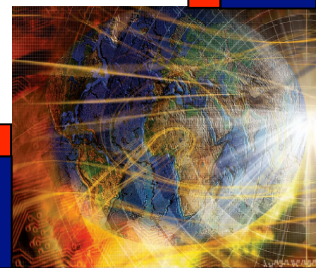
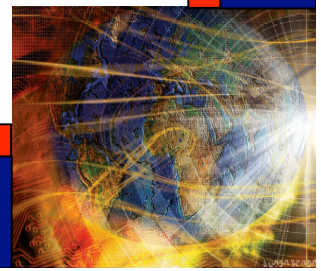
Chapter 9

Path Testing–Part 1



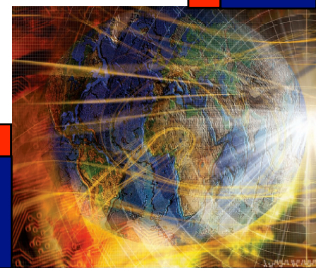
Structural (Code-Based) Testing

- Complement of/to Functional Testing
- Based on Implementation
- Powerful mathematical formulation
  - program graph
  - define-use path
  - Program slices
- Basis for Coverage Metrics (a better answer for gaps and redundancies)
- Usually done at the unit level
- Not very helpful to identify test cases
- Extensive commercial tool support

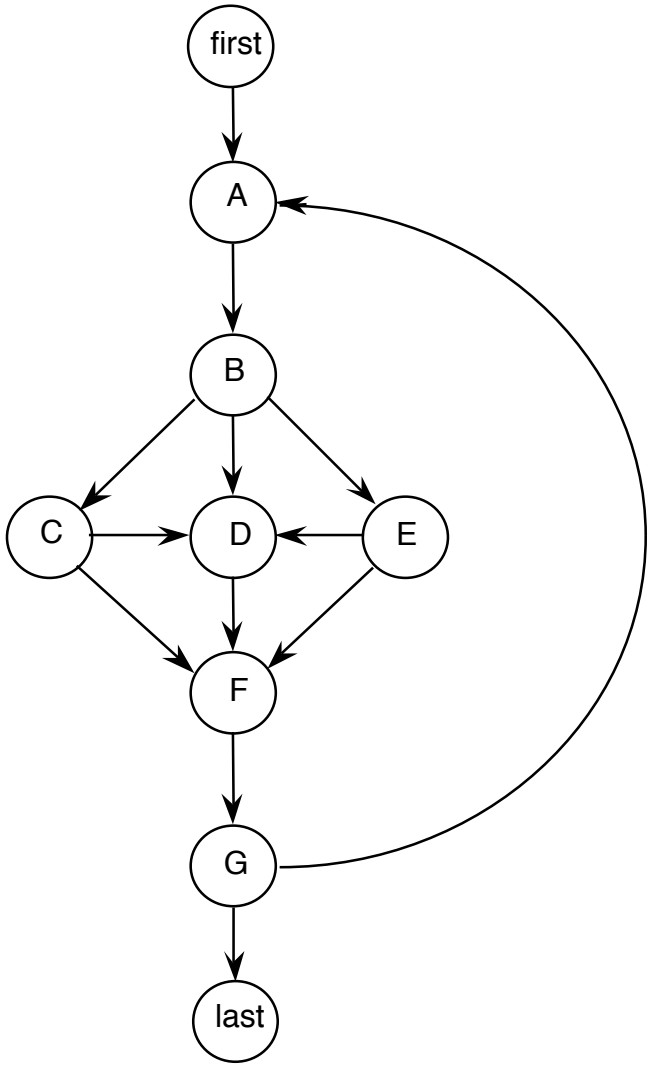


Path Testing

- Paths derived from some graph construct.
- When a test case executes, it traverses a path.
- Huge number of paths implies some simplification is needed.
- Big Problem: infeasible paths.
- Big Question: what kinds of faults are associated with what kinds of paths?
- By itself, path testing can lead to a false sense of security.



The Common Objection: Trillions of Paths



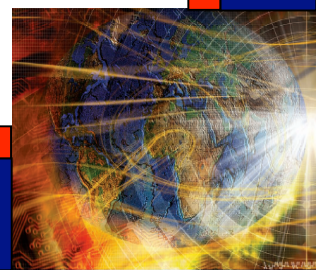
If the loop executes up to 18 times, there are 4.77 Trillion paths.

(5^{18} = 3,814,697,265,625)
(Actually, it is 4,768,371,582,030 paths)

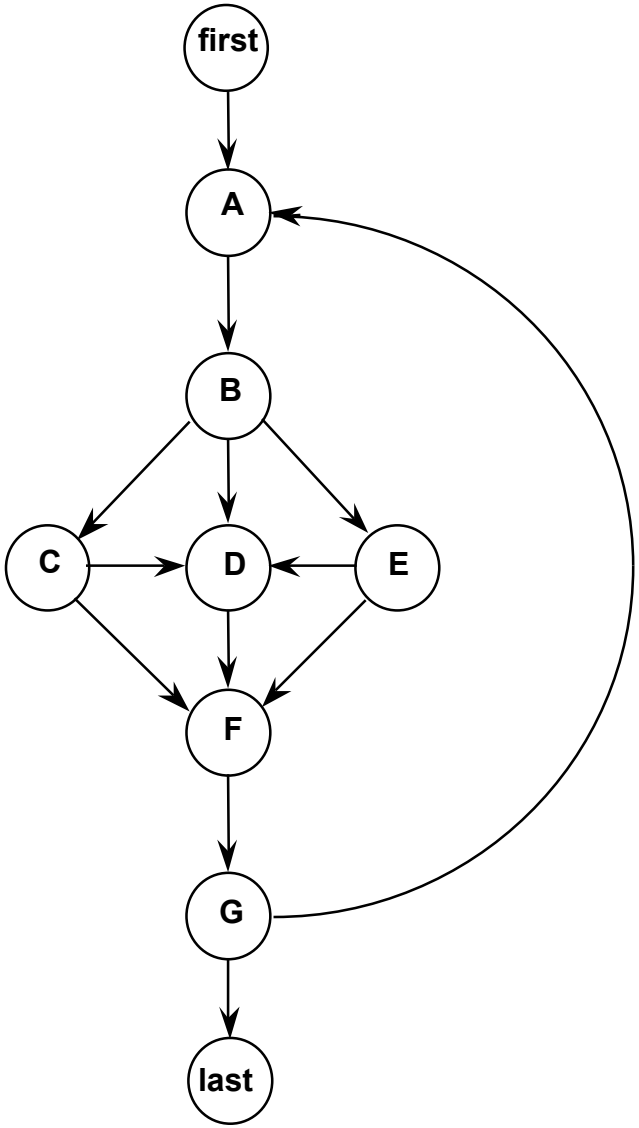
BUT

- **What who would ever test all of these?**
- **We will have an elegant, mathematically sensible alternative**

Stephen R. Schach, *Software Engineering*, (2nd edition) Richard D. Irwin, Inc. and Aksen Associates, Inc. 1993 (also in all later editions!)



