SOFTWARE TESTING

WHITE BOX TESTING

(Path & Data Flow)

Gogte Institute of Technology Belagavi

COURTESY SKM

PATH TESTING, DATA FLOW TESTING

CONCEPTS DISCUSSED

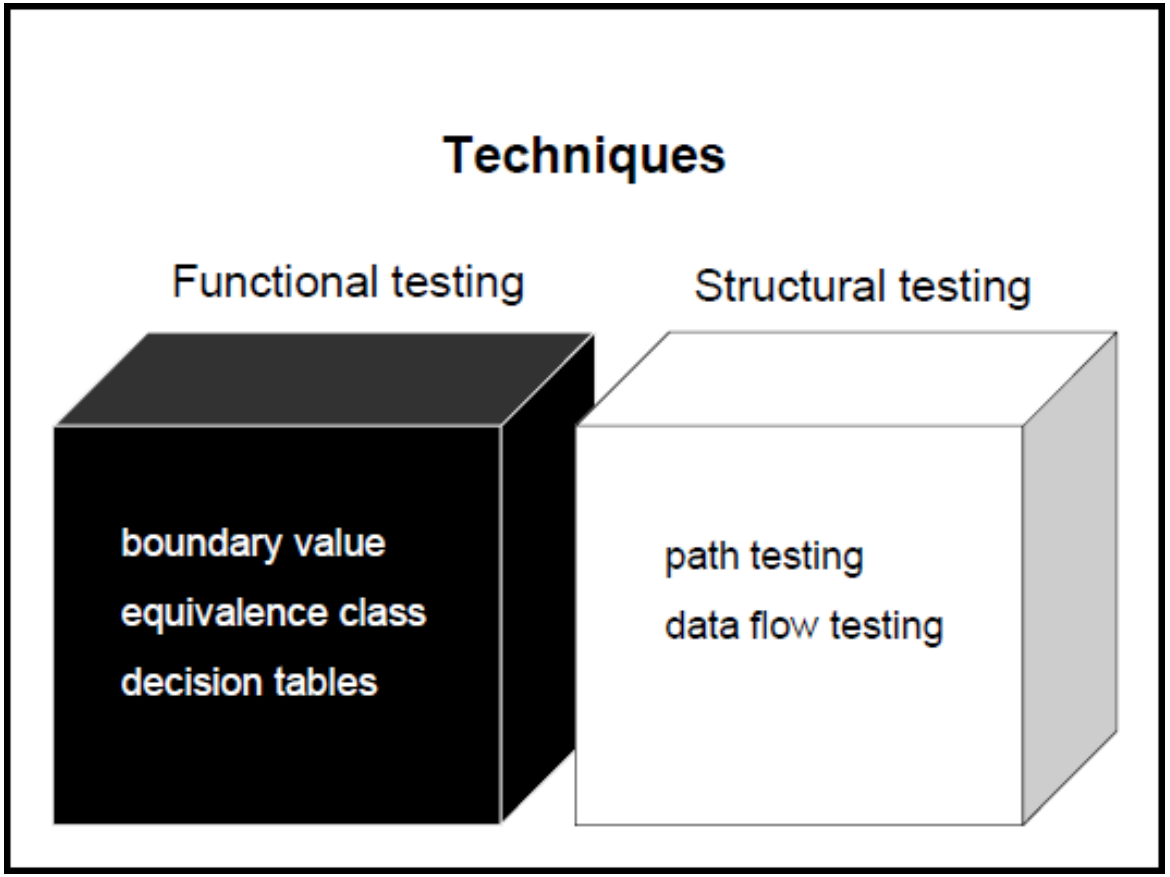
- **Path Testing**

Program Graphs, DD-Paths, Metrics, Basis Path Testing, Test Coverage

- **Data Flow Testing**

Define/Use Testing, Slice-Based Testing, Program Slicing Tools

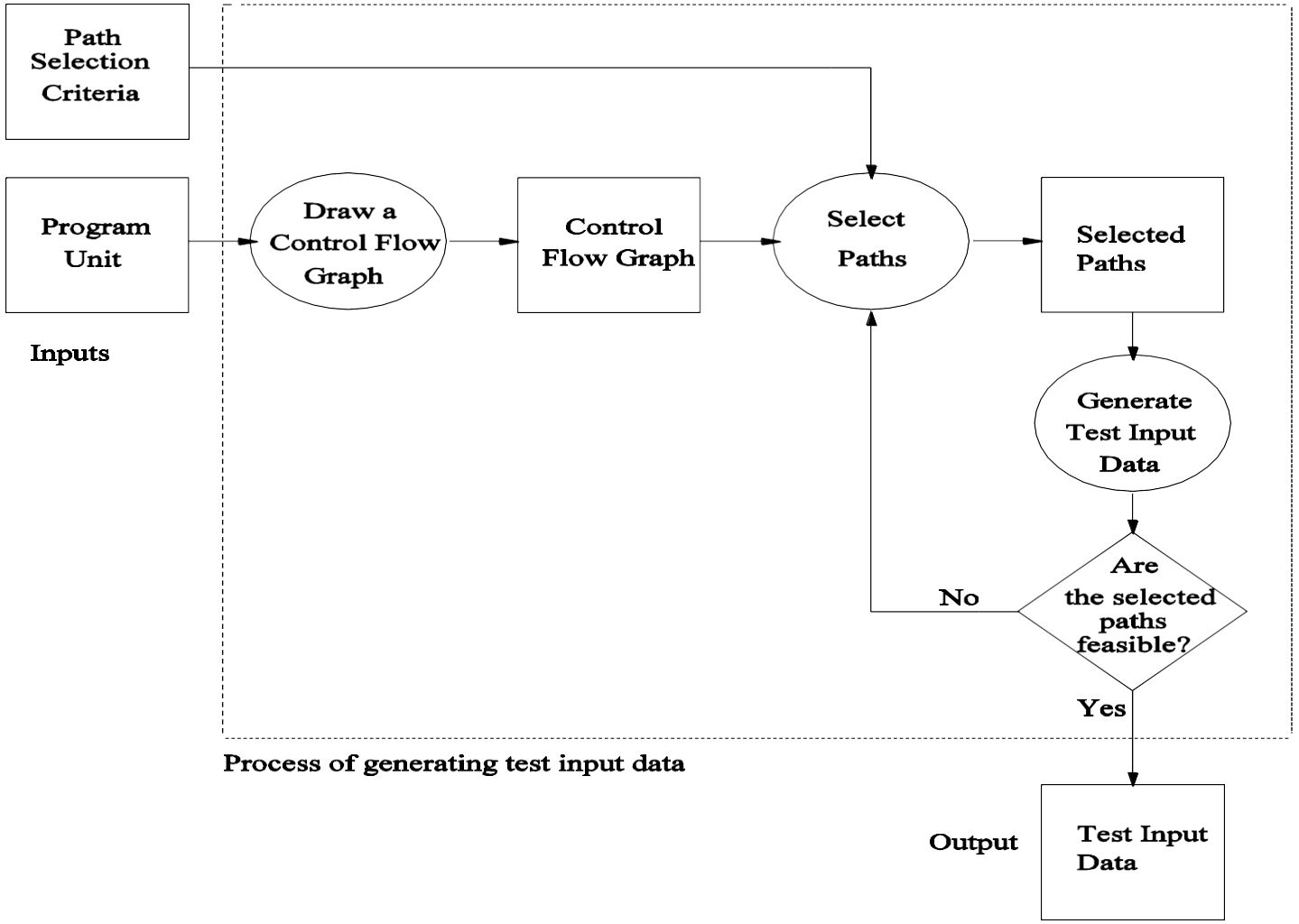
STRUCTURAL TESTING



BASIC IDEA

- **Kinds of program statements:**
  - Assignment statements (Ex. x = 2*y;)
  - Conditional statements (Ex. if(), for(), while(), …)
- **Control Flow**
  - Successive execution of program statements is viewed as flow of control.
  - Conditional statements alter the default flow.
- **Program Path**
  - A program path is a sequence of statements from entry to exit.
  - There can be a large number of paths in a program.
  - There is an (input, expected output) pair for each path.
  - Executing a path requires invoking the program unit with the right test input.
  - Paths are chosen by using the concepts of path selection criteria.

OUTLINE OF CONTROL FLOW TESTING



The process of generating test input data for control flow testing

PATH TESTING

- Structural testing technique or White-box technique

- **Definition:**

Path testing is a structural testing method that involves using the source code of a program to attempt to find every possible executable path

- **Why Path Testing?**
  - The idea is that we are then able to test each individual path in as many ways as possible in order to **maximize the coverage of each test case**

- **What it does?**
  - This gives the best possible chance of **discovering all faults within a piece of code**

PATH TESTING TECHNIQUES

- **Control Flow Graph (CFG) -** The Program is converted into Flow graphs by representing the code into nodes, regions and edges.
- **Decision to Decision path (D-D) -** The CFG can be broken into various Decision to Decision paths and then collapsed into individual nodes.
- **Independent (basis) paths -** Independent path is a path through a DD-path graph which cannot be reproduced from other paths by other methods.

STEPS INVOVLED IN PATH TESTING

- Generate flow graph of a program
- Generate DD path graph
- Identify independent paths

Outline of Control Flow Testing

- **Inputs to the test generation process**
  - Source code
  - Path selection criteria: statement, branch, …
- **Generation of control flow graph (CFG)**
  - A CFG is a graphical representation of a program unit.
  - Compilers are modified to produce CFGs. (You can draw one by hand.)
- **Selection of paths**
  - Enough entry/exit paths are selected to satisfy path selection criteria.
- **Generation of test input data**
  - Two kinds of paths
    - Executable path: There exists input so that the path is executed.
    - Infeasible path: There is no input to execute the path.
  - Solve the path conditions to produce test input for each path.

Path Selection Criteria

- Program paths are selectively executed.
- Question: What paths do I select for testing?
- The concept of *path selection criteria* is used to answer the question.
- Advantages of selecting paths based on defined criteria:
  - Ensure that all program constructs are executed at least once.
  - Repeated selection of the same path is avoided.
  - One can easily identify what features have been tested and what not.
- Path selection criteria
  - Select all paths.
  - Select paths to achieve complete statement coverage.
  - Select paths to achieve complete branch coverage.
  - Select paths to achieve predicate coverage.

PROGRAM GRAPH

- **PROGRAM GRAPH**

A **directed graph** in which **nodes** are either:

-entire statements or fragments of statement,

- edges represent flow of control

- The importance of program graph is that **program executions** correspond to **paths from source to the sink node**
- Circle → Represents "NODES"
- Arrow → Represents "EDGE" or "LINK"

PROGRAM GRAPH

- **DEFINITION**

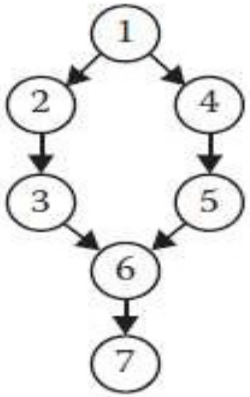
If i and j are nodes in the program graph, there is an edge from node i to node j if and only if the statement (fragment) corresponding to node j can be executed immediately after the statement (fragment) corresponding to node i

- **Program Graph is derived from program**

STYLE CHOICES FOR PROGRAM GRAPH

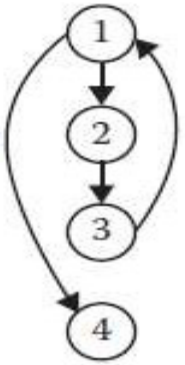
If–Then–Else

```
1  If <condition>
2     Then
3        <then statements>
4     Else
5        <else statements>
6  End If
7  <next statement>
```



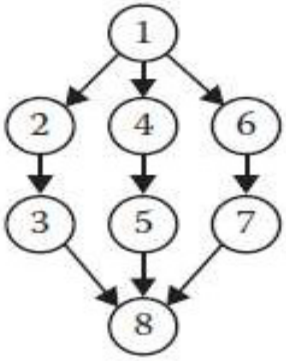
Pretest loop

```
1  While <condition>
2     <repeated body>
3  End While
4  <next statement>
```



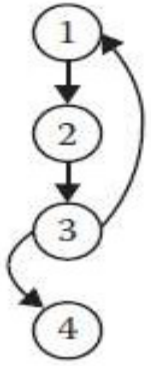
Case/Switch

```
1  Case n of 3
2     n=1:
3        <case 1  statements>
4     n=2:
5        <case 2  statements>
6     n=3:
7        <case 3  statements>
8  End Case
```



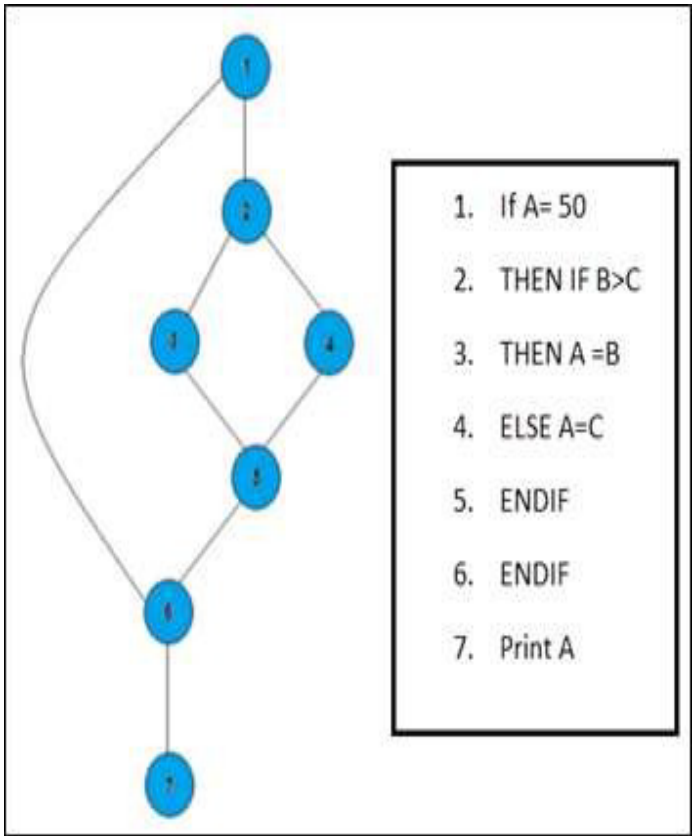
Posttest loop

```
1  Do
2     <repeated body>
3  Until <condition>
4  <next statement>
```



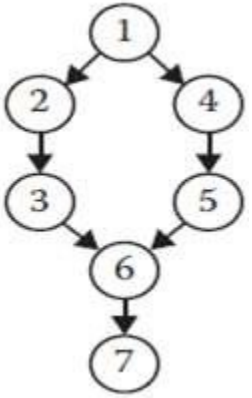
Program graphs of four structured programming constructs.

LINE NUMBER → REFERS TO THE STATEMENT & STATEMENT FRAGMENT



If–Then–Else

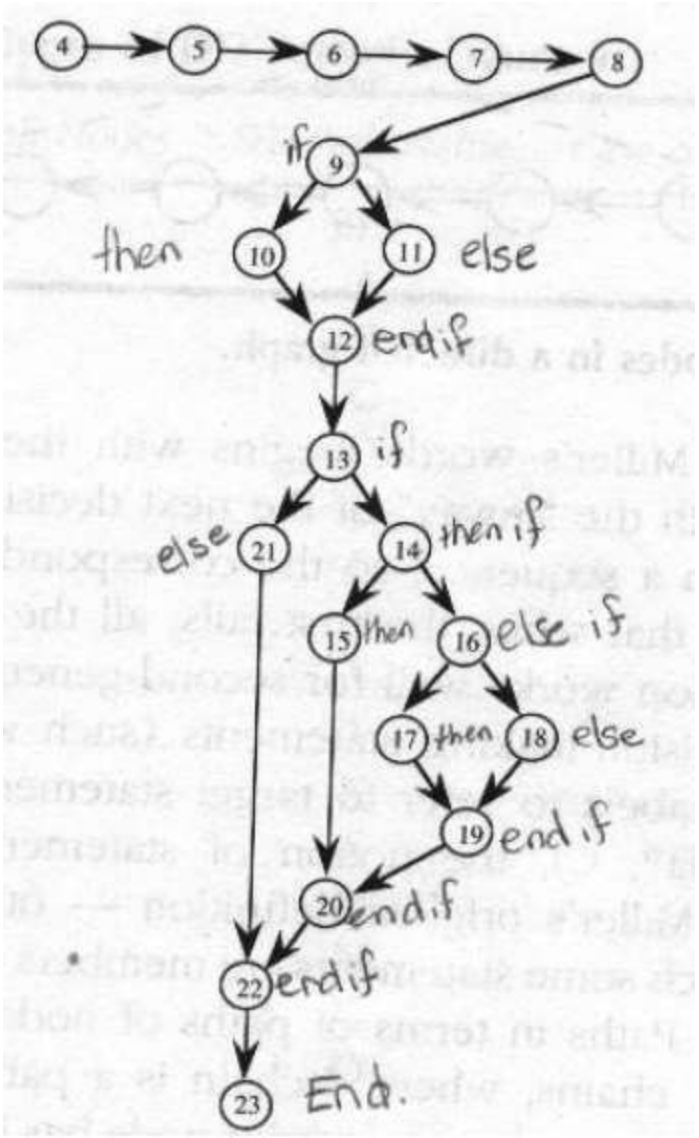
```
1  If <condition>
2     Then
3        <then statements>
4     Else
5        <else statements>
6  End If
7  <next statement>
```



Arrows → edges indicates flow of control. Circles → nodes indicates one or more actions.
Areas bounded by edges and nodes are called regions.
A predicate node is a node containing a condition.

