

2. V(G) = 11 edges _ 9 nodes + 2 = 4.

3. V(G) = 3 predicate nodes + 1 = 4.

Therefore, the cyclomatic complexity of the flow graph in Figure B is 4.

Deriving Test Cases

1. **Using the design or code as a foundation, draw a corresponding flow graph.**

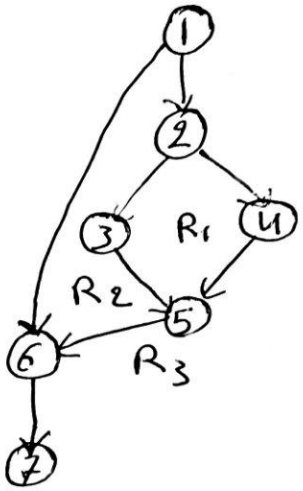
2**. Determine the cyclomatic complexity**

3. **Determine a basis set of linearly independent paths.**

4. **Assign values to independent paths**

Example :-

```
1 if A = 10
2 then if B > C
3 then A = B
4 else A = C
5 ENDIF
6 ENDIF
7 { print A
  { print B
  { print C
```



cyclomatic complexity = E − N + 2 = 8 − 7 + 2 = 3

Path 1 : 1, 2, 3, 5, 6, 7
path 2 : 1, 2, 4, 5, 6, 7
path 3 : 1, 6, 7