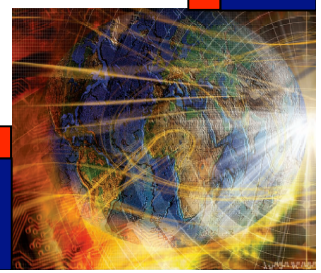
Test Cases for Schach's "Program"



1. First-A-B-C-F-G-Last
2. First-A-B-C-D-F-G-Last
3. First-A-B-D-F-G-A-B-D-F-G-Last
4. First-A-B-E-F-G-Last
5. First-A-B-E-D-F-G-Last

These test cases cover

- Every node
- Every edge
- Normal repeat of the loop
- Exiting the loop

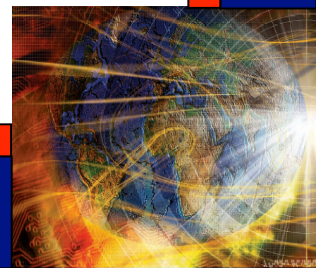


Program Graph Definitions

Given a program written in an imperative programming language, its *program graph* is a directed graph in which…

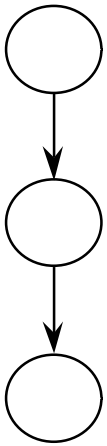
(Traditional Definition) nodes are program statements, and edges represent flow of control (there is an edge from node i to node j iff the statement corresponding to node j can be executed immediately after the statement corresponding to node i).

(improved definition) nodes are either entire statements or fragments of a statement, and edges represent flow of control (there is an edge from node i to node j iff the statement (fragment) corresponding to node j can be executed immediately after the statement or statement fragment corresponding to node i).

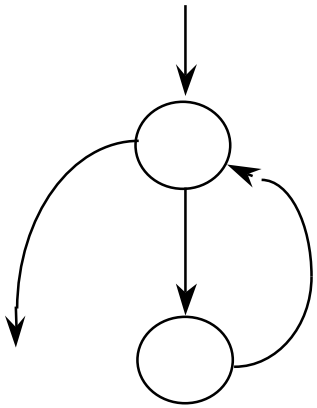


Program Graphs of Structured Programming Constructs

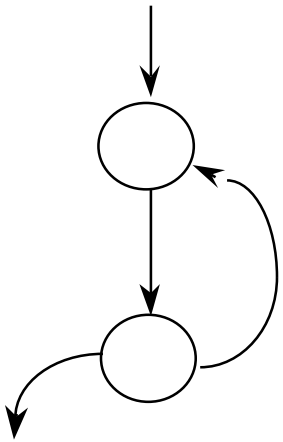
Sequence



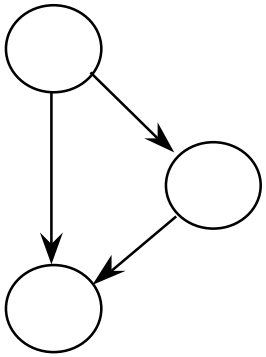
Pre-test Loop



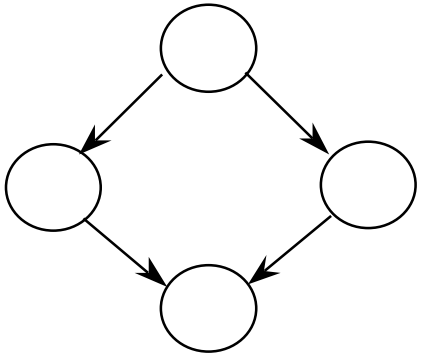
Post-test Loop



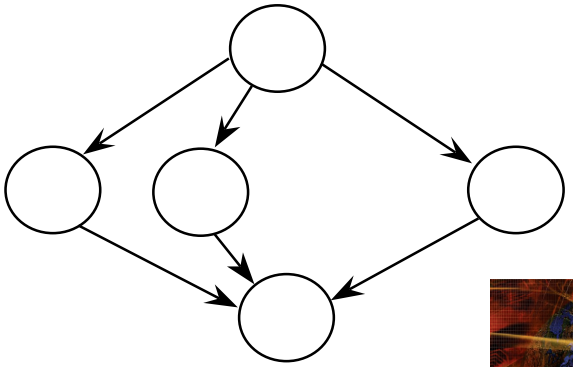
If-Then



If-Then-Else

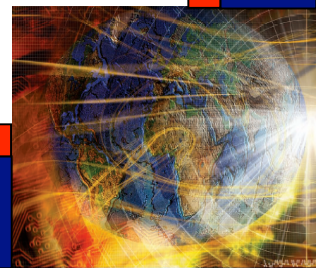


Case



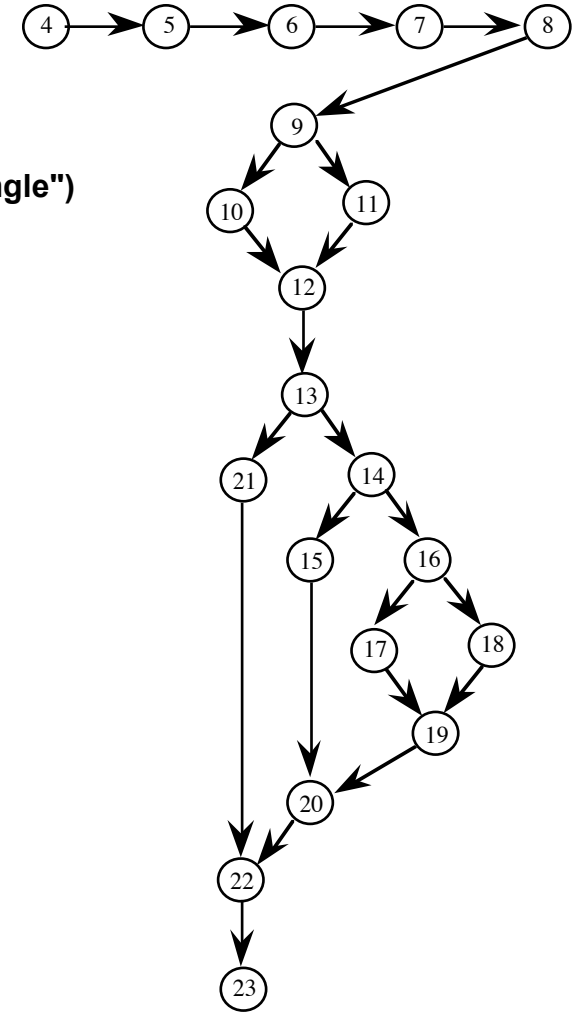
Triangle Program Specification

- Inputs: a, b, and c are non-negative integers, taken to be sides of a triangle
- Output: type of triangle formed by a, b, and c
  - Not a triangle
  - Scalene (no equal sides)
  - Isosceles (exactly 2 sides equal)
  - Equilateral (3 sides equal)
- To be a triangle, a, b, and c must satisfy the triangle inequalities:
  - $a < b + c$,
  - $b < a + c$, and
  - $c < a + b$