PROGRAM GRAPH FOR TRIANGLE PROBLEM

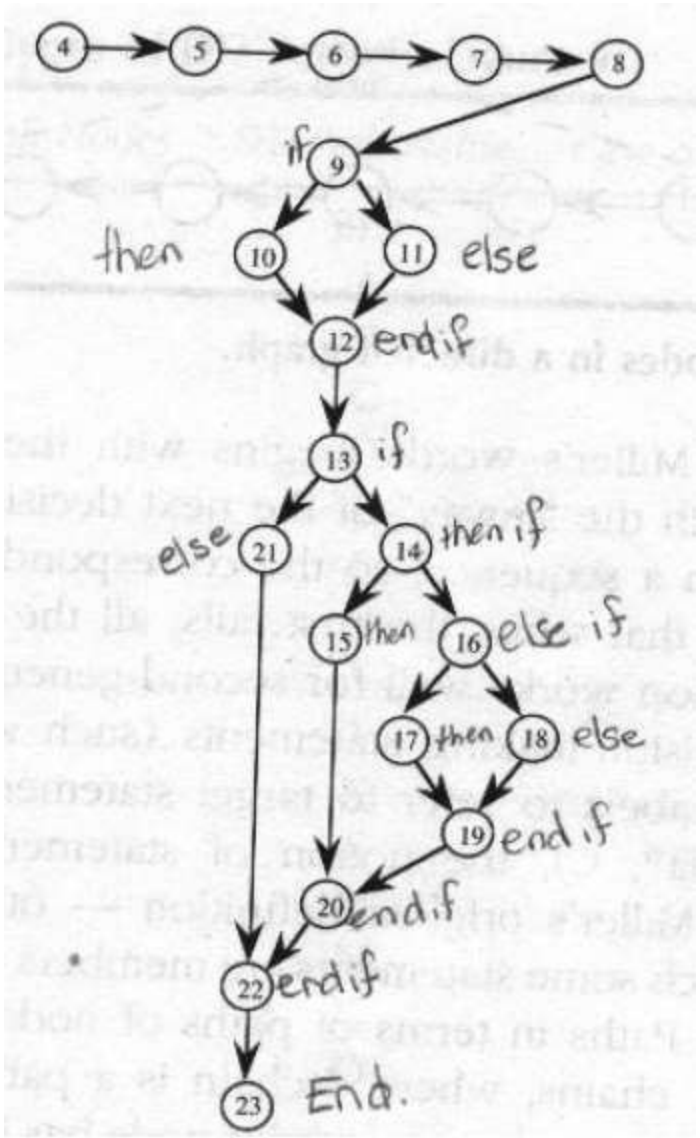
```
1.  Program triangle
2.  Dim a,b,c As Integer
3.  Dim IsATriangle As Boolean
    'Step 1:  Get Input
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
9.  If (a < 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
```



LINE NUMBER → REFERS TO THE STATEMENT & STATEMENT FRAGMENT

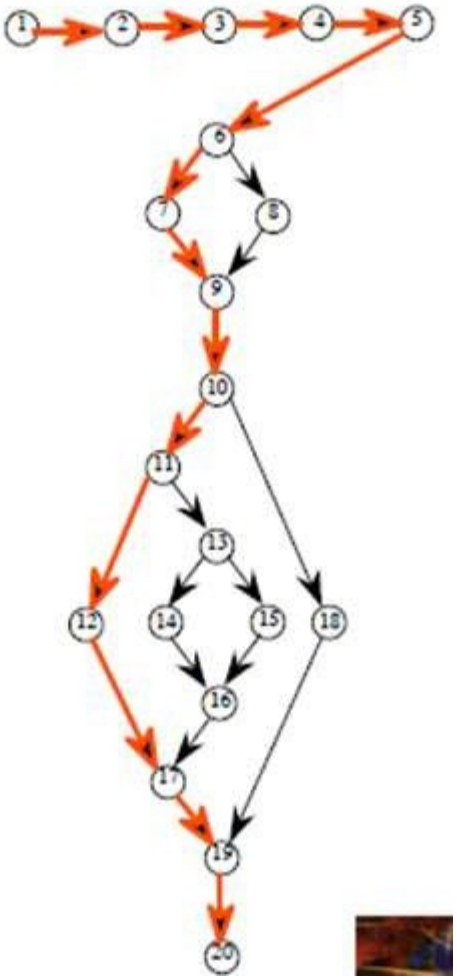
Observations

- Nodes 4-8 are a sequence, there is no branching
- Nodes 9-12 are an IF-THEN-ELSE construct
- Nodes 13-22 are nested IF-THEN-ELSE constructs
- Nodes 4 and 23 are the program source and sink nodes
  - Single entry, single exit
- There are no loops, so this is a directed acyclic graph



