Deriving Test Cases

The basis path testing method can be applied to a procedural design or to source code. 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.** Referring to the PDL for average in Figure below, a flow graph is created by numbering those PDL statements that will be mapped into corresponding flow graph nodes.

2**. Determine the cyclomatic complexity** of the resultant flow graph:

V(G) = 6 regions

V(G) = 17 edges _ 13 nodes + 2 = 6

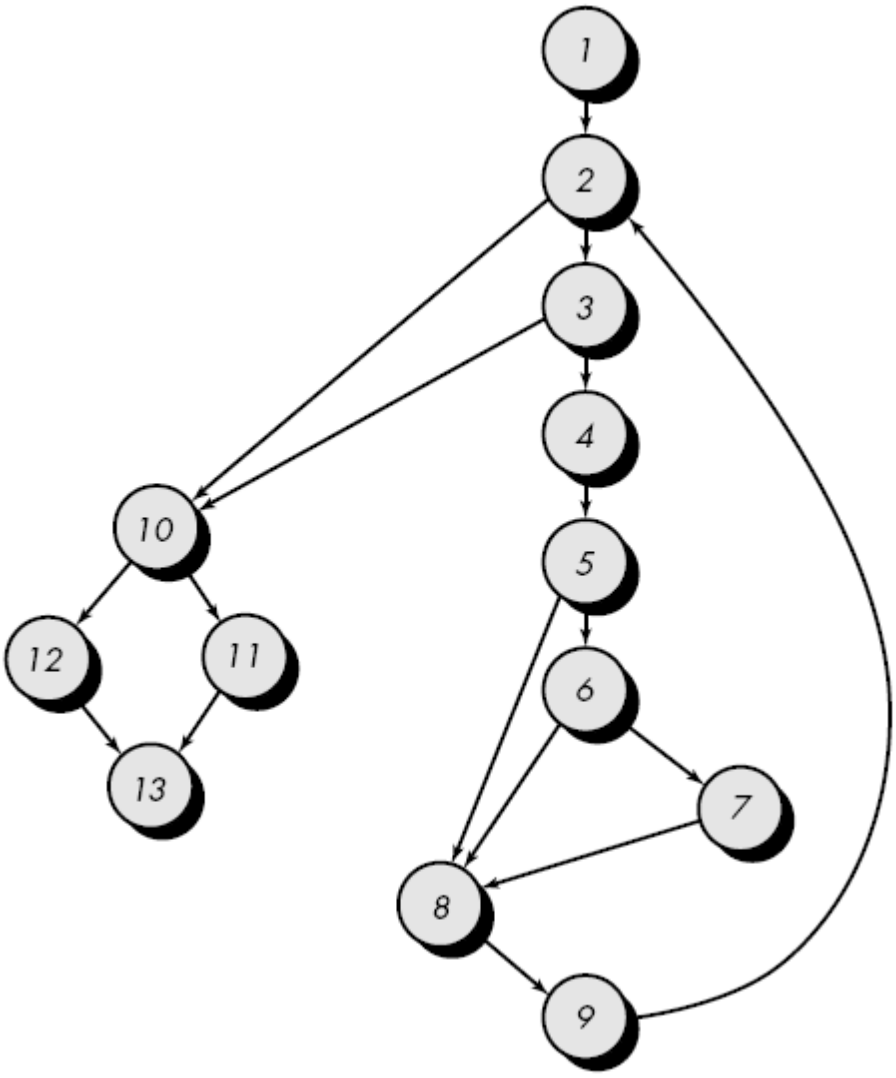
V(G) = 5 predicate nodes + 1 = 6

```
PROCEDURE average;

    *  This procedure computes the average of 100 or fewer
       numbers that lie between bounding values; it also computes the
       sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
       minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;
    sum = 0;
    DO WHILE value[i] <> -999 AND total.input < 100
4       increment total.input by 1;
        IF value[i] > = minimum AND value[i] < = maximum
5           THEN increment total.valid by 1;
7                sum = s sum + value[i]
            ELSE skip
        ENDIF
8       increment i by 1;
9   ENDDO
    IF total.valid > 0
11      THEN average = sum / total.valid;
12      ELSE average = -999;
13  ENDIF
END average
```



3. **Determine a basis set of linearly independent paths.** The value of V(G) provides the number of linearly independent paths through the program control structure. In the case of procedure average, we expect to specify six paths:

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-. . .

4. **Assign values to independent paths**

Path 1 test case:
value(k) = valid input, where k < i for $2 \leq i \leq 100$
value(i) = _999 where $2 \leq i \leq 100$
Expected results: Correct average based on k values and proper totals.
Note: Path 1 cannot be tested stand-alone but must be tested as part of path 4, 5, and 6 tests.

Path 2 test case:
value(1) = _999
Expected results: Average = _999; other totals at initial values.

Path 3 test case:
Attempt to process 101 or more values.
First 100 values should be valid.
Expected results: Same as test case 1.

Path 4 test case:
value(i) = valid input where i < 100
value(k) < minimum where k < i
Expected results: Correct average based on k values and proper totals.

Path 5 test case:
value(i) = valid input where i < 100
value(k) > maximum where k <= i
Expected results: Correct average based on n values and proper totals.

Path 6 test case:
value(i) = valid input where i < 100
Expected results: Correct average based on n values and proper totals.
Each test case is executed and compared to expected results. Once all test cases have been completed, the tester can be sure that all statements in the program have been executed at least once.
It is important to note that some independent paths (e.g., path 1 in our example) cannot be tested in stand-alone fashion. That is, the combination of data required to traverse the path cannot be achieved in the normal flow of the program. In such cases, these paths are tested as part of another path test.