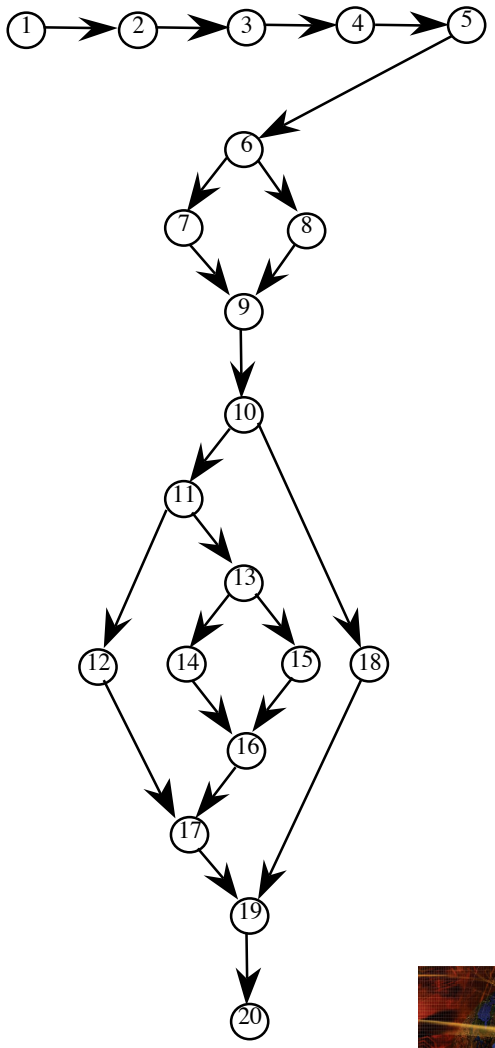
Sample Program Graph

```
1.  Program triangle
2.  Dim a,b,c As Integer
3.  Dim IsATriangle As Boolean
    'Step 1:  Get Input
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
9.  If (a < b + c) AND (b < a + c) AND (c < a + b)
10.    Then IsATriangle = True
11.    Else IsATriangle = False
12  EndIf
    'Step 3:  Determine Triangle Type
13.  If IsATriangle
14.    Then   If (a = b) AND (b = c)
15.               Then Output ("Equilateral")
16.               Else   If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
17.                          Then   Output ("Scalene")
18.                          Else   Output ("Isosceles")
19.                      EndIf
20.           EndIf
21.    Else   Output("Not a Triangle")
22.  EndIf
23.  End triangle2
```



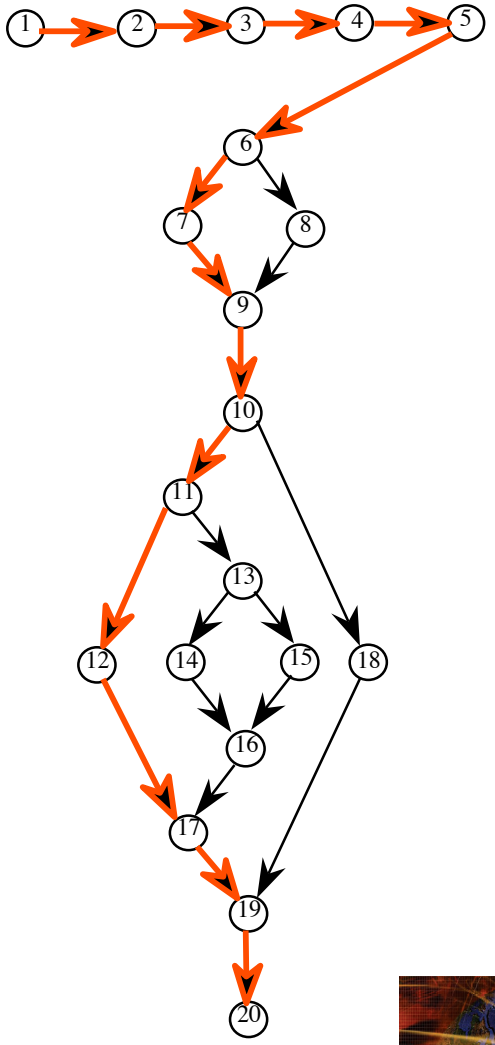
Trace Code for a = 5, b = 5, c = 5

```
Program Triangle
Dim a, b, c As Integer
Dim IsATriangle As Boolean
'Step 1:  Get Input
1.   Output ("Enter 3 integers which are sides
      of a triangle")
2.   Input (a,b,c)
3.   Output ("Side A is ",a)
4.   Output ("Side B is ",b)
5.   Output ("Side C is ",c)
'Step 2:  Is A Triangle?
6.   If (a < b + c) AND (b < a + c) AND (c < a +
      b)
7.    Then IsATriangle = True
8.    Else IsATriangle = False
9.   Endif
'Step 3:  Determine Triangle Type
10. If IsATriangle
11.   Then If (a = b) AND (b = c)
12.                    Then Output
      ("Equilateral")
13.                     Else If (a ≠ b) AND (a ≠
      c) AND (b ≠ c)
14.                                          Then
      Output ("Scalene")
15.                                           Else
      Output ("Isosceles")
16.                                  Endif
17.             Endif
18.   Else Output ("Not a Triangle")
19.  Endif
```