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
```

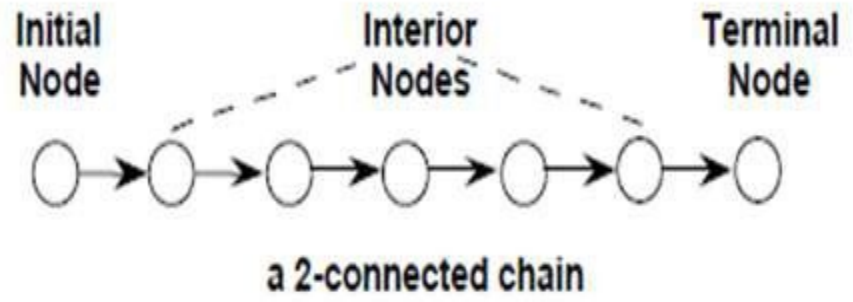


DD PATH GRAPH

SAMPLE QUESTIONS

- Write a structured triangle program, draw the program graph and find the DD paths, DD path graph for the triangle program.

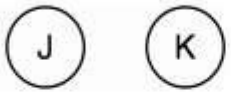
- A decision-to-decision path (DD-Path) is a path chain in a program graph such that
  - Initial and terminal nodes are distinct
  - Every interior node has indeg =1 and outdeg = 1
  - The initial node is 2-connected to every other node in the path



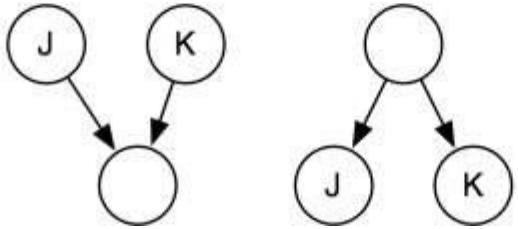
4-types of connectedness

Two nodes J and K in a directed graph are

- 0-connected iff no path exists between them

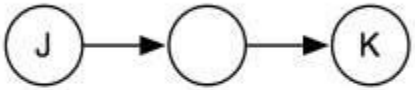


- 1-connected iff a semi-path but no path exists between them

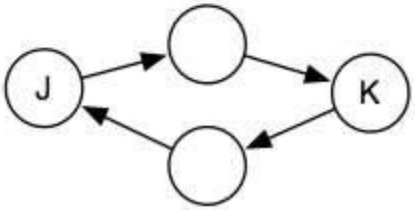


Two nodes J and K in a directed graph are

- 2-connected iff a path exists between between them



- 3-connected iff a path goes from J to K , and a path goes from K to n_1



STEPS TO BUILD A DD-PATH GRAPH

1. Number the program statements and/or fragments

2. Draw a program graph

3. Divide program graph into DD-paths
   - Identify which program graph nodes form each DD-path (according to the 5 cases in the definition of DD-paths)
   - Name each DD-path (A,B, etc.)

4. Build the DD-Paths Graph

DECISION TO DECISION PATH – DD PATH

Best-known structural testing, based on constructing a decision-to- decision path (DD-Path)

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

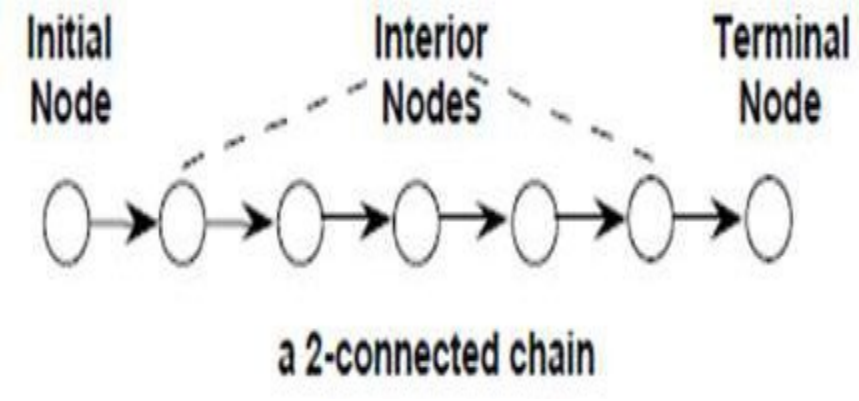
Case 3: it consists of a single node with indegree >= 2 or outdegree >= 2, or

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 DD-path is a chain in a program graph such that:

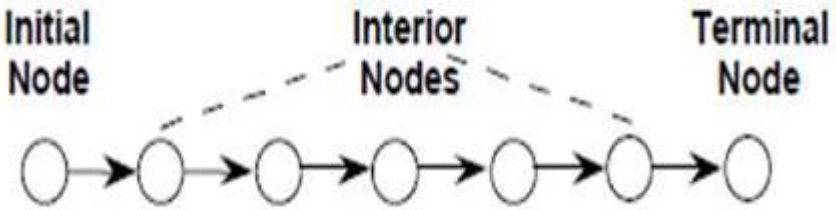
- Case 1: it consists of a single node with in-degree = 0
  - This is the *source* node (the initial DD-path)
- Case 2: it consists of a single node with out-degree = 0
  - This is the *sink* node (the final DD-path)
- Case 3: it consists of a single node with in-degree ≥ 2
  OR out-degree ≥ 2
  - Assures that no node is contained in more than 1 DD-path



- Case 4: it consists of a single node with in-degree = 1 AND out-degree = 1
  - Needed for short branches

- Case 5: it is a maximal chain of length ≥ 1
  - Normal case: single entry, single exit sequence of nodes
  - Each node is 2-connected to every other node
    - » i.e. there is a path from node n_i to n_j (& not the reverse)

A chain of nodes in a directed graph of length = **6**



The length of the chain = the number of edges

DD-paths:

- Paths of nodes in a directed graph
  - Paths = chains
  - Chain:
    - a path in which the initial & terminal nodes are distinct
    - every interior node has indegree = 1 and outdegree = 1
    - Length of chain is the number of edges

DD – PATH GRAPH

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.

The DD-Path graph is a form of condensation graph, in this condensation:

- 2-connected components are collapsed into individual nodes that correspond to case 5 DD-Paths
- Single node DD-paths (cases 1-4) are required to assure that every statement is in exactly one DD-Path

STEPS TO BUILD A DD-PATH GRAPH

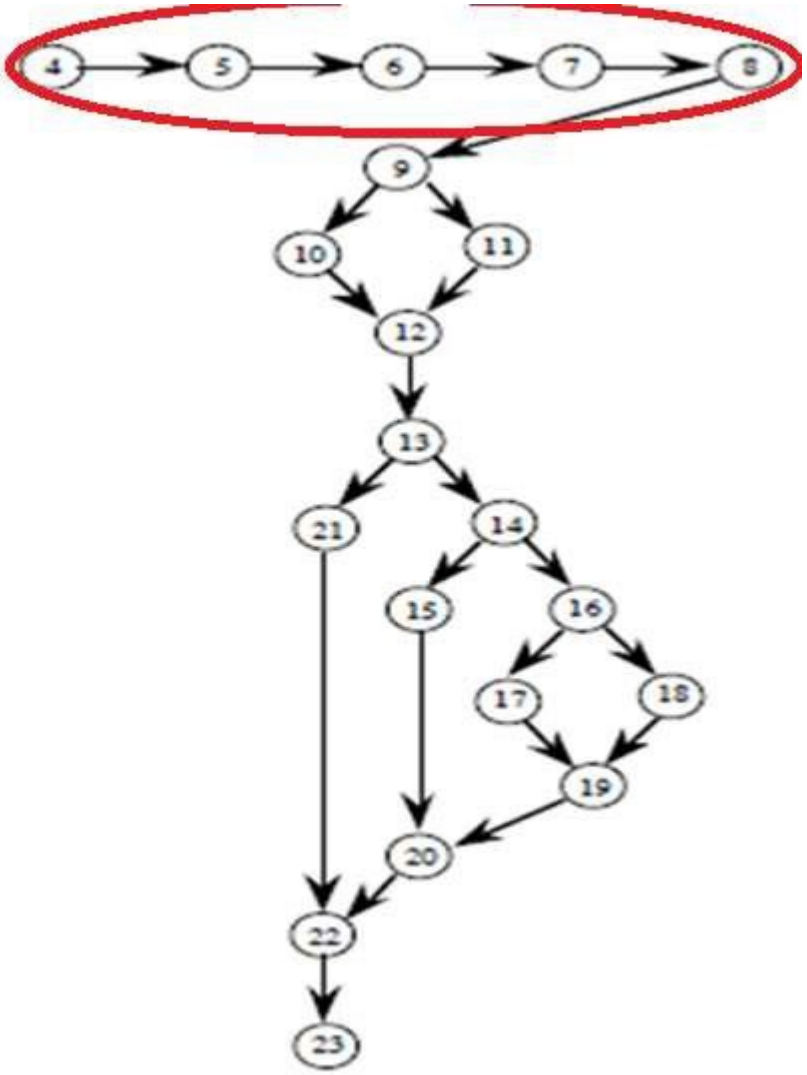
1. Number the program statements and/or fragments

2. Draw a program graph

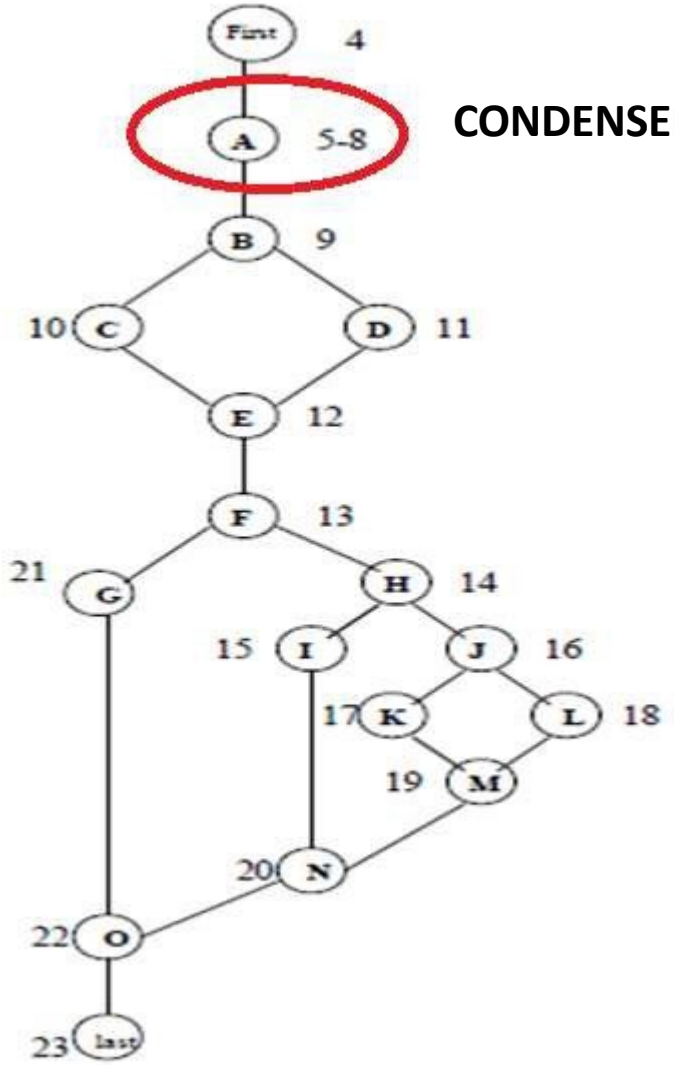
3. Divide program graph into DD-paths
   - Identify which program graph nodes form each DD-path (according to the 5 cases in the definition of DD-paths)
   - Name each DD-path (A,B, etc.)

4. Build the DD-Paths Graph

DD PATH GRAPH FOR TRIANGLE PROBLEM



PROGRAM GRAPH



CONDENSED NODE

DD-PATH GRAPH

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

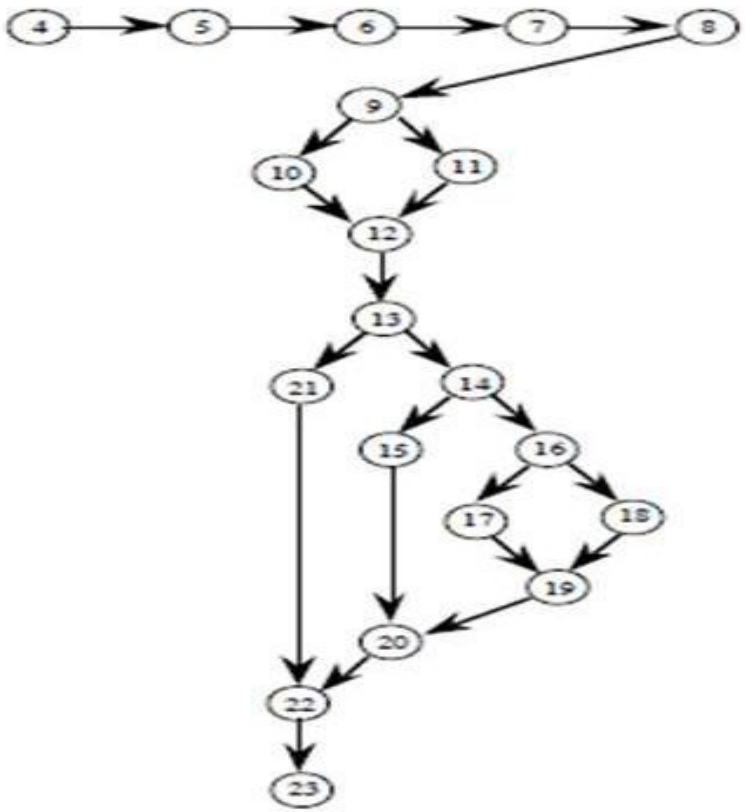
Case 1: it consists of a single node with indegree = 0, or

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

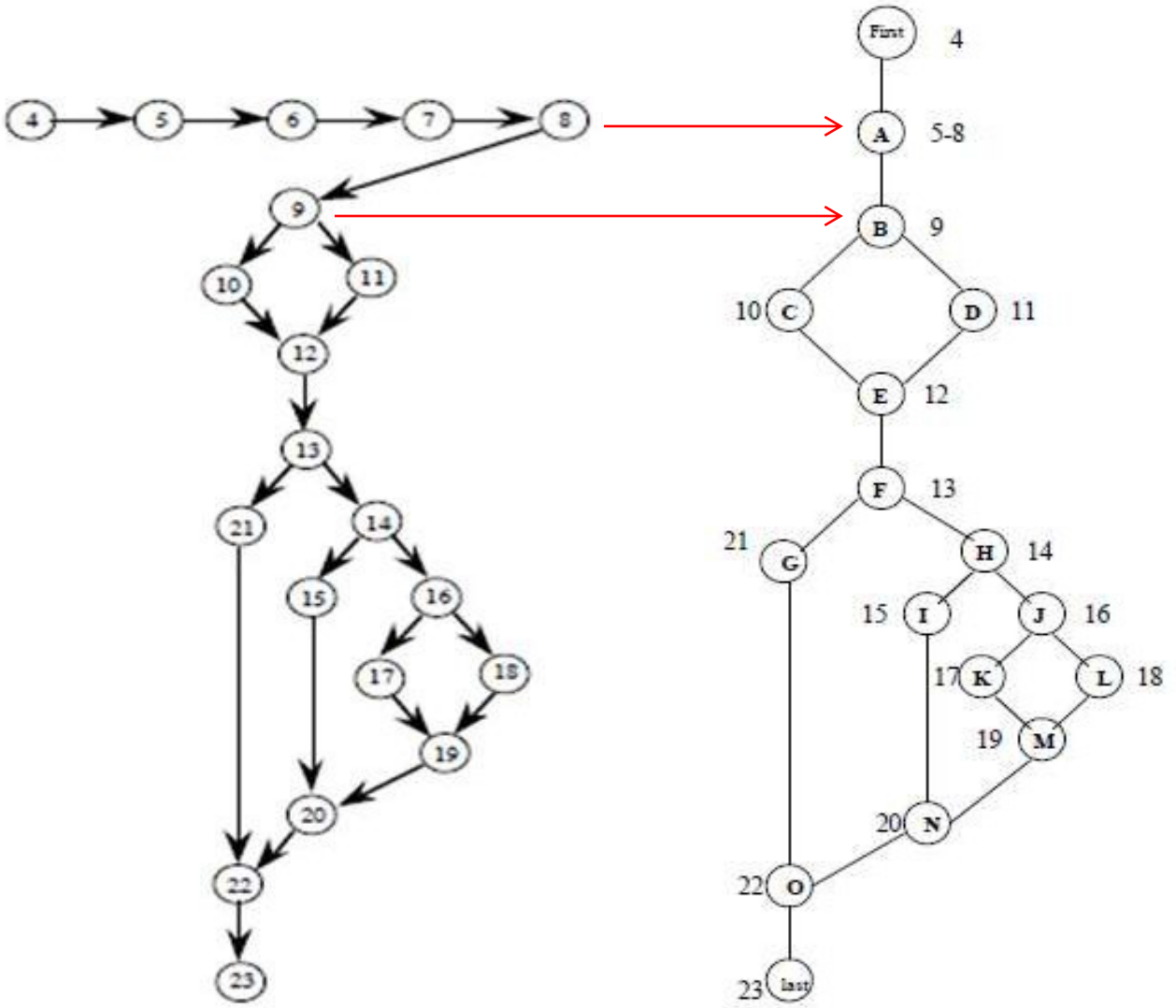
Case 5: it is a maximal chain of length >= 1.



Program Graph Nodes	DD-Path Name	Case of Definition
4	First	1
5-8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	Last	2

TYPES OF DD-PATHS IN TRIANGLE PROBLEM

Program Graph Nodes	DD-Path Name	Case of Definition
4	First	1
5-8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	Last	2



TEST COVERAGE METRICS

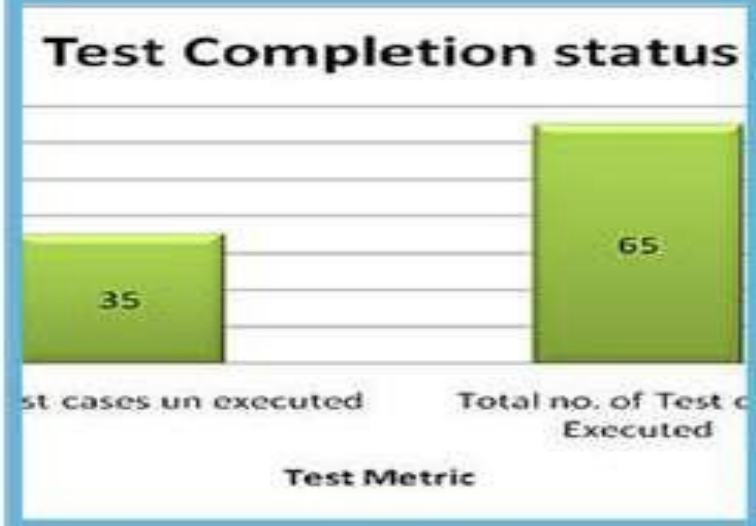
TEST COVERAGE METRICS

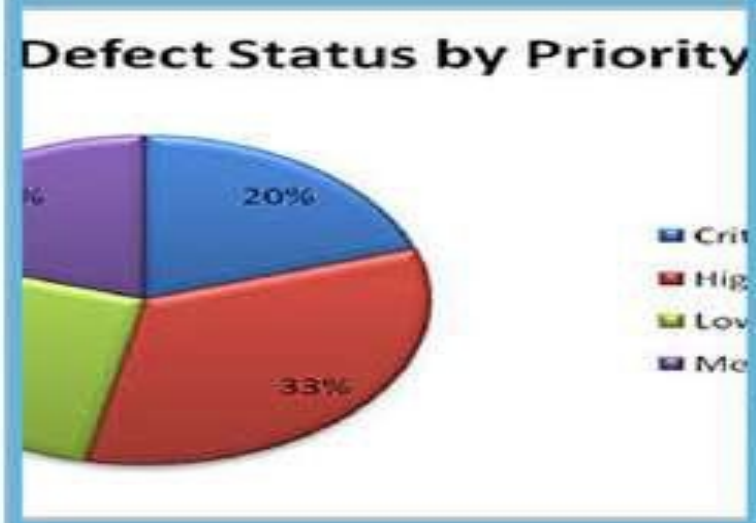
- The limitation of functional testing is that:
  - Hard to know either the extent of redundancy or the possibility of gaps corresponding to the way a set of functional test cases exercises a program

- Test Coverage Metrics
  - device to measure the extent to which a set of test cases covers a program

- **Software Metrics are used to measure the quality of the project**

Test Metrics and Measurements





How can we measure the quality of the software by using Metrics?

1. How many test cases have been designed per requirement?
2. How many test cases are yet to design?
3. How many test cases are executed?
4. How many test cases are passed/failed/blocked?
5. How many test cases are not yet executed?
6. How many defects are identified & what is the severity of those defects?
7. How many test cases are failed due to one particular defect?

- Based on the project needs we can have more metrics than above mentioned list, to know the status of the project in detail.

BENEFITS OF TEST METRICS

- **Based on the metrics, test lead/manager will get the understanding of the below mentioned key points.**

  a)%ge of work completed
  b)%ge of work yet to be completed
  c)Time to complete the remaining work
  d)Whether the project is going as per the schedule or lagging? etc.

DATA FROM TEST ANALYST

S.No.	Testing Metric	Data retrieved during test case development & execution
1	No. of Requirements	5
2	Avg. No. of Test cases written per Requirement	20
3	Total no. of Test cases written for all requirements	100
4	Total no. of Test cases Executed	65
5	No. of Test cases Passed	30
6	No. of Test cases Failed	26
7	No. of Test cases Blocked	9
8	No. of Test cases un executed	35
9	Total No. of Defects identified	30
10	Critical Defects count	6
11	High Defects Count	10
12	Medium Defects Count	6
13	Low Defects Count	8

PROGRAM GRAPH-BASED COVERAGE METRICS

- Given a **program graph**, we can define the following set of test coverage metrics.
  - Node Coverage
  - Edge Coverage
  - Chain Coverage
  - Path Coverage

STRUCTURAL TEST COVERAGE METRICS
(MILLER'S TEST COVERAGE METRICS)

Metric	Description of Coverage
C_0	Every Statement (Statement Coverage based Testing)
C_1	Every DD-Path (Predicate Coverage based Testing)
C_{1P}	Every predicate to each outcome
C_2	C_1 Coverage + Loop Coverage
C_{MCC}	Multiple Condition Coverage
C_d	C_1 Coverage + Every dependent pair of DD-Paths
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

METRIC-BASED TESTING

- **Statement Coverage based Testing (C_0)**:
  - Aims to devise **test cases** that collectively exercise all statements in a program
  - COVERAGE.pdf

- **Predicate Coverage (or Branch Coverage, or Decision Coverage) based Testing (C_1):**
  - Aims to devise test cases that evaluate each simple predicate of the program to **True and False**
  - either a single predicate or a compound Boolean expression considered as a single unit that evaluates to True or False

- This amounts to traversing every edge in the DD-Path graph

EXAMPLE FOR PREDICATE COVERAGE

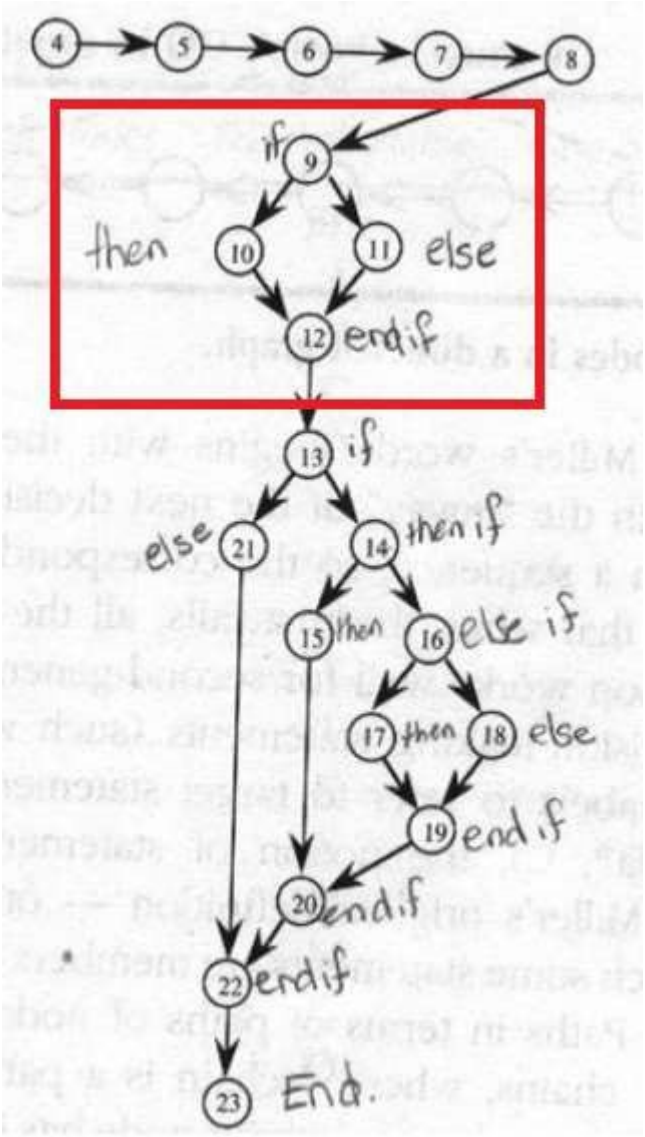
- In predicate coverage, for the condition

if (A or B) then C

We could consider the test cases,

A	B	EXPECTED TEST CASE RESULT (A OR B)
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE

In triangle problem, nodes 9, 10, 11 and 12 are a complete if-then-else statement



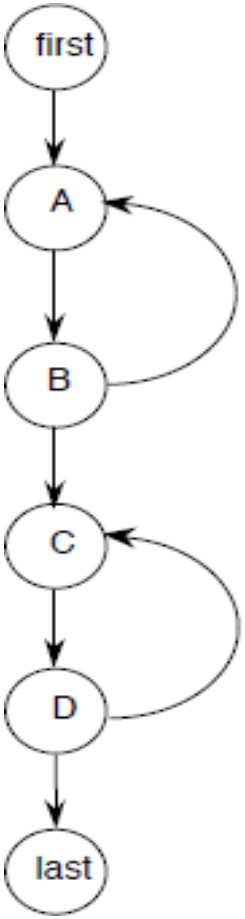
CONDITION TESTING – C_{1P}

- **Predicate Coverage:**
  - is good for exercising faults in the way a computation has been decomposed into cases
- **Condition Coverage:**
  - takes this decomposition in more detail
  - forcing execution of **not only possible outcome** of a Boolean expression
  - but **also of different combinations of the individual conditions** in compound Boolean expression

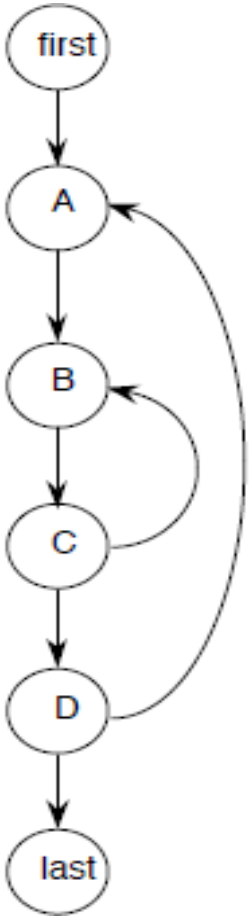
LOOP COVERAGE – C_2

- Loops are a highly prone portion of source code
- **Types of Loops**
- **Concatenated Loops:**
  - a sequence of disjoint loops
- **Nested Loops:**
  - one is contained inside the other
- **Knotted Loops/Unstructured Loops:**
  - When it is possible to branch into or out from the middle of a loop, and these branches are internal to other loops

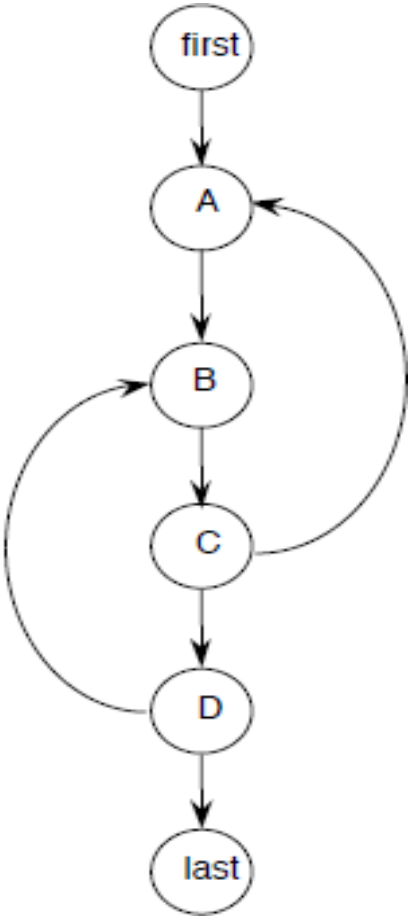
TYPES OF LOOPS



CONCATENATED



NESTED

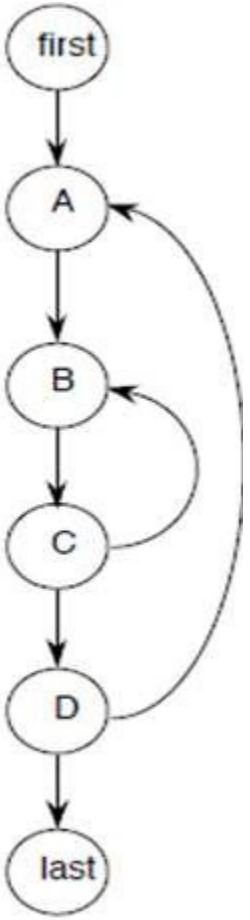


KNOTTED/UNSTRUCTURED

LOOP COVERAGE Cont…

- **Simple Loops,** where *n* is the maximum number of allowable passes through the loop.
  - Skip loop entirely
  - Only one pass through loop
  - Two passes through loop
  - m passes through loop where m<n
  - (n-1), n, and (n+1) passes through the loop

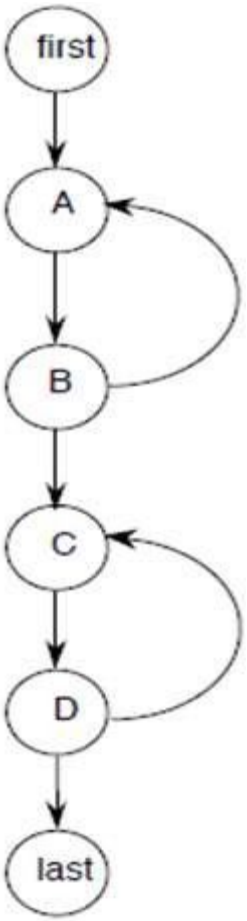
- **Nested Loops**
  - Start with inner loop. Set all other loops to minim values.
  - Conduct simple loop testing on inner loop
  - Work outwards
  - Continue until all loops tested



LOOP COVERAGE Cont…

- **Concatenated Loops**
  - If **independent loops**, use simple loop testing.
  - If **dependent**, treat as nested loops.

- **Unstructured loop**
  - Don't test - redesigr



LOOP COVERAGE Cont…

- Every loop involves a decision, and we need to test both outcomes of the decision (traverse loop or exit)
- **Boundary Value Analysis (BVA) or Robustness Testing can be done on the index of the loops**
- If the body of a **simple loops** is a DD-Path that performs a complex calculation, functional testing could also be used
- Once a loop has been tested, it should be condensed into a single node
- If **loops are nested**, this process is repeated starting with the

innermost loop and **working outward**

MULTIPLE CONDITION COVERAGE - C_{MCC}

- 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)

MULTIPLE CONDITION COVERAGE Cont…

- Multiple Condition testing for:

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

MULTIPLE CONDITION COVERAGE Cont…

Truth Table for Triangle Inequality

(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

MULTIPLE CONDITION COVERAGE Cont…

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 COVERAGE Cont…

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

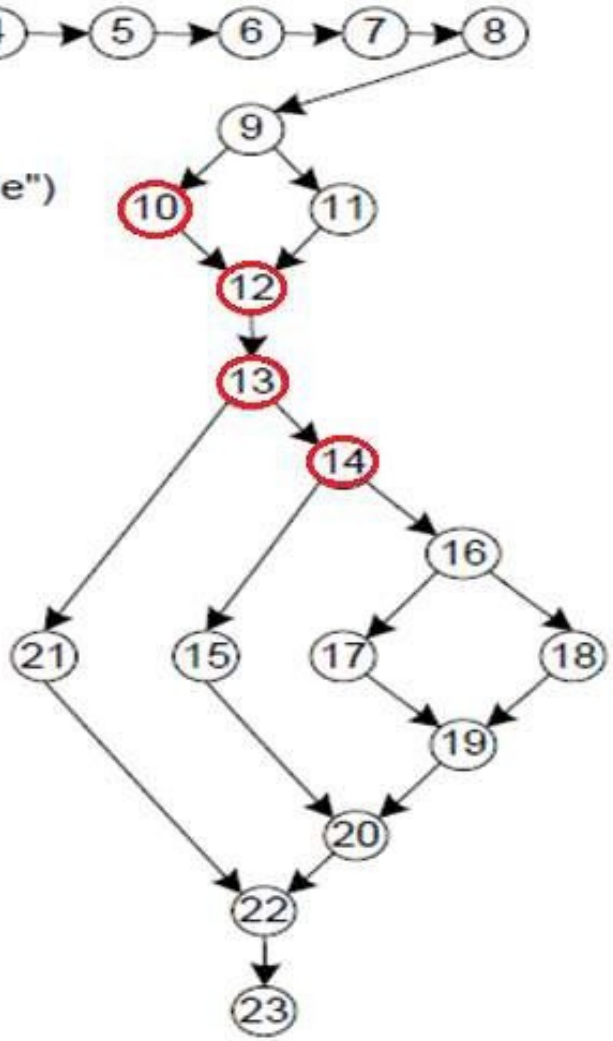
Note: could add test cases for c = a + b, b = a + c, and a = b + c.

DEPENDENT DD-PATHS C_d COVERAGE

- In the triangle problem code and program graph:
  - 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 metrics

if a path traverses node 10
Then IsATriangle = True,
then it must traverse node 14

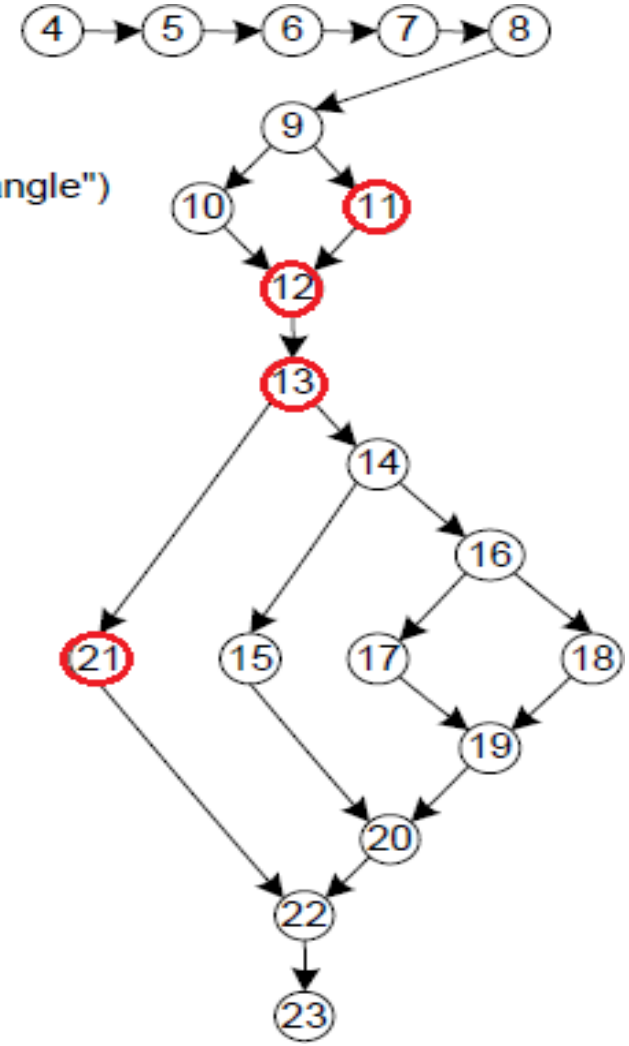
```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATriangle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is ",a)
7  Output("Side B is ",b)
8  Output("Side C is ",c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10   Then IsATriangle = True
11   Else IsATriangle = False
12 EndIf
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
```



Paths through nodes 10 and 21 are "infeasible"

if a path traverses node 11 (Else IsATriangle= False), then it must traverse node 21