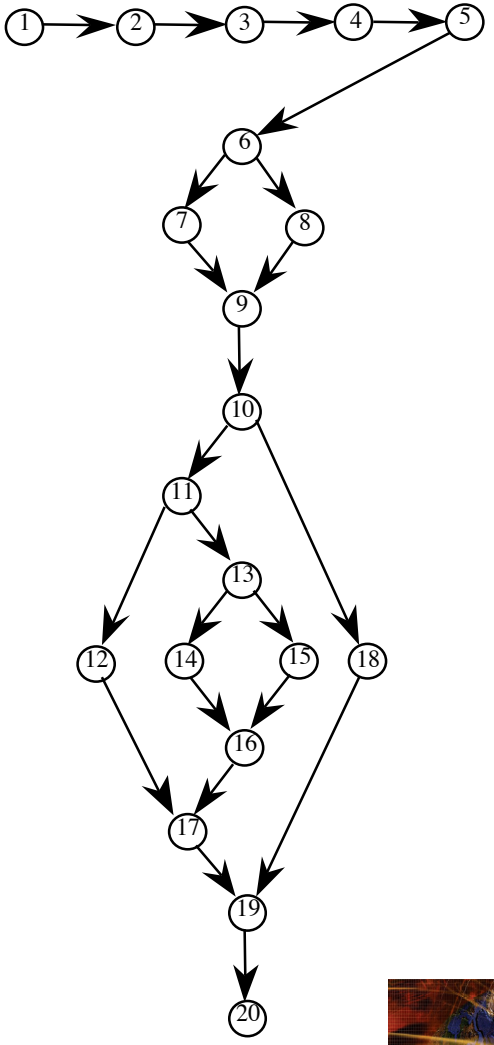
Trace Code for a = 5, b = 5, c = 5

```
Program Triangle
Dim a, b, c As Integer
Dim IsATriangle As Boolean
'Step 1:  Get Input
1.   Output ("Enter 3 integers which are sides
      of a triangle")
2.   Input (a,b,c)
3.   Output ("Side A is ",a)
4.   Output ("Side B is ",b)
5.   Output ("Side C is ",c)
'Step 2:  Is A Triangle?
6.   If (a < b + c) AND (b < a + c) AND (c < a +
      b)
7.    Then IsATriangle = True
8.    Else IsATriangle = False
9.   Endif
'Step 3:  Determine Triangle Type
10. If IsATriangle
11.   Then If (a = b) AND (b = c)
12.                    Then Output
      ("Equilateral")
13.                     Else If (a ≠ b) AND (a ≠
      c) AND (b ≠ c)
14.                                          Then
      Output ("Scalene")
15.                                           Else
      Output ("Isosceles")
16.                                    Endif
17.                Endif
18.   Else Output ("Not a triangle")
19.   Endif
```



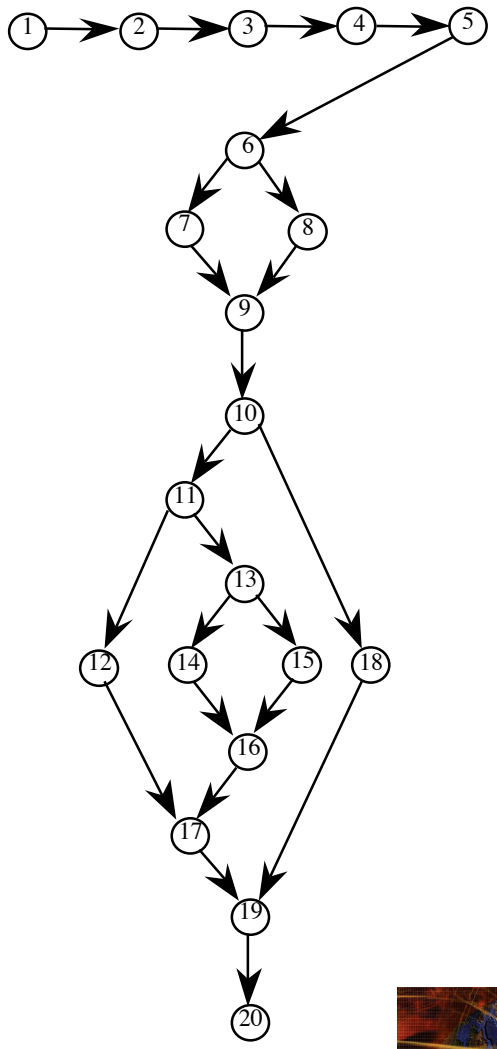
Trace Code for a = 2, b = 5, c = 5

```
Program Triangle
Dim a, b, c As Integer
Dim IsATriangle As Boolean
'Step 1:  Get Input
1.   Output ("Enter 3 integers which are sides
      of a triangle")
2.   Input (a,b,c)
3.   Output ("Side A is ",a)
4.   Output ("Side B is ",b)
5.   Output ("Side C is ",c)
'Step 2:  Is A Triangle?
6.   If (a < b + c) AND (b < a + c) AND (c < a
      + b)
7.    Then IsATriangle = True
8.    Else IsATriangle = False
9.   Endif
'Step 3:  Determine Triangle Type
10. If IsATriangle
11.   Then If (a = b) AND (b = c)
12.                    Then Output
      ("Equilateral")
13.                     Else If (a ≠ b) AND (a ≠
      c) AND (b ≠ c)
14.                                          Then
      Output ("Scalene")
15.                                           Else
      Output ("Isosceles")
16.                                    Endif
17.               Endif
18.   Else Output ("Not a Triangle")
19.  Endif
```



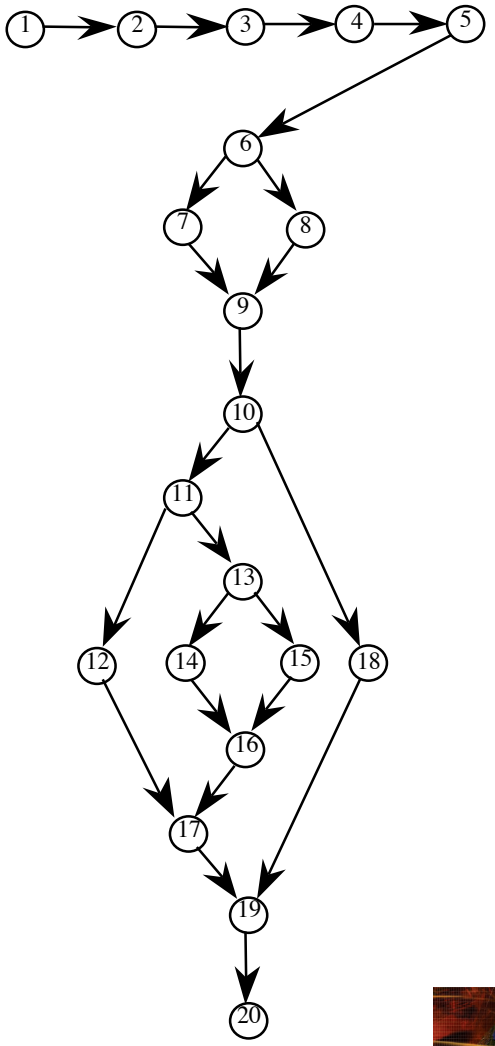
Trace Code for a = 3, b = 4, c = 5

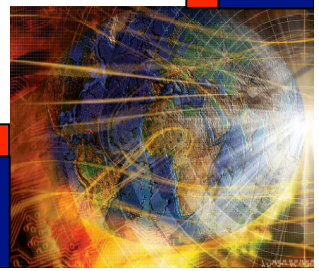
```
Program triangle
Dim a, b, c As Integer
Dim IsATriangle As Boolean
'Step 1:  Get Input
1.   Output ("Enter 3 integers which are sides
      of a triangle")
2.   Input (a,b,c)
3.   Output ("Side A is ",a)
4.   Output ("Side B is ",b)
5.   Output ("Side C is ",c)
'Step 2:  Is A Triangle?
6.   If (a < b + c) AND (b < a + c) AND (c < a
      + b)
7.    Then IsATriangle = True
8.    Else IsATriangle = False
9.   Endif
'Step 3:  Determine Triangle Type
10. If IsATriangle
11.   Then IF (a = b) AND (b = c)
12.                    Then Output
      (“Equilateral”)
13.                     Else If (a ≠ b) AND (a ≠
      c) AND (b ≠ c)
14.                                          Then
      Output (“Scalene”)
15.                                           Else
      Output (“Isosceles”)
16.                                  Endif
17.               Endif
18.   Else Output (“Not a Triangle”)
19.  Endif
```



Trace Code for a = 2, b = 3, c = 7

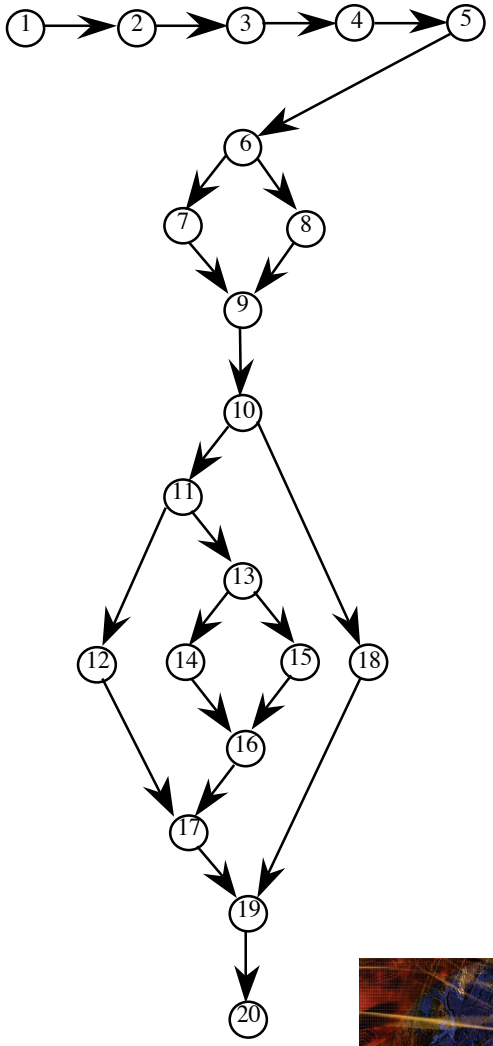
```
Program triangle
Dim a, b, c As Integer
Dim IsATriangle As Boolean
'Step 1:  Get Input
1.   Output ("Enter 3 integers which are sides
      of a triangle")
2.   Input (a,b,c)
3.   Output ("Side A is ",a)
4.   Output ("Side B is ",b)
5.   Output ("Side C is ",c)
'Step 2:  Is A Triangle?
6.   If (a < b + c) AND (b < a + c) AND (c < a
      + b)
7.    Then IsATriangle = True
8.    Else IsATriangle = False
9.   Endif
'Step 3:  Determine Triangle Type
10. If IsATriangle
11.   Then IF (a = b) AND (b = c)
12.                    Then Output
      ("Equilateral")
13.                     Else If (a ≠ b) AND (a ≠
      c) AND (b ≠ c)
14.                                          Then
      Output ("Scalene")
15.                                           Else
      Output ("Isosceles")
16.                                    Endif
17.                Endif
18.   Else Output ("Not a Triangle")
19.   Endif
```





Can you find values of a, b, and c such that the path traverses nodes 7 and 18?

```
Program triangle
Dim a, b, c As Integer
Dim IsATriangle As Boolean
'Step 1:  Get Input
1.   Output ("Enter 3 integers which are sides
      of a triangle")
2.   Input (a,b,c)
3.   Output ("Side A is ",a)
4.   Output ("Side B is ",b)
5.   Output ("Side C is ",c)
'Step 2:  Is A Triangle?
6.   If (a < b + c) AND (b < a + c) AND (c < a +
      b)
7.    Then IsATriangle = True
8.    Else IsATriangle = False
9.   Endif
'Step 3:  Determine Triangle Type
10. If IsATriangle
11.   Then IF (a = b) AND (b = c)
12.                  Then Output
      ("Equilateral")
13.                   Else If (a ≠ b) AND (a ≠
      c) AND (b ≠ c)
14.                                        Then
      Output ("Scalene")
15.                                         Else
      Output ("Isosceles")
16.                                    Endif
17.                 Endif
18.   Else Output ("Not a Triangle")
```



DD-Paths

A DD-Path (decision-to-decision) is a chain in a program graph such that

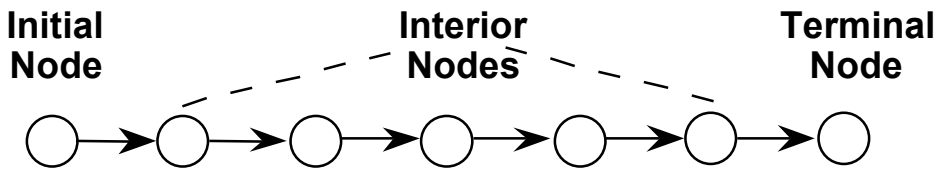
Case 1: it consists of a single node with indegree = 0, or

Case 2: it consists of a single node with outdegree = 0, or

Case 3: it consists of a single node with indegree >= 2 or outdegree >= 2, or

Case 4: it consists of a single node with indegree = 1 and outdegree = 1, or

Case 5: it is a maximal chain of length >= 1.