```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATriangle As Boolean
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)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12 EndIf
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
```



Paths through nodes 11 and 14 are "infeasible"

BASIC PATH TESTING

SAMPLE QUESTIONS

- Explain about test coverage metrics.
- Explain McCabe's basis path method.
- Discuss test coverage metrics and basis path testing with example.
- Explain in detailaboutMcCabe's basis path method using graph theory.

BASIC PATH TESTING

- White-box testing technique proposed by Tom McCabe
- Enables the test case designer to **derive a logical complexity** measure of a procedural design
- Uses 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

- **Circle:**
  - in a graph represents a node, which stands for a sequence of one or more procedural statements

- **Node:**
  - Containing a **simple conditional expression** is referred to as a predicate node
  - Each **compound condition** in a conditional expression containing one or more Boolean operators (e.g., and, or) is represented by a separate predicate node
  - A predicate node has **two** edges leading out from it (True and False)

FLOW GRAPH NOTATION Cont…

- **Edge / Link:**
  - An arrow representing flow of control in a specific direction
  - An edge must start and terminate at a node
  - An edge does not intersect or cross over another edge

- **Regions:**
  - Areas bounded by a set of edges and nodes are called regions
  - When counting regions, include the area outside the graph as a region, too

BASIS PATH TESTING PROCESS

- **INPUT**
  - Source code and a path selection criterion

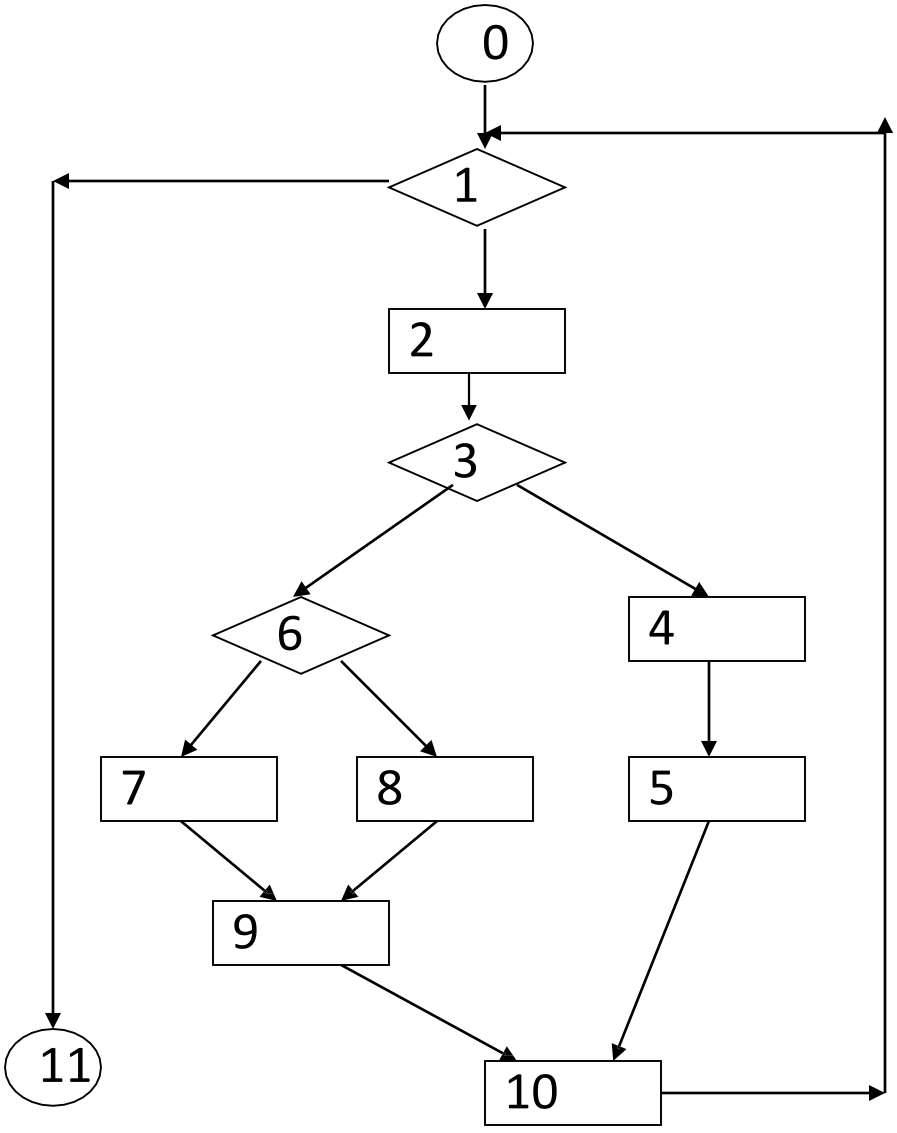
- **PROCESS**
  1. Generation/Construction of **Control Flow Graph**

using the design or code as foundation

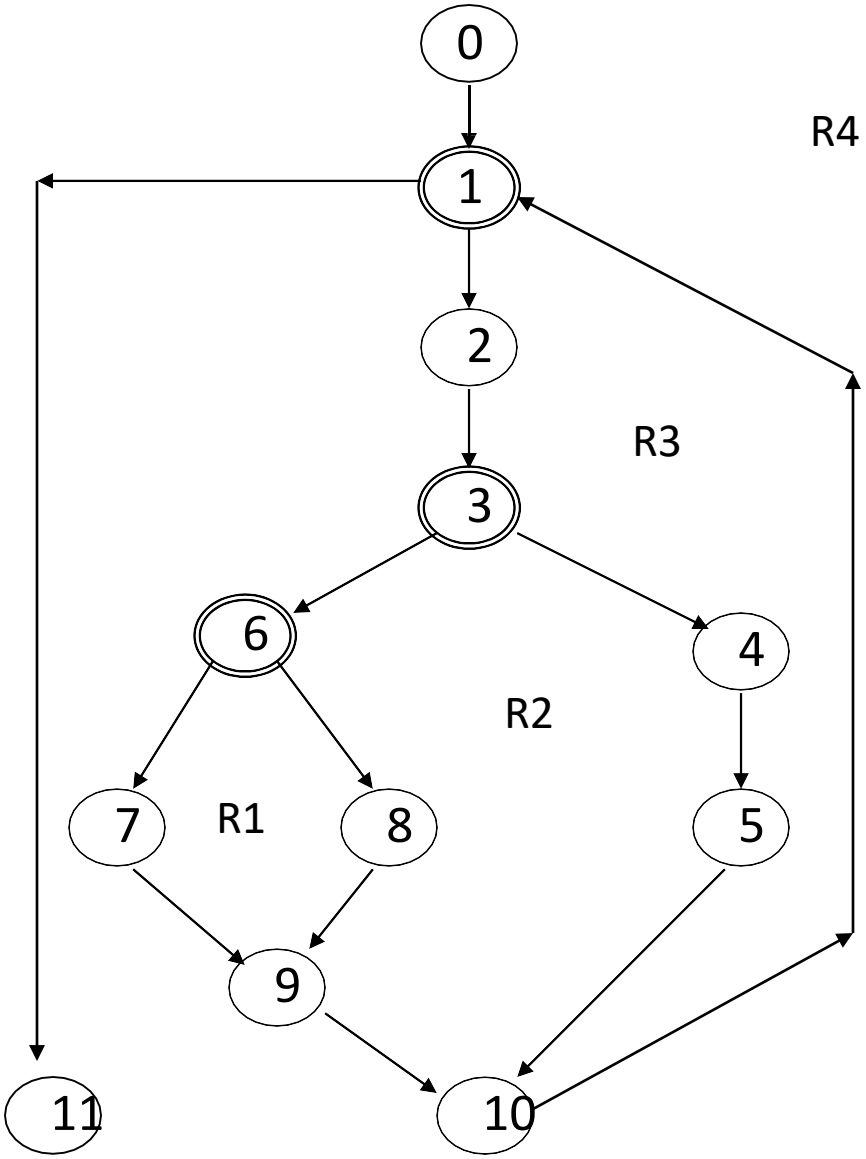
  2. Determine the **Cyclomatic Complexity** of the resultant flow graph
  3. Determine a **basis set of linearly independent paths**
  4. Selection of Paths
  5. **Prepare test cases** that will force execution of each path

1. FLOW GRAPH NOTATION

FLOW CHART



CONTROL FLOW GRAPH



2. CYCLOMATIC COMPLEXITY

- Provides a quantitative measure of the **logical complexity** of a program
- Defines the **number of independent paths** in the basis set
- Provides an **upper bound** for the number of tests that must be conducted to ensure **all statements** have been executed **at least once**

COMPUTING CYCLOMATIC COMPLEXITY

- Can be computed **three** ways

1.Number of regions 2. V(G) = E – N + 2

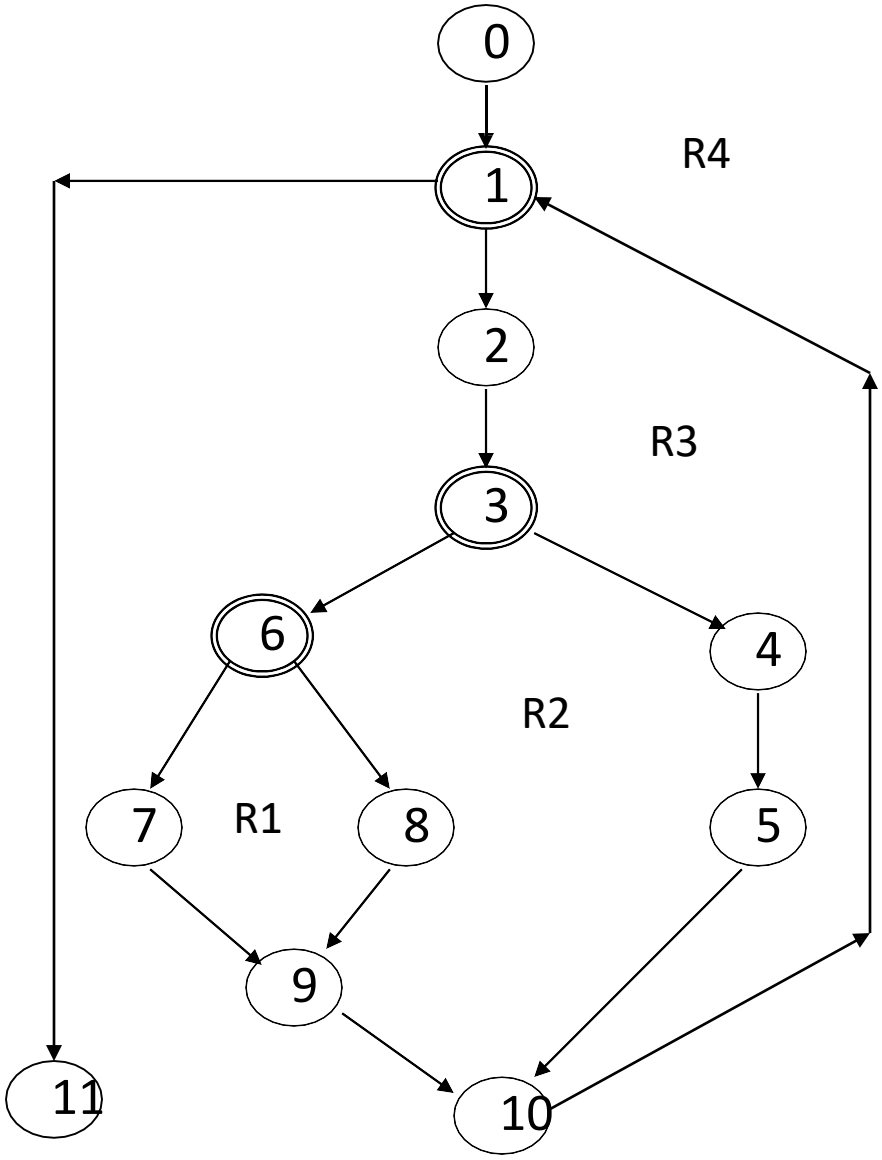
where

- E → Number of edges and
- N → Number of nodes in graph G

3. V(G) = P + 1

where

- P → Number of **predicate nodes** in the flow graph G



Number of regions = 4

V(G) = E – N + 2
E = 14, N = 12

V(G) = P + 1
P = 3

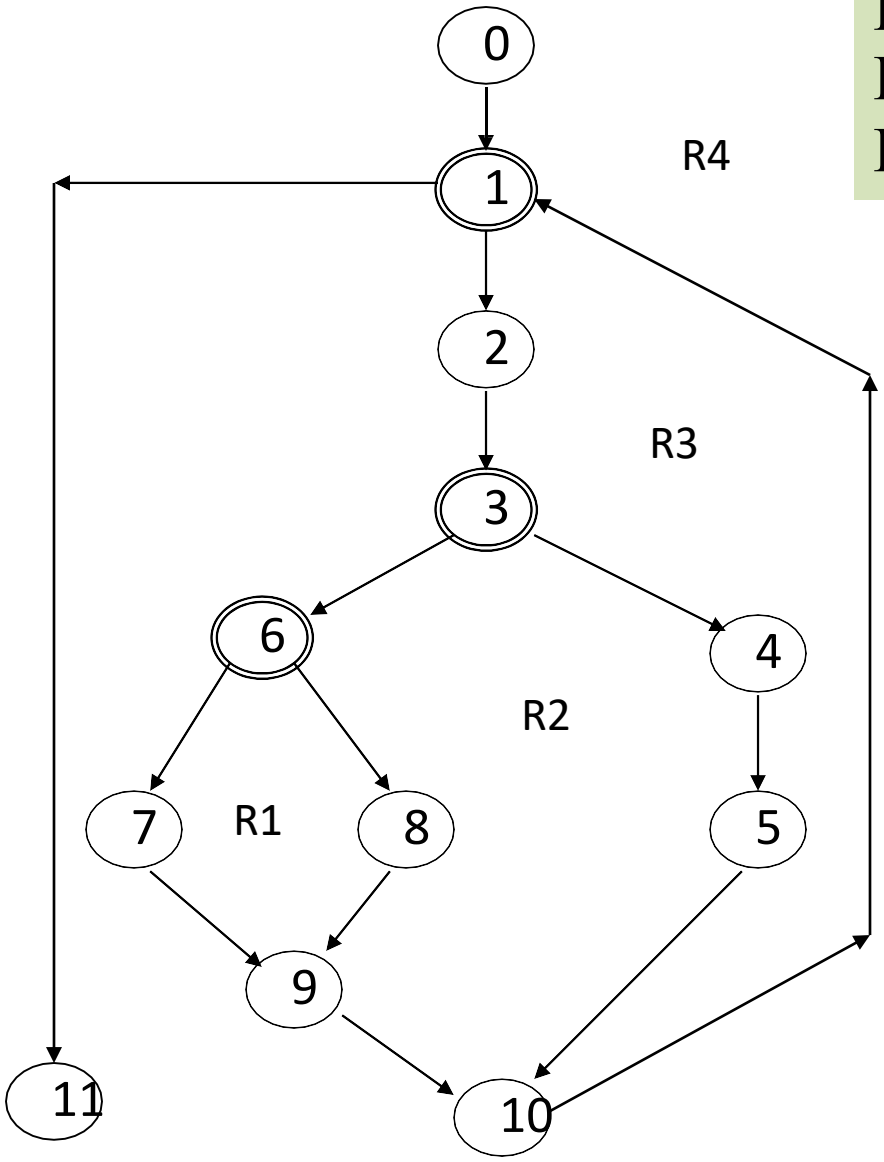
COMPLEXITY NUMBER AND ITS MEANING

Complexity Number	Meaning
1-10	Structured and well written code High Testability Cost and Effort is less
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

INDEPENDENT PROGRAM PATHS

- **INDEPENDENT PATH**
  - the program from the **start node** until the **end node** that introduces **at least one new set of processing statements or a new condition** (i.e., new nodes)
  - Must move along **at least one** edge that has not been traversed before by a previous path

- The **number of paths** in the basis set is determined by the **Cyclomatic Complexity**



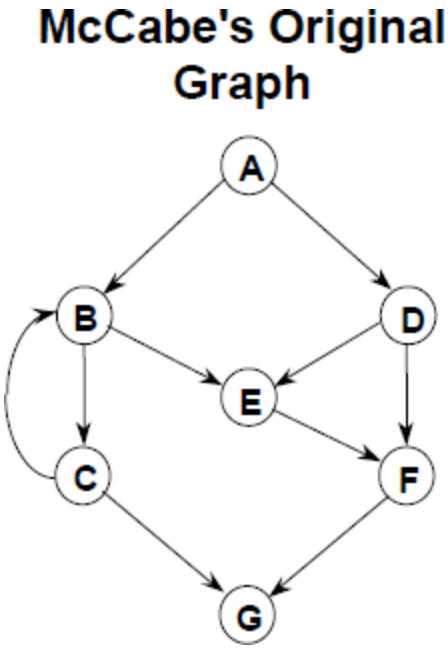
Basis set for flow graph Path 1:
0-1-11
Path 2: 0-1-2-3-6-7-9-10-1-11
Path 3: 0-1-2-3-6-8-9-10-1-11
Path 4: 0-1-2-3-4-5-10-1-11

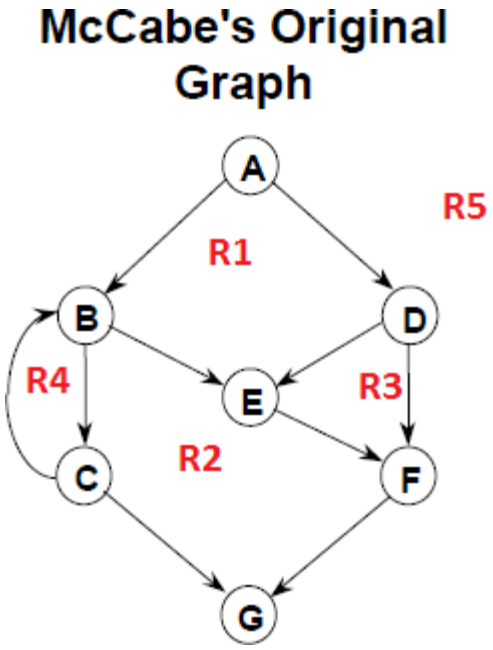
Number of regions = 4

V(G) = E – N + 2
E = 14, N = 12

V(G) = P + 1
P = 3

McCABE'S EXAMPLE CONTROL FLOW GRAPH



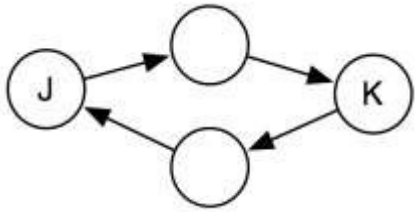


LINEAR INDEPENDENT PATH FROM THE SOURCE NODE TO SINK NODE

Some source gives the formula as: **V(G)=e-n+2p**

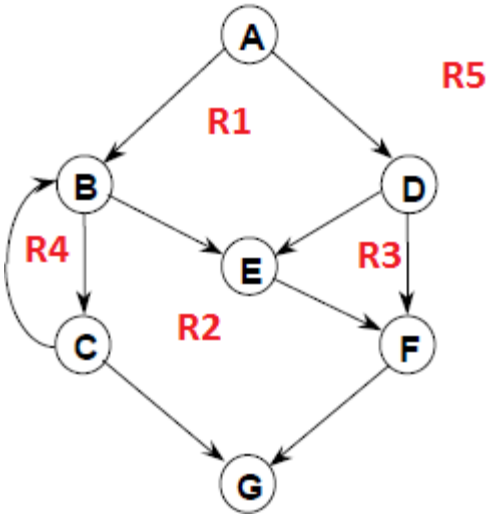
McCABE'S EXAMPLE CONTROL FLOW GRAPH – STRONGLY CONNECTED GRAPH

- **CIRCUIT**
  - A circuit is similar to a chain; no internal loops or decisions occur, but the **initial node is the terminal node**
  - A circuit is a set of **3-connected nodes**

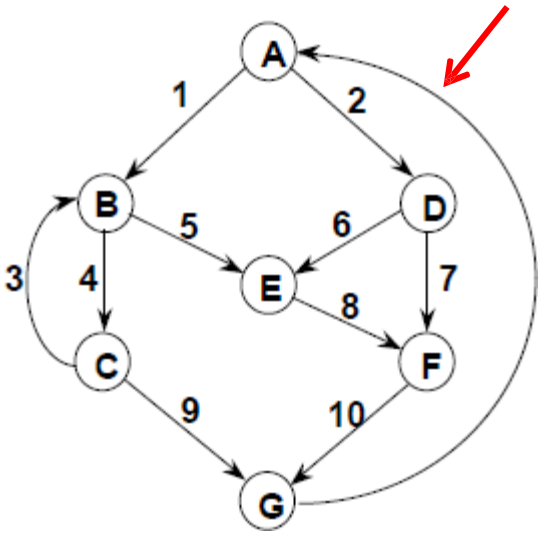


- **Creating Strongly Connected Graph**
  - Can be created by **adding an edge from the (every) sink node to the (every) source node**

McCabe's Original Graph



Derived, Strongly Connected Graph



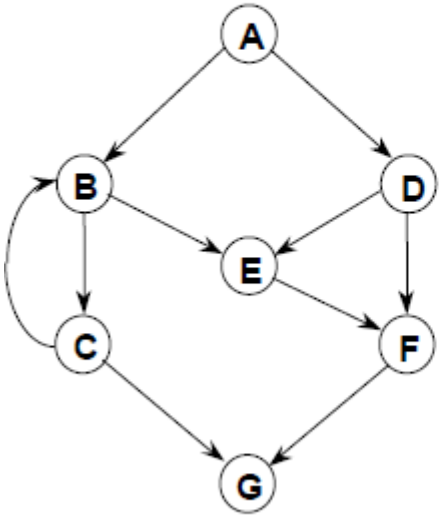
LINEAR INDEPENDENT CIRCUITS

Some source gives the formula as:

V(G)=e-n+p

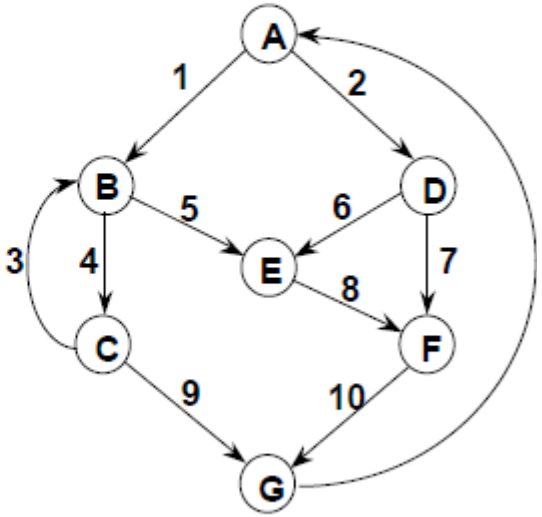
McCabe's Example

McCabe's Original Graph

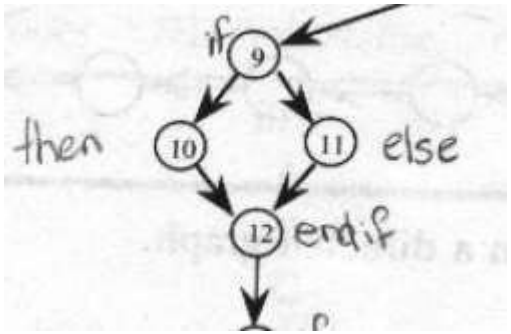


V(G) = 10 - 7 + 2(1)
= 5

Derived, Strongly Connected Graph



V(G) = 11 - 7 + 1
= 5



LINEAR INDEPENDENT CIRCUITS

Some source gives the formula as:

V(G)=e-n+p

VECTOR SPACE OF PROGRAM PATH

- **VECTOR SPACE – DEFINITION**