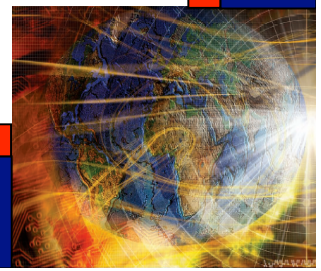


a 2-connected chain

DD-Path Graph

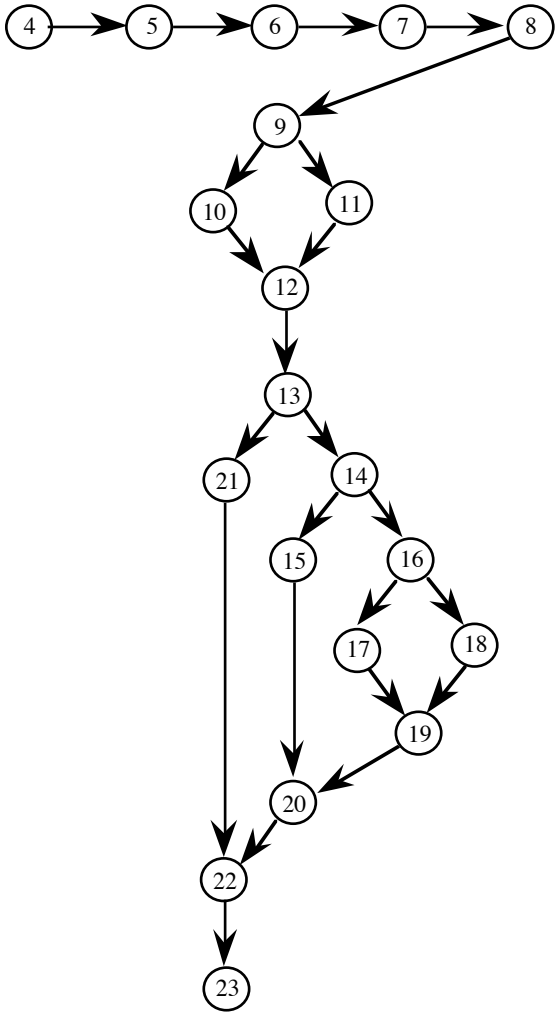
Given a program written in an imperative language, its *DD-Path graph* is the directed graph in which nodes are DD-Paths of its program graph, and edges represent control flow between successor DD-Paths.

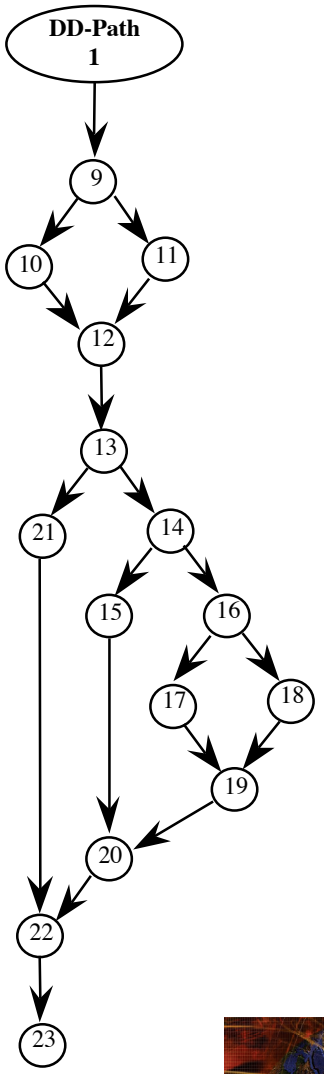
- **a form of condensation graph**
- **2-connected components are collapsed into an individual node**
- **single node DD-Paths (corresponding to Cases 1 - 4) preserve the convention that a statement fragment is in exactly one DD-Path**

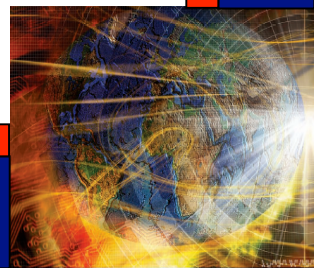


DD-Path Graph

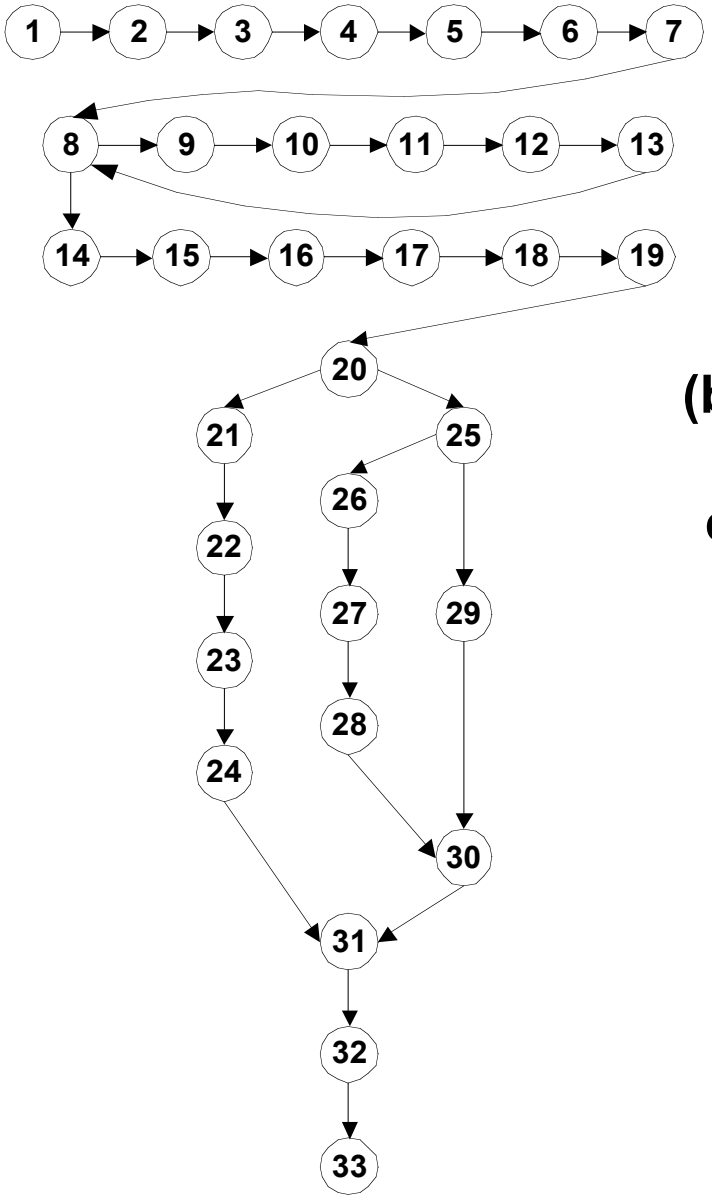
(not much compression because this example is control intensive, with little sequential code.)



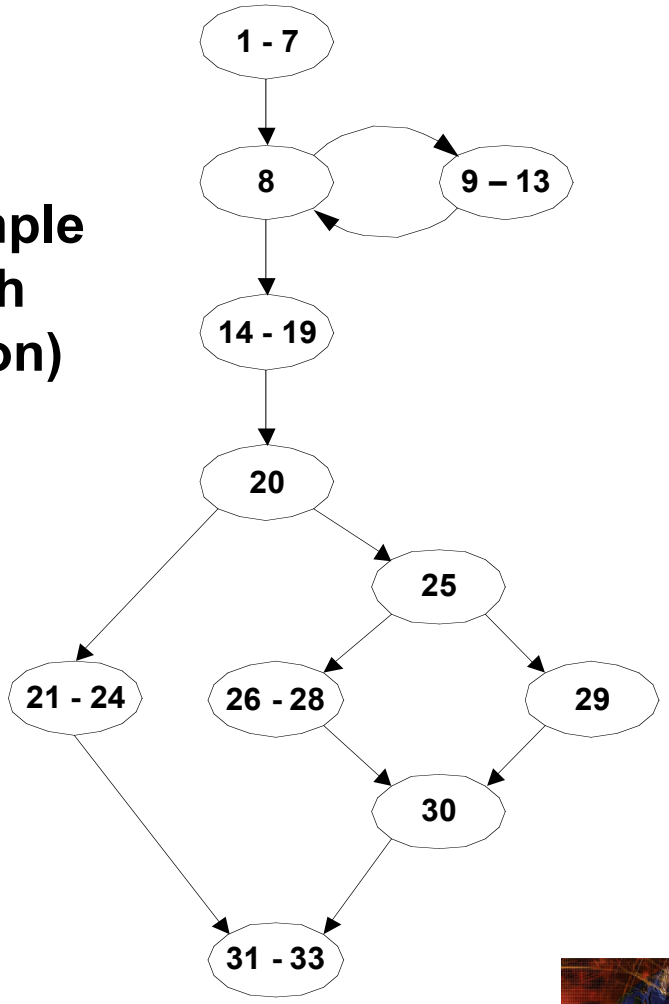




DD-Path Graph of Commission Problem



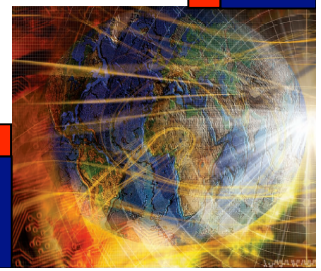
(better example of DD-Path compression)



Structural Test Coverage Metrics

(E. F. Miller, 1977 dissertation)

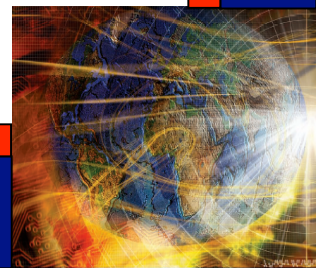
- C_0: Every statement
- C_1: Every DD-Path
- C_{1p}: Every predicate outcome
- C_2: C_1 coverage + loop coverage
- C_d: C_1 coverage +every pair of dependent DD-Paths
- C_{MCC}: Multiple condition coverage
- C_{ik}: Every program path that contains up to k repetitions of a loop (usually k = 2)
- C_{stat}: "Statistically significant" fraction of paths
- C_∞: All possible execution paths



Graph-Based Coverage Metrics

1. Every node
2. Every edge
3. Successive pairs of edges
4. Every path

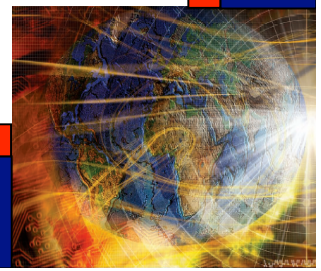
Exercise: How do these compare with structural test coverage metrics?



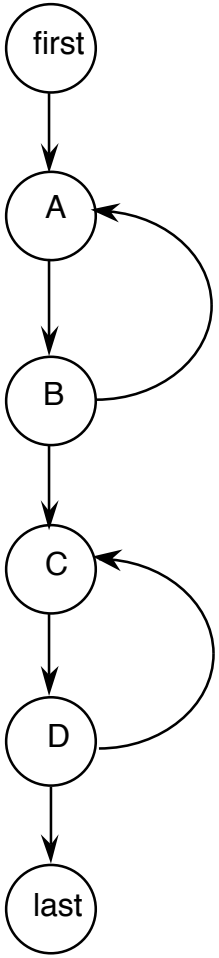
Testing Loops

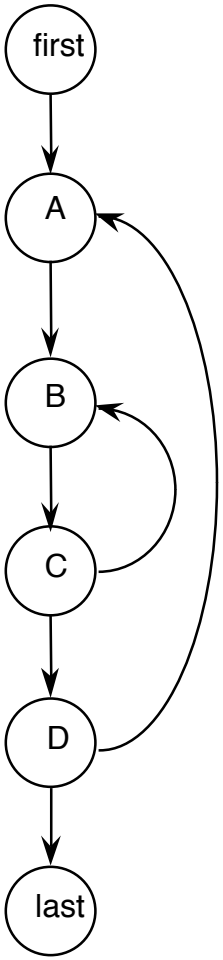
Huang's Theorem: (Paraphrased)
Everything interesting will happen in two loop traversals: the normal loop traversal and the exit from the loop.

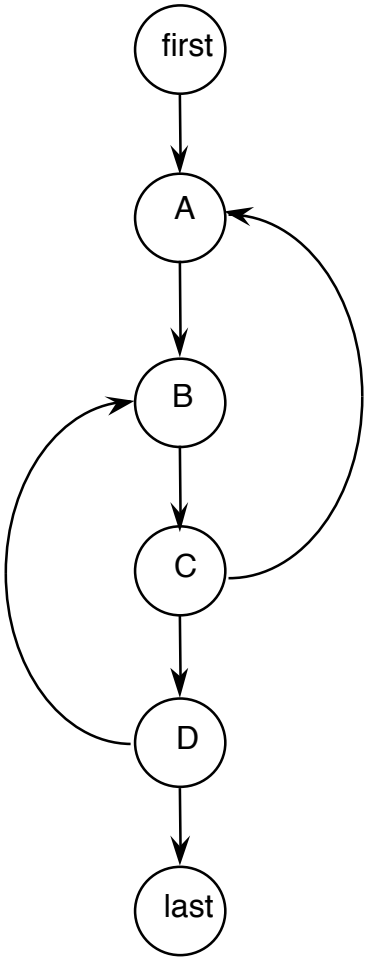
Exercise:
Discuss Huang's Theorem in terms of graph based coverage metrics.