  – A vector space is a **set of elements** along with certain **operations that can be performed** upon these elements

  – **Notations defined**:
    – **addition** and
    – **multiplication**

  – **Path Addition** → Simply one path followed by another path

  – **Path Multiplication** → Corresponds to repetitions of path

- **Two important points:**

1. If there is a **loop**, it only has to be **traversed once**, or **else** the basis will **contain redundant paths**.

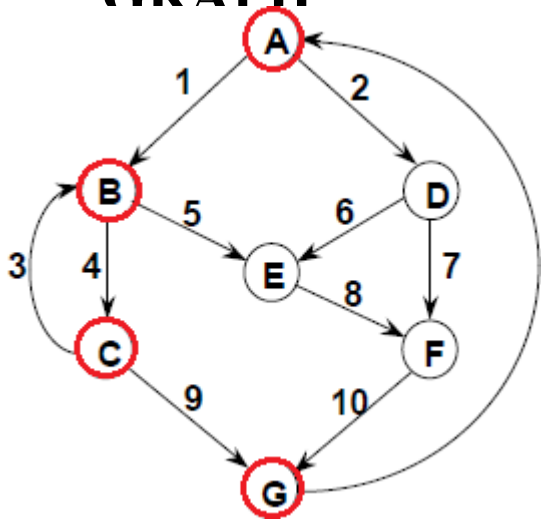
2. It is possible for there to be more than one basis; the property of uniqueness is one not required.

CYCLOMATIC COMPLEXITY OF STRONGLY CONNECTED GRAPH

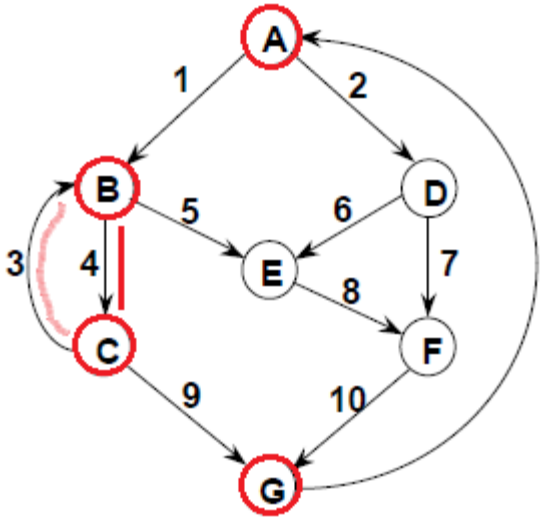
FIVE LINEAR INDEPENDENT CIRCUITS

P1: A, B, C, G
P2: A, B, C, B, C, G
P3: A, B, E, F, G
P4: A, D, E, F, G
P5: A, D, F, G

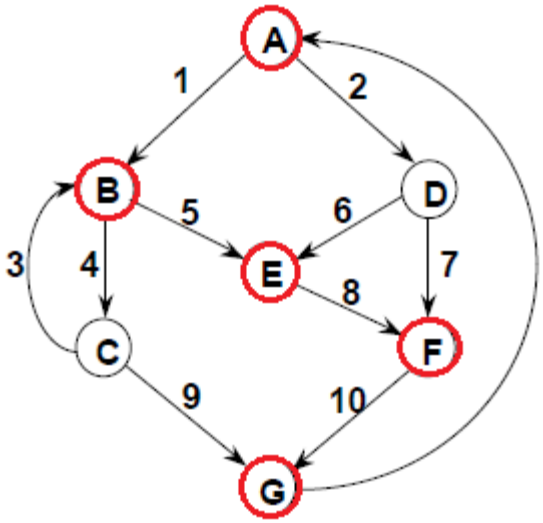
P1



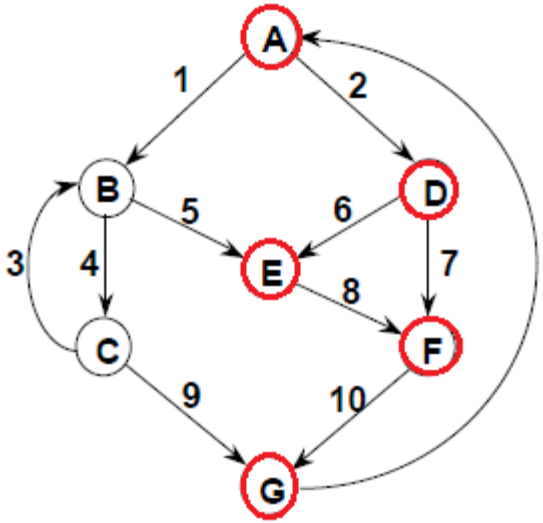
P2



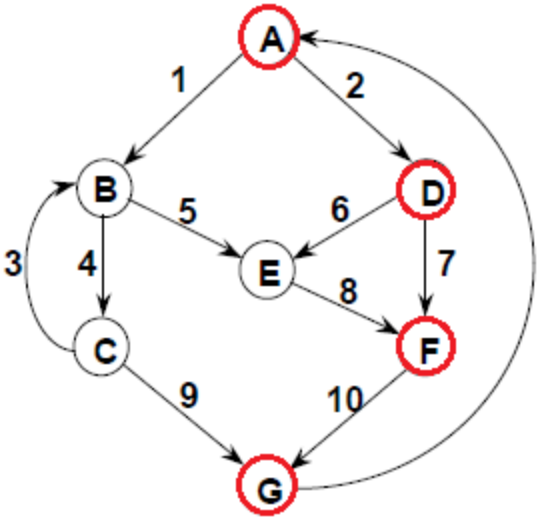
P3



P4



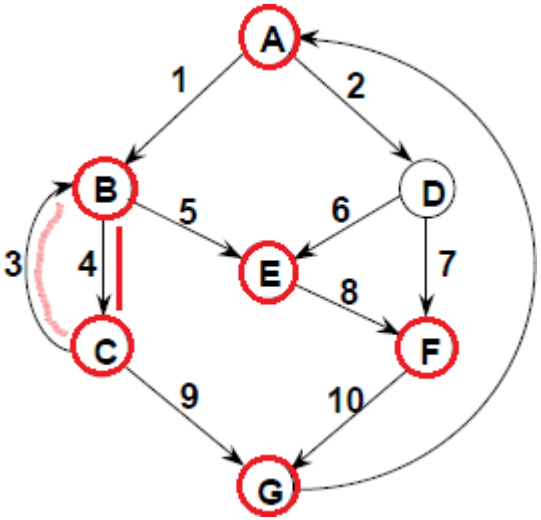
P5



- Basic notions of scalar multiplication and addition:
  - we should now be able to construct any path from our basis
- Let us attempt to create a **6th path:**
  - A, B, C, B, E, F, G.
- This is the basis **sum p2 + p3 – p1.**
- This equation means to concatenate paths 2 and 3 together to form the path:

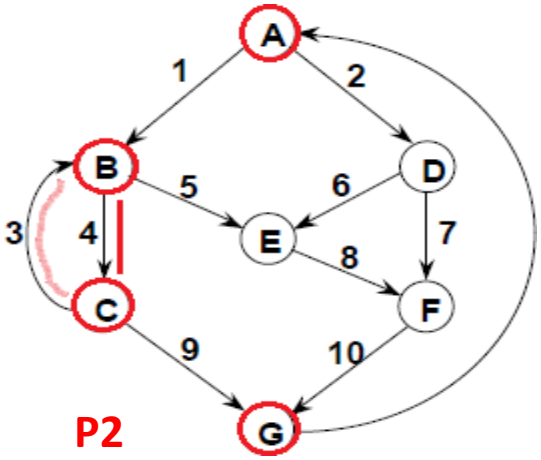
  A, B, C, **B, C, G, A,** B, E, F, G

  and then remove the four nodes that appear in path 1, resulting in the required path 6.



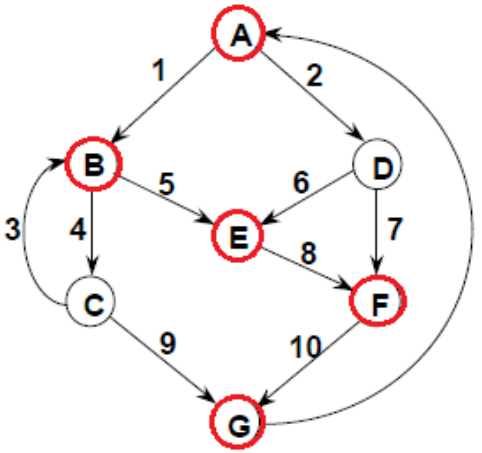
P2: A, B, C, B, C, G
P3: A, B, E, F, G
P1: A, B, C, G

P6=P2+P3-P1



P2

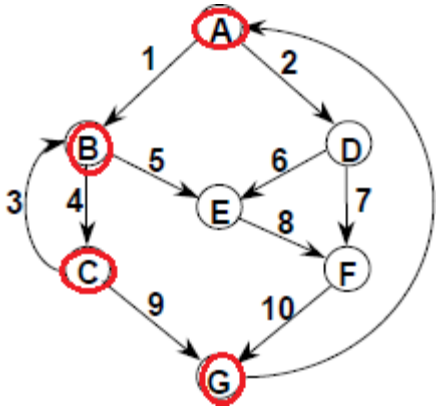
P3



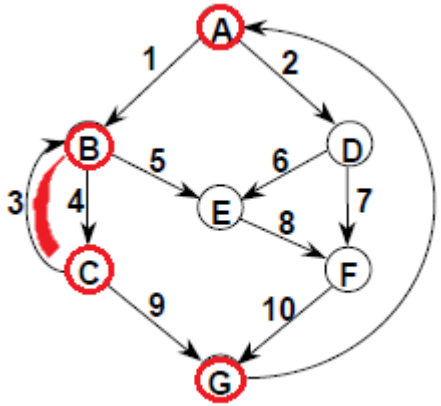
ex1 = p2 + p3 - p1
ex2 = 2p2 - p1

Basis Paths

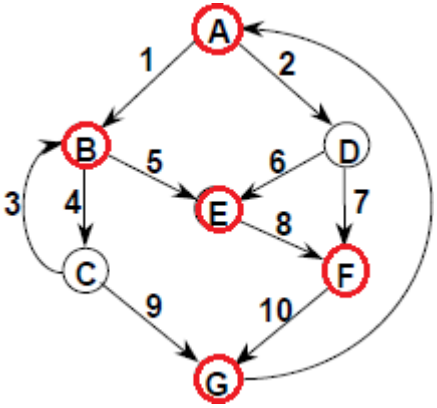
path \ edges traversed	1	2	3	4	5	6	7	8	9	10
p1: A, B, C, G	1	0	0	1	0	0	0	0	1	0
p2: A, B, C, B, C, G	1	0	1	2	0	0	0	0	1	0
p3: A, B, E, F, G	1	0	0	0	1	0	0	1	0	1
p4: A, D, E, F, G	0	1	0	0	0	1	0	1	0	1
p5: A, D, F, G	0	1	0	0	0	0	1	0	0	1
ex1: A, B, C, B, E, F, G	1	0	1	1	1	0	0	1	0	1
ex2: A, B, C, B, C, B, C, G	1	0	2	3	0	0	0	0	1	0



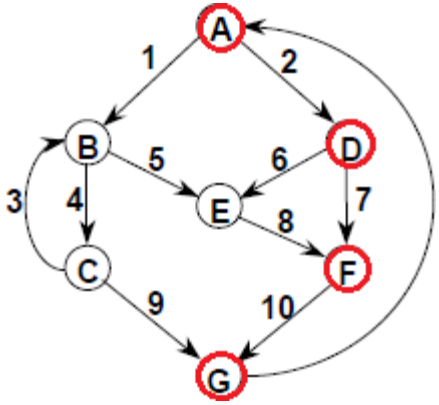
p1



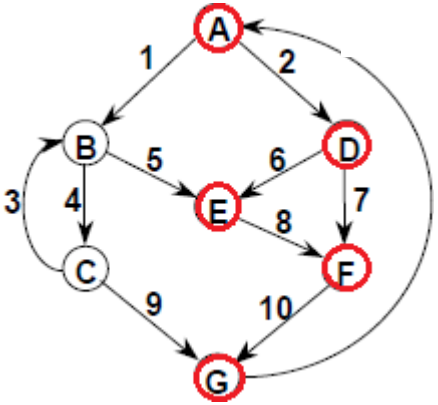
p2



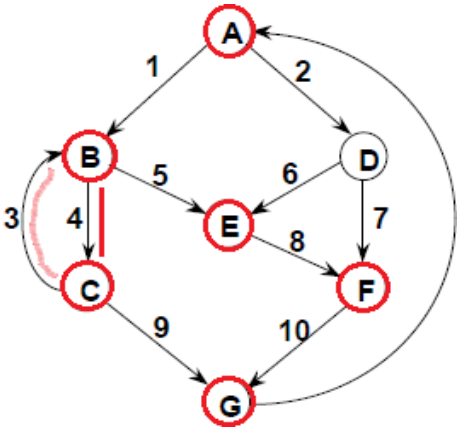
p3



p4



p5



ex1

Table 3 Path/Edge Traversal

path \ edges traversed	1	2	3	4	5	6	7	8	9	10
p1: A, B, C, G	1	0	0	1	0	0	0	0	1	0
p2: A, B, C, B, C, G	1	0	1	2	0	0	0	0	1	0
p3: A, B, E, F, G	1	0	0	0	1	0	0	1	0	1
p4: A, D, E, F, G	0	1	0	0	0	1	0	1	0	1
p5: A, D, F, G	0	1	0	0	0	0	1	0	0	1
ex1: A, B, C, B, E, F, G	1	0	1	1	1	0	0	1	0	1
ex2: A, B, C, B, C, B, C, G	1	0	2	3	0	0	0	0	1	0

p1: A, B, C, G
p2: A, B, C, B, C, G
p3: A, B, E, F, G
p4: A, D, E, F, G
p5: A, D, F, G

ex1=p2+p3-p1

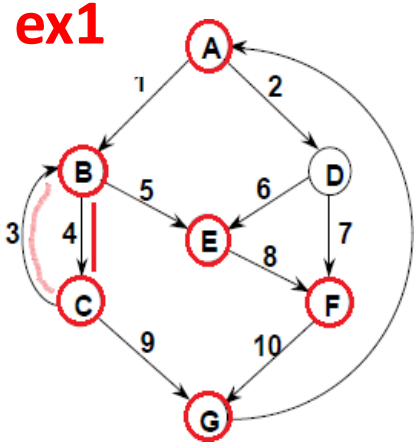
Ex1= (1,0,1,2,0,0,0,0,1,0) +(1,0,0,0,1,0,0,1,0,1)-(1,0,0,1,0,0,0,0,1,0)

Ex1 = (1,0,1,1,1,0,0,1,0,1)=A,B,C,B,E,F,G

ex2=2p2-p1

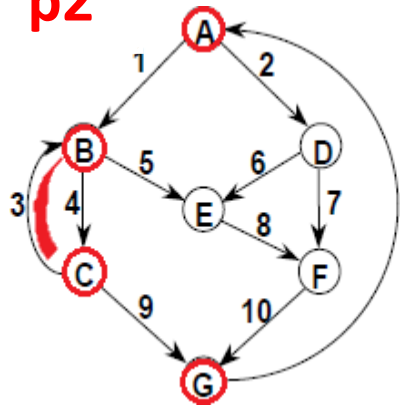
Ex2=(2,0,2,4,0,0,0,0,2,0) – (1,0,0,1,0,0,0,0,1,0)= (1,0,2,3,0,0,0,0,1,0)=A,B,C,B,C,B,C,G

ex1



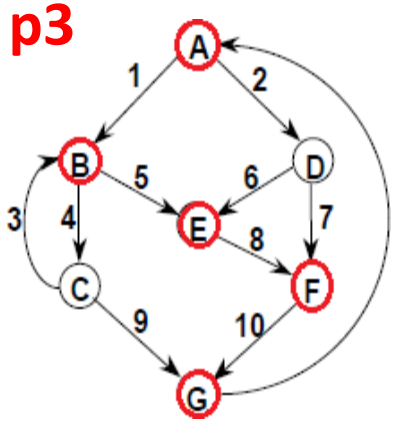
=

p2



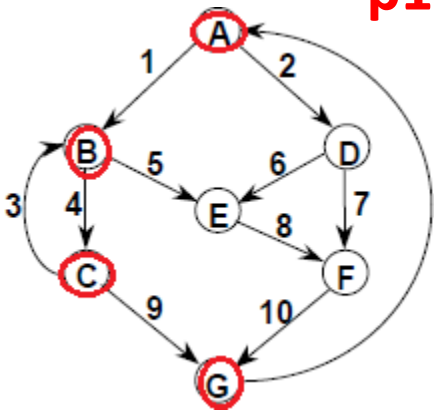
+

p3



-

p1



PROBLEM WITH BASIS PATH METHOD

- When constructing the basis set of a program graph, it is possible that some of the **paths** that make up the basis could **be infeasible**.

- **INFEASIBLE PATH**
  - An infeasible path is simply any path that cannot be traversed by test cases (or) path that cannot be verified by any set of possible input values.

- **SOLUTION**
  - McCabe develops an **algorithmic procedure called baseline method to determine a set of basis paths**

McCABE'S BASELINE METHOD

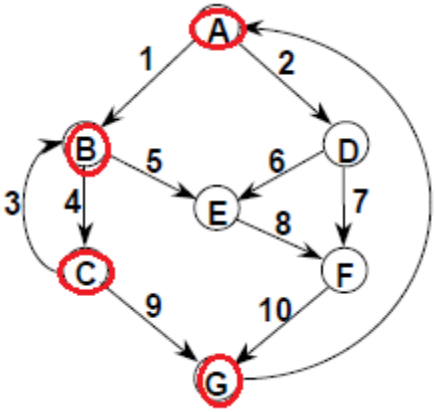
Method to find the independent paths using Baseline Path

- To determine a set of basis paths,

1. Pick a "baseline" path that corresponds to normal execution and generate one feasible path. (The baseline should have as many decisions as possible.)

2. To get succeeding basis paths, retrace the baseline until you reach a destination node. "Flip" the decision (take another alternative) when a node of outdegree >=2 is reached, a different edge must be taken.

3. Repeat this until all decisions have been flipped. When you reach V(G) basis paths, you are done



Resulting basis paths

First baseline path

p1: A, B, C, G

Flip decision at C

p2: A, B, C, B, C, G

Flip decision at B

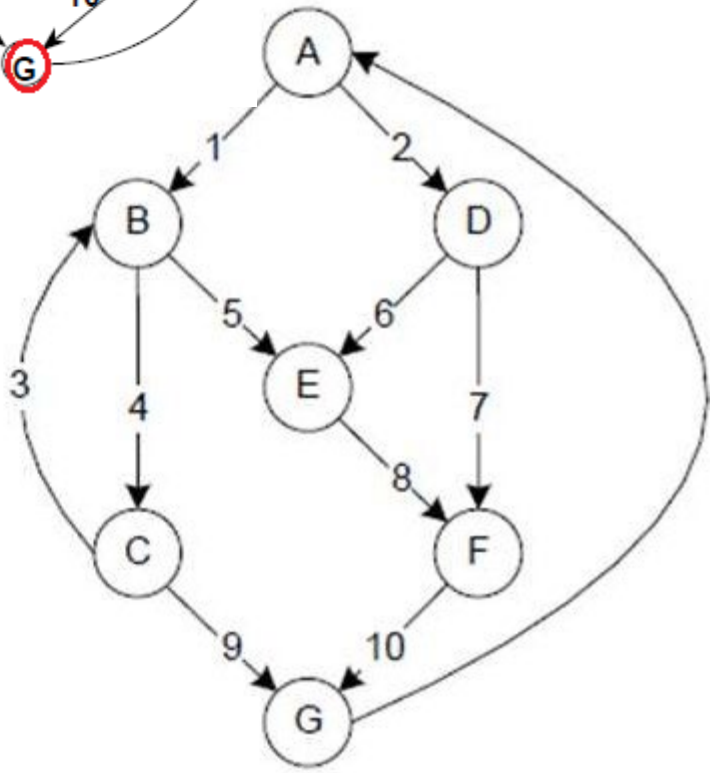
p3: A, B, E, F, G

Flip decision at A

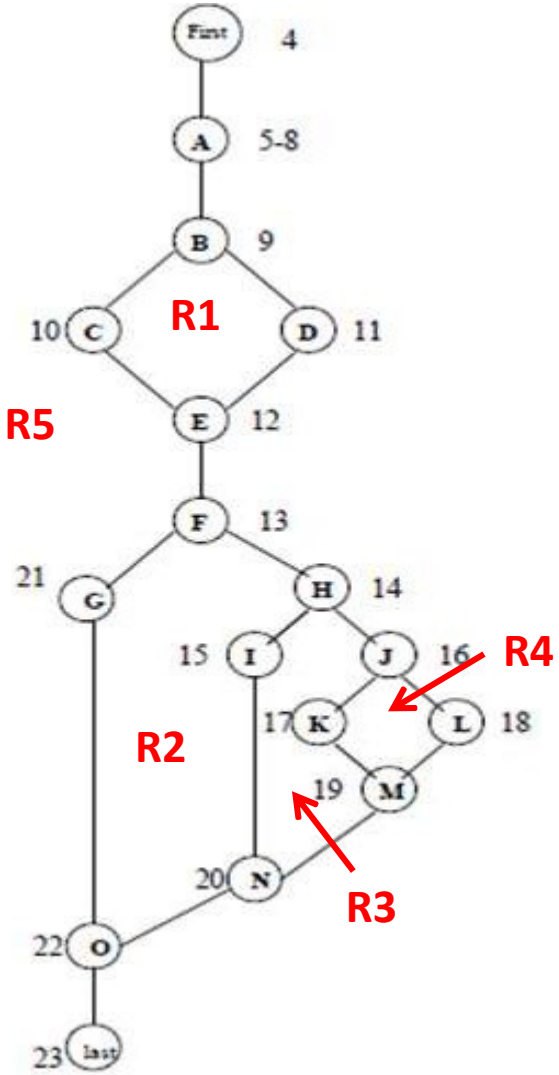
p4: A, D, E, F, G

Flip decision at D

p5: A, D, F, G

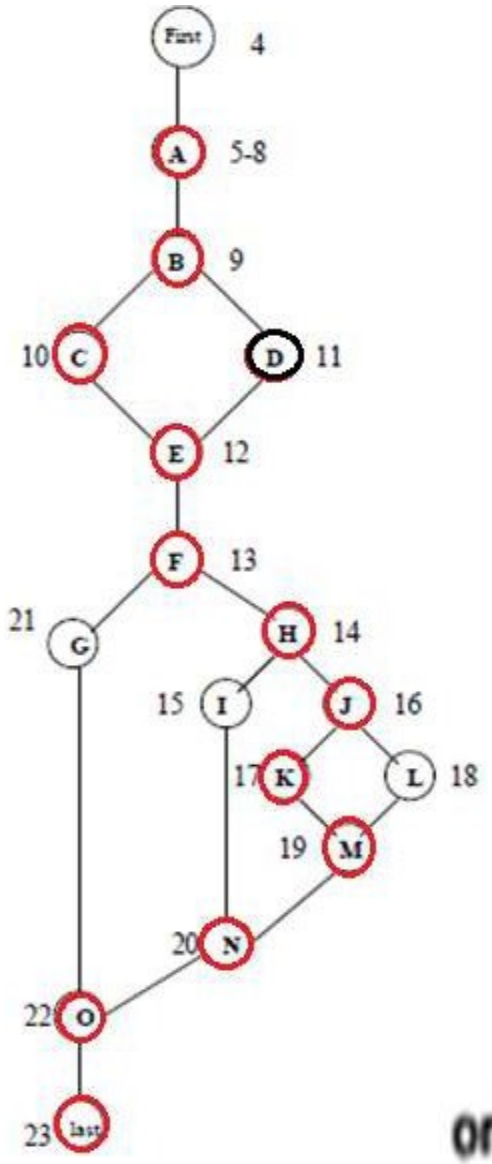


OBSERVATIONS ON McCABE'S BASIS PATH METHOD



There are 8 topologically possible paths.
4 are feasible, and 4 are infeasible.

original	p1: A-B-C-E-F-H-J-K-M-N-O-Last.	Scalene
flip p1 at B	p2: A-B-D-E-F-H-J-K-M-N-O-Last	Infeasible
flip p1 at F	p3: A-B-C-E-F-G-O-Last	Infeasible
flip p1 at H	p4: A-B-C-E-F-H-I-N-O-Last	Equilateral
flip p1 at J	p5: A-B-C-E-F-H-J-L-M-N-O-Last	Isosceles

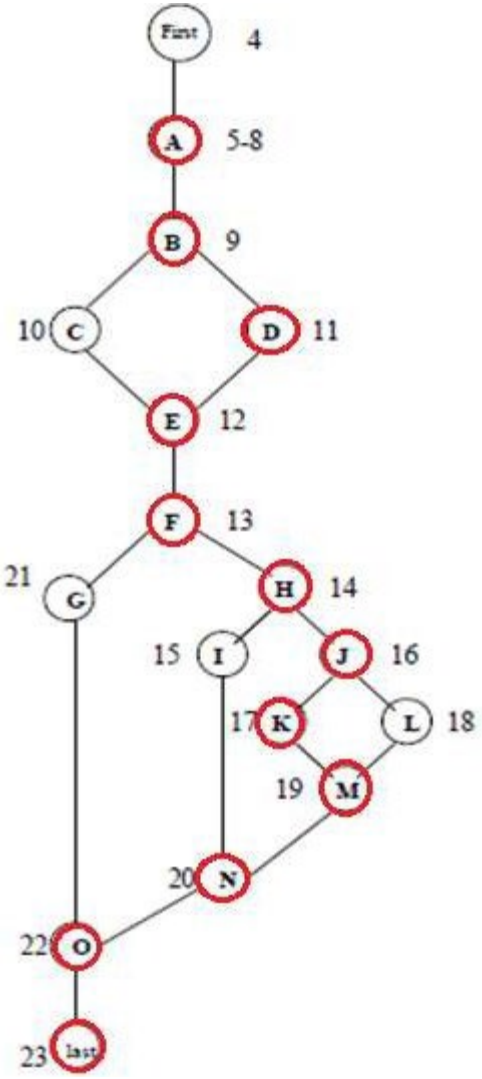