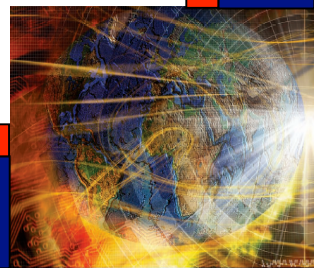


Concatenated, Nested, and Knotted Loops



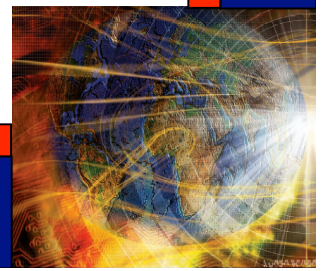






Strategy for Loop Testing

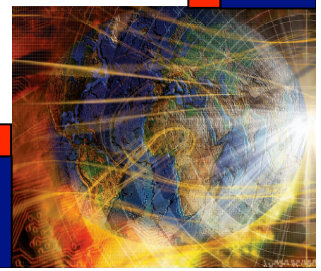
- **Huang's theorem suggests/assures 2 tests per loop is sufficient.** (Judgment required, based on reality of the code.)
- **For nested loops:**
  - Test innermost loop first
  - Then "condense" the loop into a single node (as in condensation graph, see Chapter 4)
  - Work from innermost to outermost loop
- **For concatenated loops: use Huang's Theorem**
- **For knotted loops: Rewrite!** (see McCabe's cyclomatic complexity)



Multiple Condition Testing

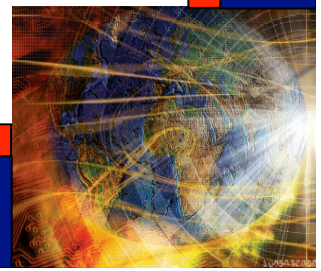
- Consider the multiple condition as a logical proposition, i.e., some logical expression of simple conditions.
- Make the truth table of the logical expression.
- Convert the truth table to a decision table.
- Develop test cases for each rule of the decision table (except the impossible rules, if any).
- Next 3 slides: multiple condition testing for

```
If (a < b + c) AND (b < a + c) AND (c < a + b)
    Then IsATriangle = True
    Else  IsATriangle = False
Endif
```



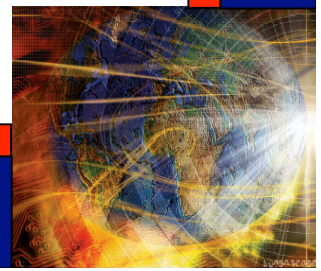
Truth Table for
(a<b+c) AND (b<a+c) AND (c<a+b)

(a<b+c)	(b<a+c)	(c<a+b)	(a<b+c) AND (b<a+c) AND (c<a+b)
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F



Decision Table for
(a<b+c) AND (b<a+c) AND (c<a+b)

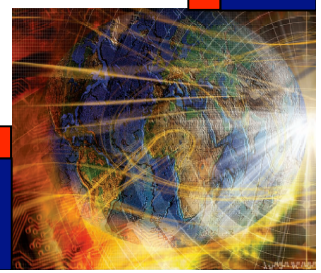
c1: a<b+c	T	T	T	T	F	F	F	F
c2: b<a+c	T	T	F	F	T	T	F	F
c3: c<a+b	T	F	T	F	T	F	T	F
a1: impossible				X		X	X	X
a2:Valid test case #	1	2	3		4			



Multiple Condition Test Cases for
(a<b+c) AND (b<a+c) AND (c<a+b)

Test Case		a	b	c	expected output
1	all true	3	4	5	TRUE
2	c ≥ a + b	3	4	9	FALSE
3	b ≥ a + c	3	9	4	FALSE
4	a ≥ b + c	9	3	4	FALSE

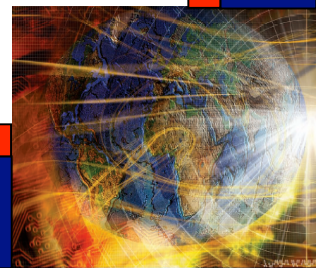
Note: could add test cases for c = a + b, b = a + c, and a = b + c.



Exercise: What test cases are needed for this code fragment ?

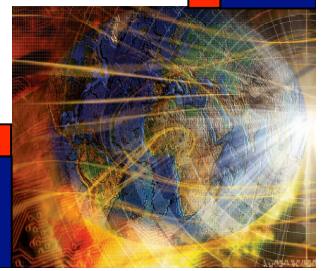
```
13. If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
14.       Then Output (“Scalene”)
15.       Else Output (“Isosceles”)
16. Endif
```

a	b	c	Expected Output



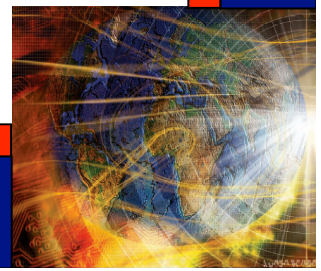
Dependent DD-Paths
(often correspond to infeasible paths)

- Look at the Triangle Program code in slide 8
- And the program graph and DD-Path graph in slide 11
- If a path traverses node 10 (Then IsATriangle = True), then it must traverse node 14.
- Similarly, if a path traverses node 11 (Else IsATriangle = False), then it must traverse node 21.
- Paths through nodes 10 and 21 are infeasible.
- Similarly for paths through 11 and 14.
- Hence the need for the C_d coverage metric.



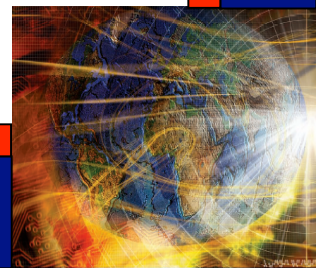
Code-Based Testing Strategy

- Start with a set of test cases generated by an "appropriate" (depends on the nature of the program) specification-based test method.
- Look at code to determine appropriate test coverage metric.
  - Loops?
  - Compound conditions?
  - Dependencies?
- If appropriate coverage is attained, fine.
- Otherwise, add test cases to attain the intended test coverage.



Test Coverage Tools

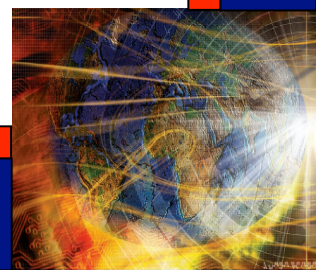
- Commercial test coverage tools use "instrumented" source code.
  - New code added to the code being tested
  - Designed to "observe" a level of test coverage
- When a set of test cases is run on the instrumented code, the designed test coverage is ascertained.
- Strictly speaking, running test cases in instrumented code is not sufficient
  - Safety critical applications require tests to be run on actual (delivered, non-instrumented) code.
  - Usually addressed by mandated testing standards.



Sample DD-Path Instrumentation

(values of array DDpathTraversed are set to 1 when corresponding instrumented code is executed.)

```
    DDpathTraversed(1) = 1
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
    DDpathTraversed(2) = 1
9.  If (a < b + c) AND (b < a + c) AND (c < a + b)
10. Then  DDpathTraversed(3) = 1
          IsATriangle = True
11. Else  DDpathTraversed(4) = 1
          IsATriangle = False
12  EndIf
```



Instrumentation Exercise:

How could you instrument the Triangle Program to record how many times a set of test cases traverses the individual DD-Paths?