```
1.  Program triangle
2.  Dim a,b,c As Integer
3.  Dim IsATriangle As Boolean
    'Step 1:  Get Input
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
9.  If (a < 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
```

original p1: A-B-C-E-F-H-J-K-M-N-O-Last. Scalene

FEASIBLE PATH - SCALENE

FLIP P1 AT B



flip p1 at B **p2: A-B-D-E-F-H-J-K-M-N-O-Last** **Infeasible**

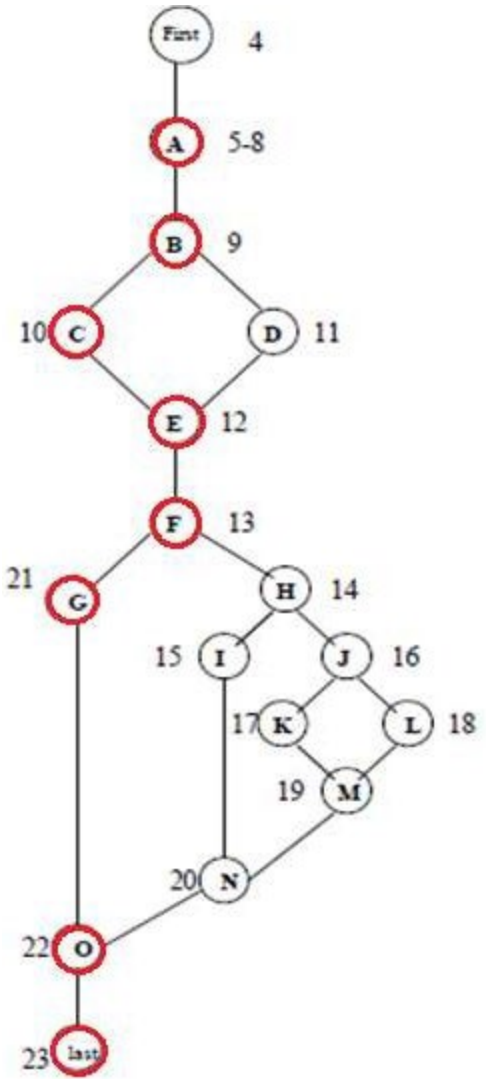
```
1.  Program triangle
2.  Dim a,b,c As Integer
3.  Dim IsATriangle As Boolean
    'Step 1:  Get Input
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
9.  If (a < 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
```

X

✓

INFEASIBLE PATH

FLIP P1 AT F



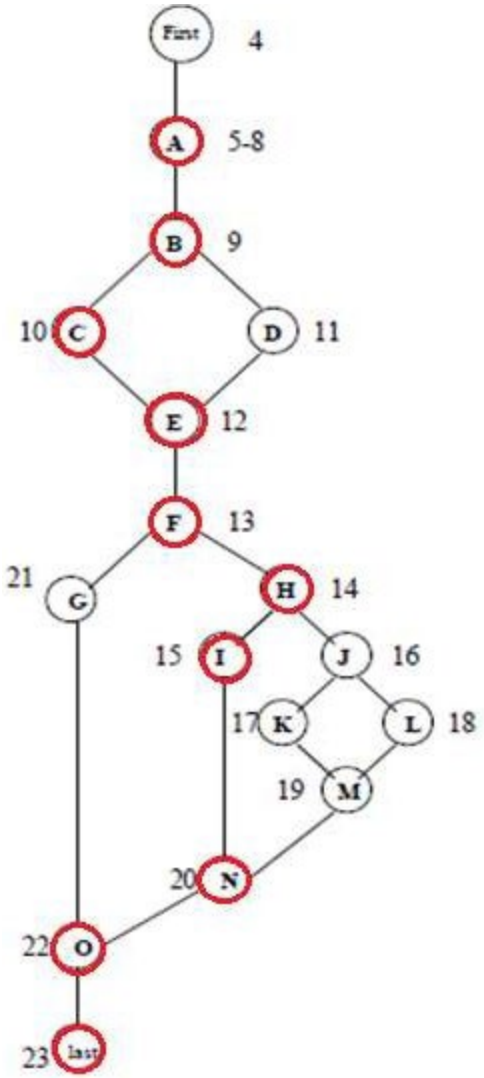

```
1.  Program triangle
2.  Dim a,b,c As Integer
3.  Dim IsATriangle As Boolean
    'Step 1:  Get Input
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
9.  If (a < 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
```

p3: A-B-C-E-F-G-O-Last Infeasible

INFEASIBLE PATH

FLIP P1 AT H

p4: A-B-C-E-F-H-I-N-O-Last Equilateral

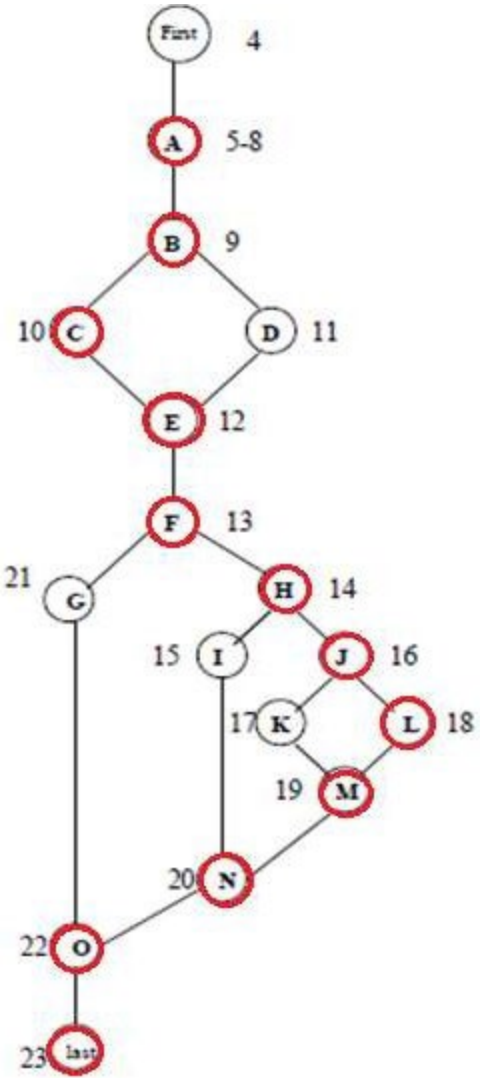



```
1.  Program triangle
2.  Dim a,b,c As Integer
3.  Dim IsATriangle As Boolean
    'Step 1:  Get Input
4.  Output("Enter 3 integers which are sides of a triangle")
5.  Input(a,b,c)
6.  Output("Side A is ",a)
7.  Output("Side B is ",b)
8.  Output("Side C is ",c)
    'Step 2:  Is A Triangle?
9.  If (a < 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
```

FEASIBLE PATH - EQUILATERAL

FLIP P1 AT J

p5: A-B-C-E-F-H-J-L-M-N-O-Last Isosceles




```
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
```

FEASIBLE PATH - ISOSCELES

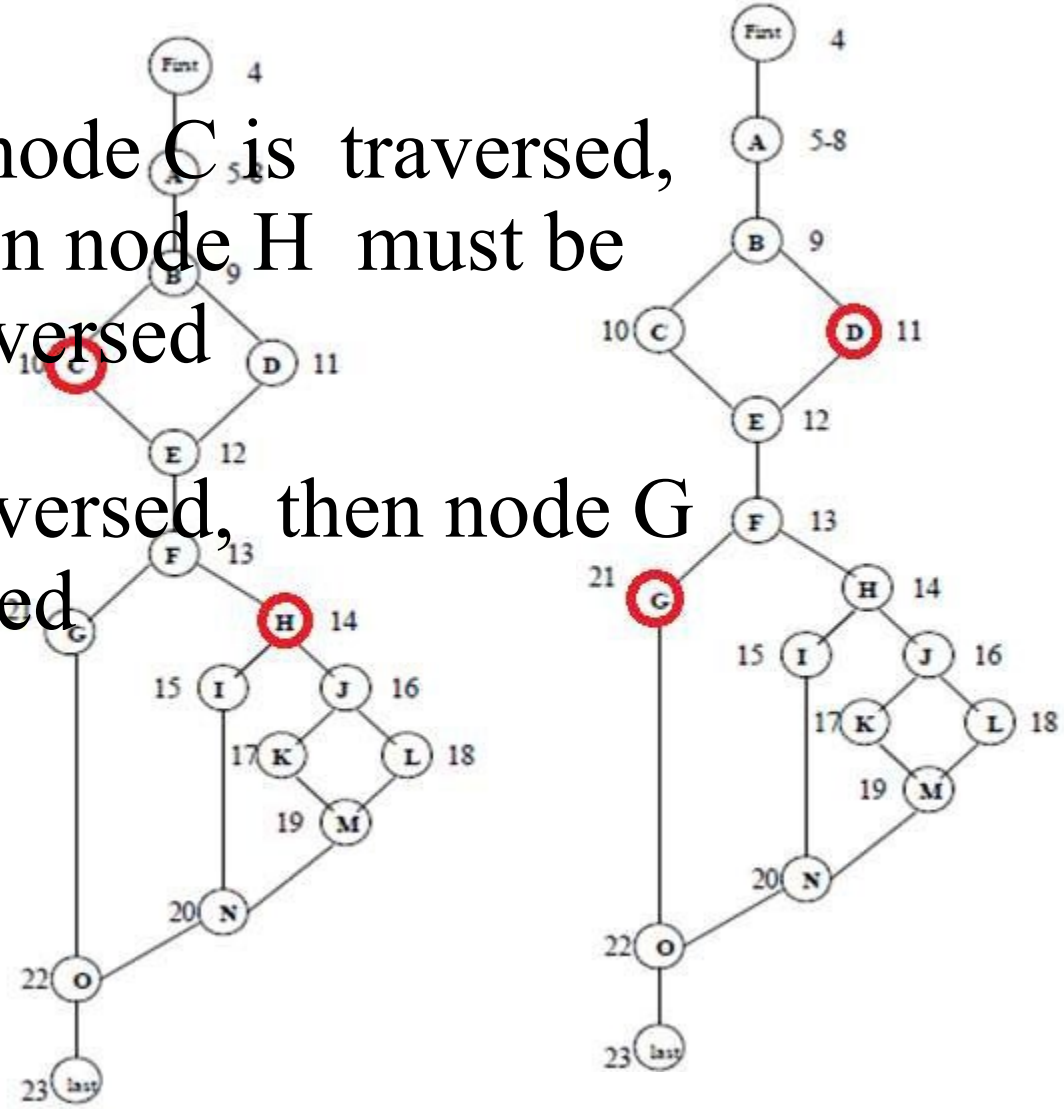
REASONS FOR INFEASIBLE PATH AND SOLUTION

- **Reason:**
  - Basis path are **topologically independent, but** when these contradict **semantic dependencies,** topologically **possible paths** are **seen** to be **logically infeasible**
  - A technically feasible path may not be feasible logically
- **Solution:**
  - Requires that flipping a decision results in a semantically feasible path
  - Check the logical dependencies before flipping

SOLUTION FOR INFEASIBLE PATHS IN TRIANGLE PROBLEM

- If node C is traversed, then node H must be traversed

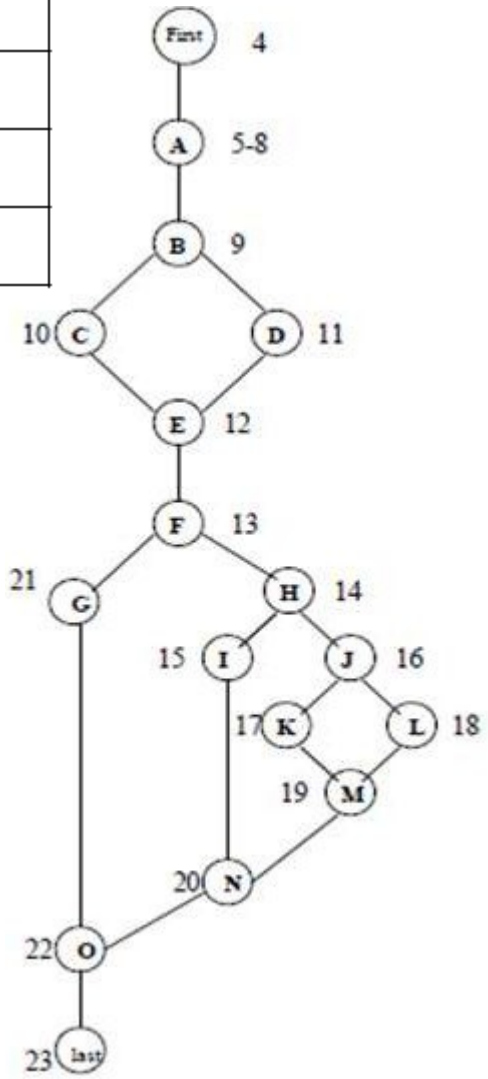
- If node D is traversed, then node G must be traversed



FEASIBLE PATHS FOR TRIANGLE PROBLEM

P1:A-B-C-E-F-H-J-K-M-N-O-Last	SCALENE
P6:A-B-D-E-F-G-O-Last	NOT A TRIANGLE
P3:A-B-C-E-F-H-I-N-O-Last	EQUILATERAL
P4:A-B-C-E-F-H-J-L-M-N-O-Last	ISOSCELES

For this special case, the test cases can be formed by using
Special Value Testing and Output Range Testing

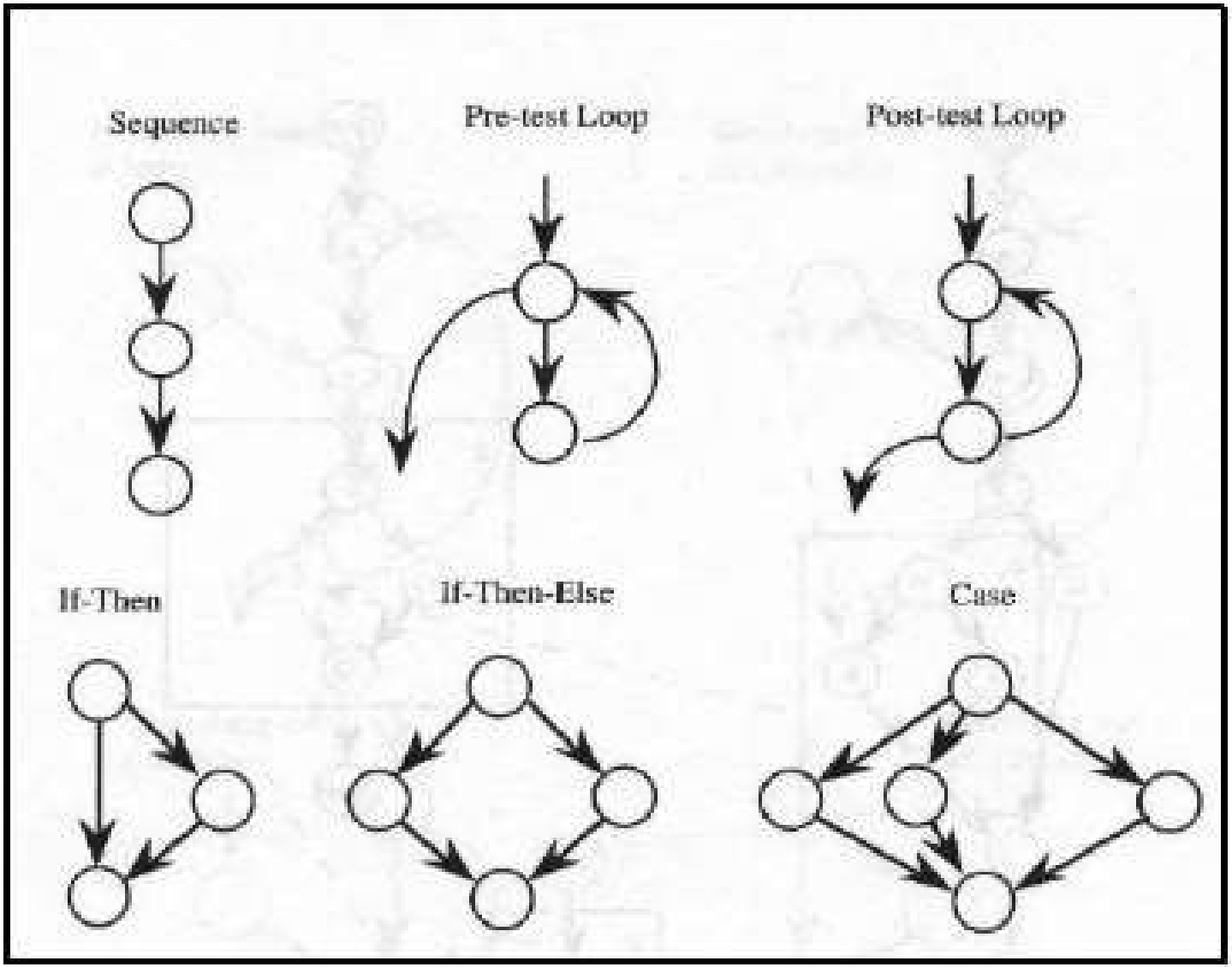


Essential Complexity

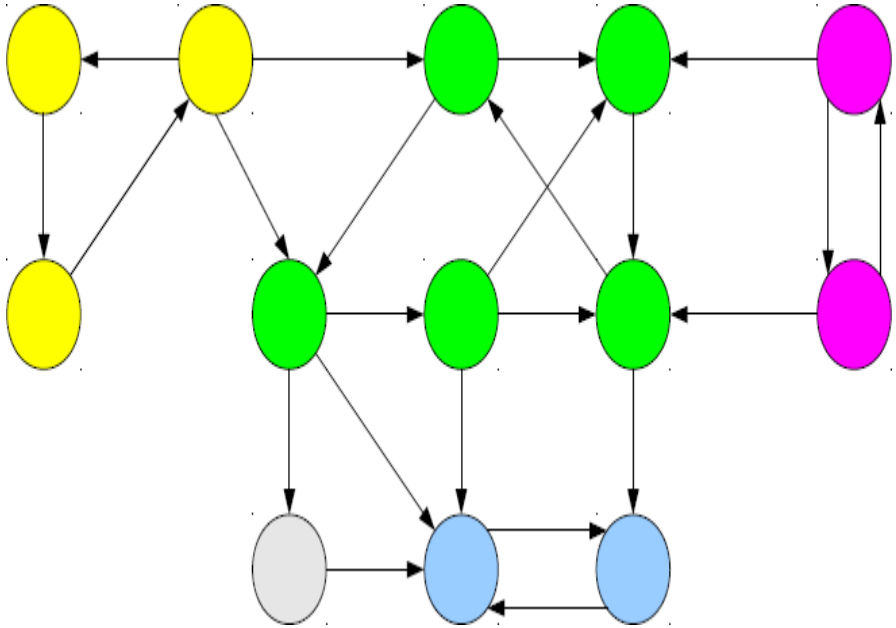
- McCabe's notion of Essential Complexity deals with the extent to which a program violates the precepts of Structured Programming.
- To find Essential Complexity of a program graph,
  - Identify a group of source statements that corresponds to one of the basic Structured Programming constructs.
  - Condense that group of statements into a separate node (with a new name)
  - Continue until no more Structured Programming constructs can be found.

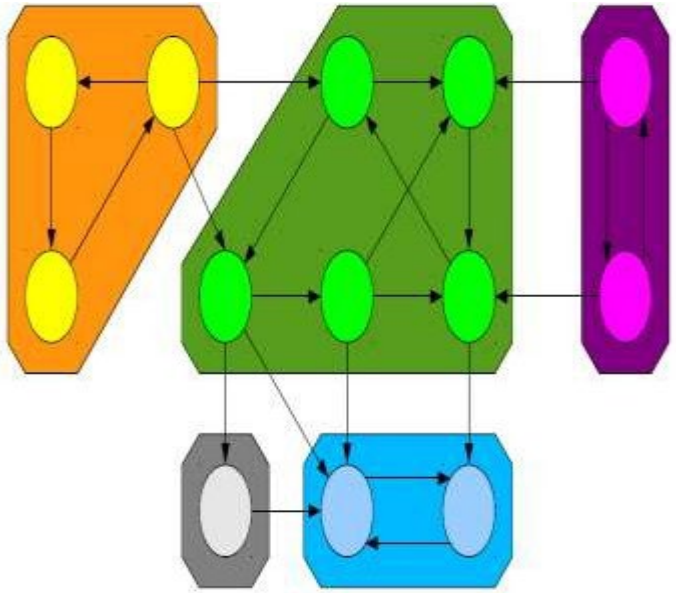
- The essential complexity of a Structured Program is 1.
- Violations of the precepts of Structured Programming increase the essential complexity.

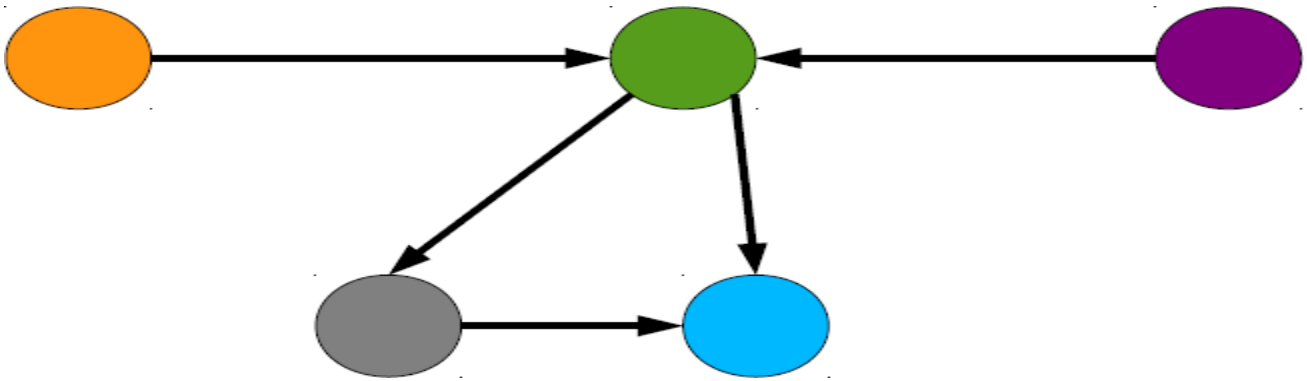
STRUCTURED PROGRAMMING CONSTRUCTS



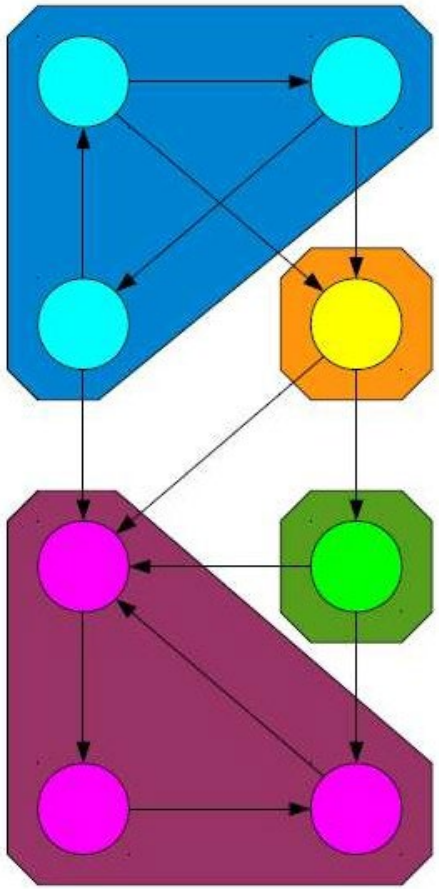
CONDENSATION GRAPH - EXAMPLE

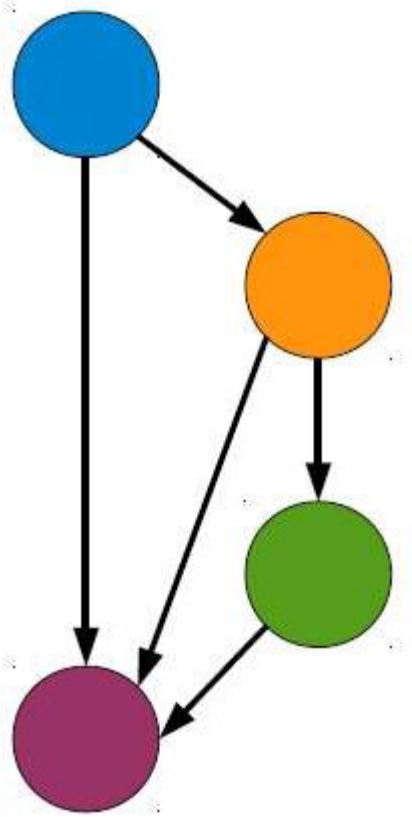




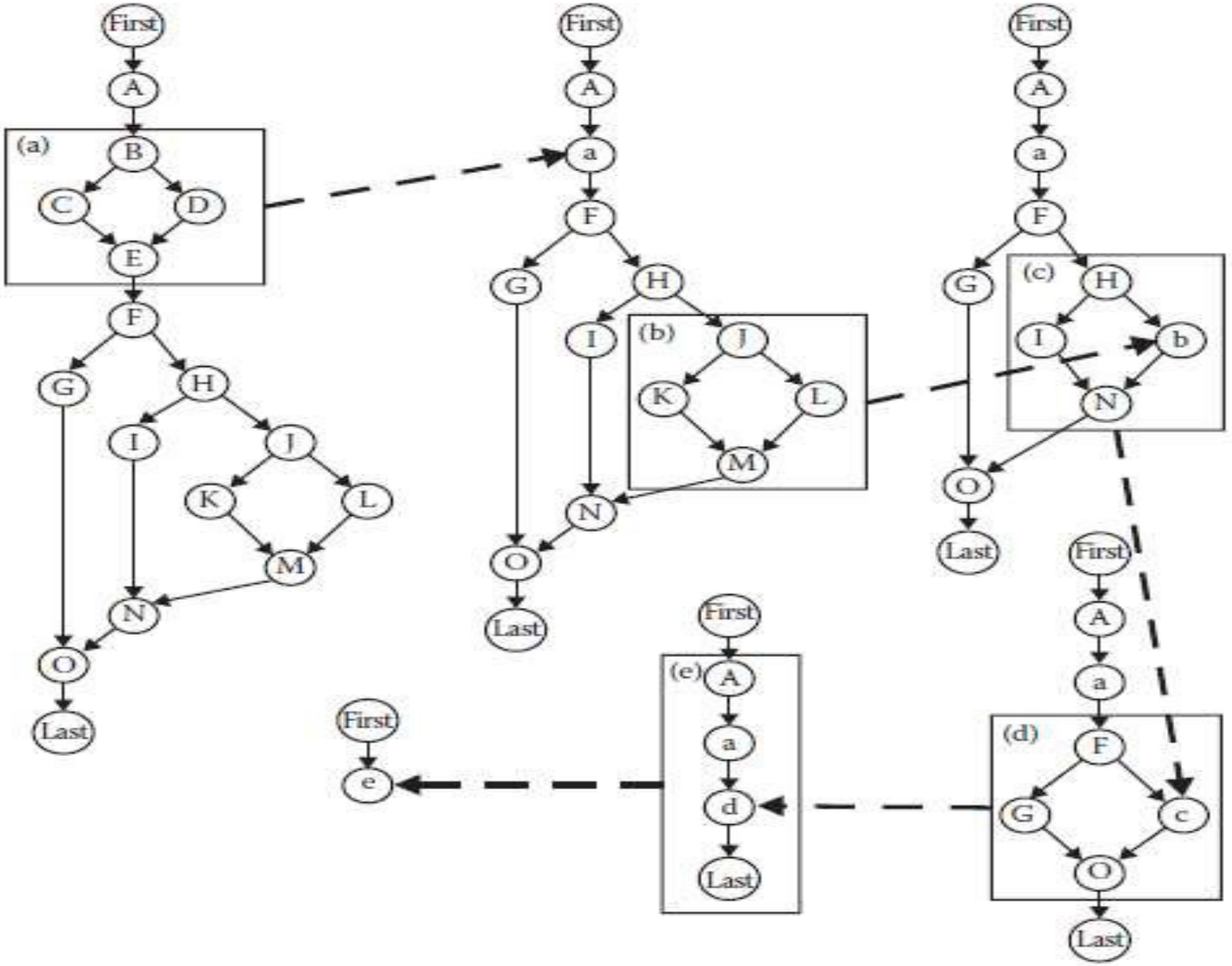


CONDENSATION GRAPH - EXAMPLE

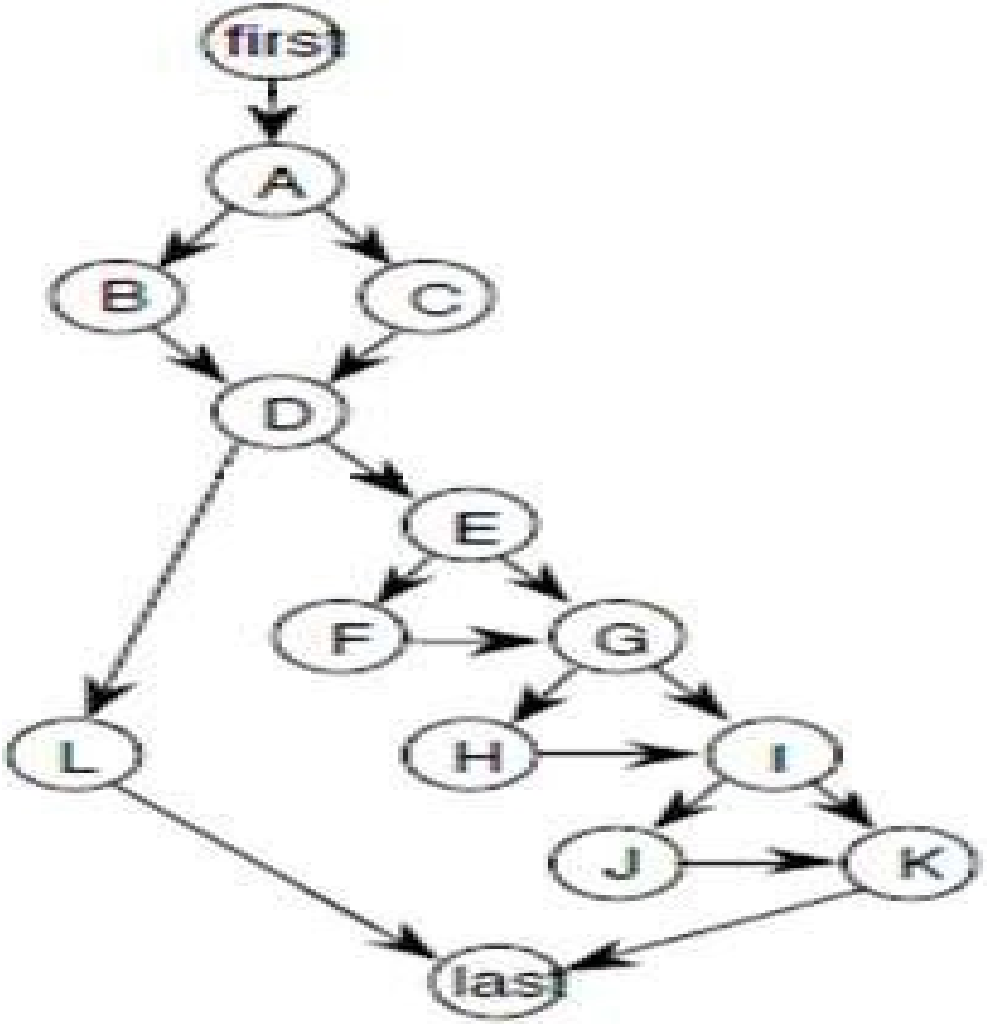




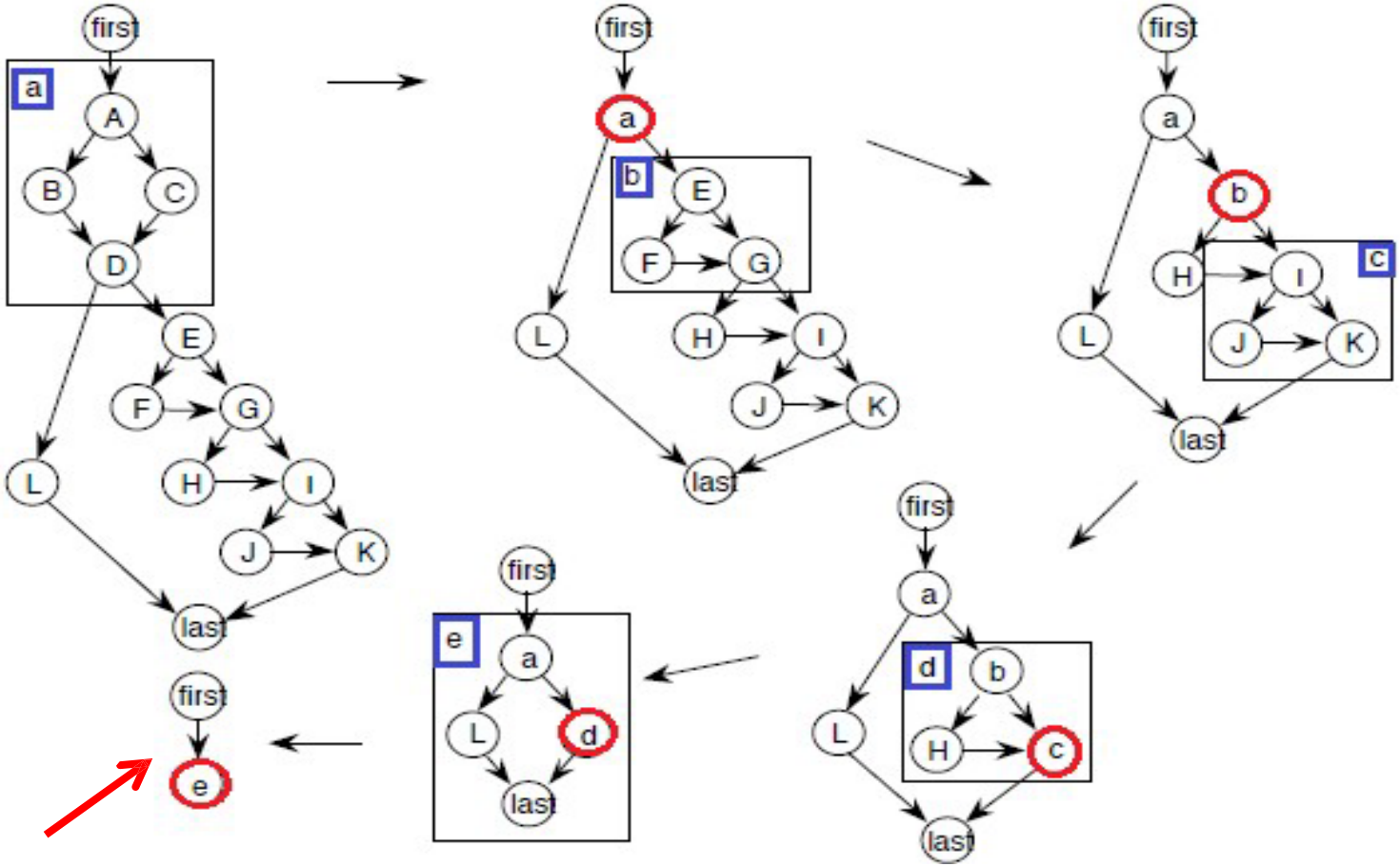
CONDENSING WITH RESPECT TO THE STRUCTURED PROGRAMMING CONSTRUCTS



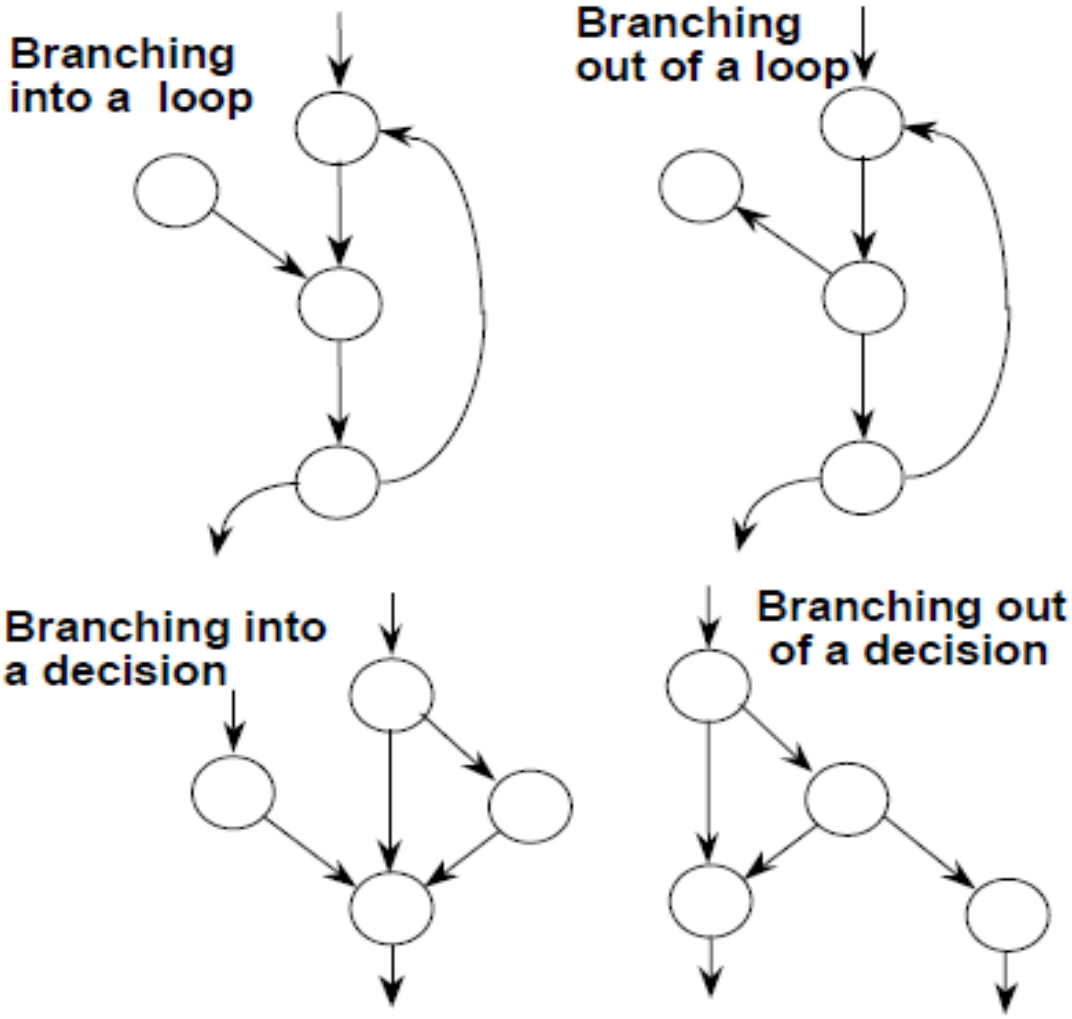
EXAMPLE GRAPH



Condensation with Structured Programming Constructs



Violations of Structured Programming Precepts



LIMITATIONS OF PATH TESTING

- Path testing as a testing technique is limited:

–Interface mismatches and mistakes are not caught

  - Not all initialization mistakes are caught by path testing
  - Specification mistakes are not caught

WHAT NEXT???

DATA FLOW TESTING

SAMPLE QUESTIONS

- Write a structured triangle program, draw the program graph and find the DD paths, DD path graph for the triangle program.
- For the program graph G(P) and a set of program variable V define the following:

  i) Defining node of variable

  ii)Usage node of variable

  iii)Definition use path with respect to variable

  iv)Definition clear path with respect to variable
- Explain about du-path test coverage metricswith data flow diagram.

DATA FLOW TESTING

- Form of structural testing
- Focuses on the points at which:
    - Variables receive values and
    - At which these values are used

DATA FLOW TESTING Cont…

- Data flow analyses centered on a **set of faults** that are known as **define/reference anomalies**
  - A variable that is defined but never used (referenced)
  - A variable that is used but never defined
  - A variable that is defined twice before it is used

- Data flow testing detects improper use of data values due to coding errors

TYPES OF DATA FLOW TESTING

- Identify potential defects commonly known to data flow anomalies

- **STATIC DATA FLOW TESTING**
  - Analyze source code
  - Do not execute source code

- **DYNAMIC DATA FLOW TESTING**
  - Involve actual program execution
  - Similar to control flow testing
    - Identify paths to execute
    - Paths are identified based on data flow testing criteria

FORMS OF DATA FLOW TESTING

- We will study **two forms of data flow testing**
  - Define/Use Testing
  - Program Slicing

TERMINOLOGY

- P → Code under test (Program)
- G(P) → Control Flow Graph of P,
  - G(P) = (V, E, s, f) (Vertices, Edges, start node, finish node)
- Path is a sequence of vertices: $v_0, v_1 \ldots, v_k$
- v is a variable of P

DEFINE / USE TESTING DEFINITION

- Program: P
- Program Graph: G(P)
  - Statement fragments(or entire statements) are nodes
  - Edges represent node sequence
  - G(P) has a single entry node & single exit node
  - There are no edges from a node to itself
- Program Variables: V

In Data Flow Testing (DFT) we are interested in the "*dependencies" among data or "relationships" among data*

VARIABLES IN STATEMENT

- Variables occur in statements as
  - **Definition – def**
    - Variables on the left side of a statement
    - via 1) initialization, 2) input, or 3) some assignment.
      - Integer X; (compiler initializes X to 0 or it will be "trash")
      - X = 3;
      - Input X;
  - **Usage**
  - **Variable is used in a computation – c-use**
    - Variables on the right-hand side of a statement (c)
    - temp := c + 5;
    - Variables used in a write statement
  - **Variable is used in a predicate – p-use**
    - if(c!= MAXLENGTH) {
    - If (X > 0) then

DATA FLOW TESTING STRATEGIES

Definition:

- Node n $\in$G(P) is a ***defining node*** of the variable v $\in$ V, written as DEF(v,n), iff the value of the variable v is defined at the statement fragment corresponding to node n.
  - For example: input , assignment, loop control statements (for int i=0;i<10;i++) and procedure calls are defining nodes
  - When the code corresponding to such statements executes, contents of the memory location associated with v is changed

Defining node, DEF(v,n), is a node, n, in the program graph where the specific variable, v, is *defined or given its value (value assignment)*.

DATA FLOW TESTING STRATEGIES Cont…

- Node n ∈G(P) is a ***usage node*** of the variable v ∈ V, written as USE(v,n), iff the value of the variable v is used at the statement fragment corresponding to node n.
  - For example: output , assignment(i:=i+1), condition, loop control statements and procedure calls are usage nodes
  - When the code corresponding to such statements executes, contents of the memory location associated with v is not changed

Usage node, USE(v,n), is a node, n, in the program graph where the specific variable, v, *is used*.

DATA FLOW TESTING STRATEGIES Cont…

- A usage node USE(v,n) is a ***predicate use*** (denoted as P-use), iff the statement n is a predicate statement; otherwise USE(v,n) is a ***computation use***, (denoted C-use)
  - Nodes corresponding to predicate uses always have an outdegree ≥ 2
  - Nodes corresponding to computation uses always have outdegree ≤ 1

A P-use node is a usage node where the variable, v, is used as a predicate (or for a branch-decision-making).

A C-use node is any usage node that is not P-used.

DATA FLOW TESTING STRATEGIES Cont…

- A *definition-use (sub) path* with respect to a variable v (denoted du-path) is a (sub) path in PATHS(P) such that for some v ∈ V, there are define and usage nodes DEF(v,m) & USE(v,n) such that m & n are the initial and final nodes of the (sub) path.

A Definition-Use path, *du-path*, for a specific variable, v, is a path where DEF(v,x) and USE(v,y) are the initial and the end nodes of that path.

DATA FLOW TESTING STRATEGIES Cont…

- A *definition-clear (sub) path* with respect to a variable v (denoted dc-path) is a definition-use(sub) path in PATHS(P) with initial and final nodes DEF(v,m) & USE(v,n) such that no other node in the (sub) path is a defining node of v

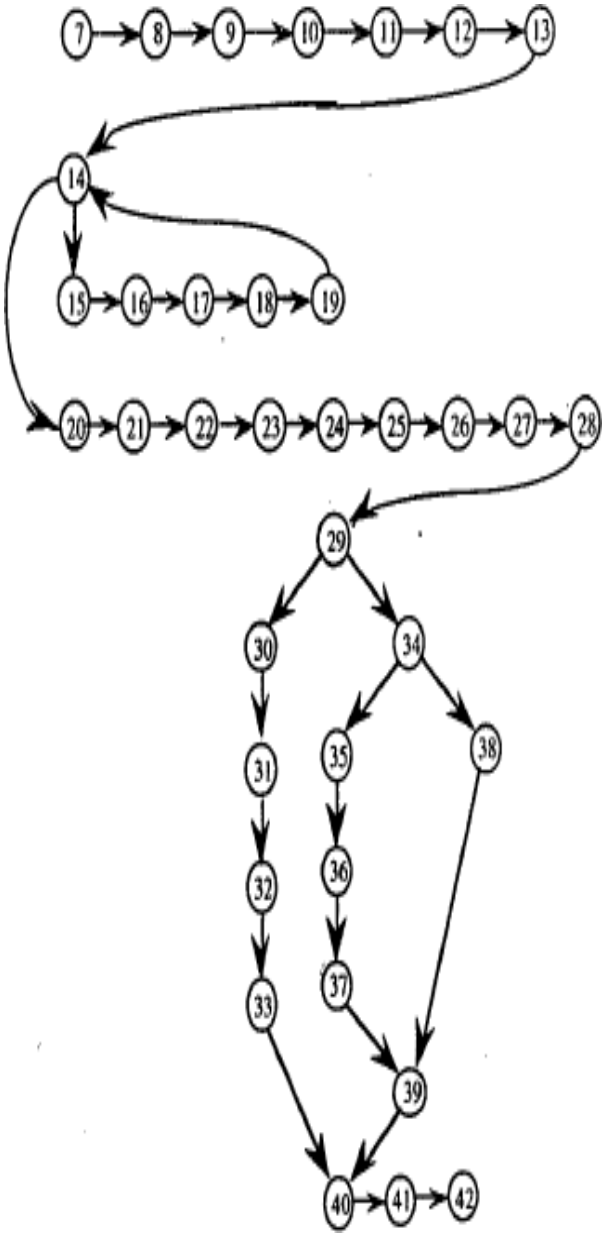
A Definition-Clear path for a specific variable, v, is a Definition-Use path with DEF(v,m) and USE(v,n) such that there is no other node in the path that is a defining node of v. (*e.g.* *v does not get reassigned in the path.)*

THE COMMISSION PROBLEM

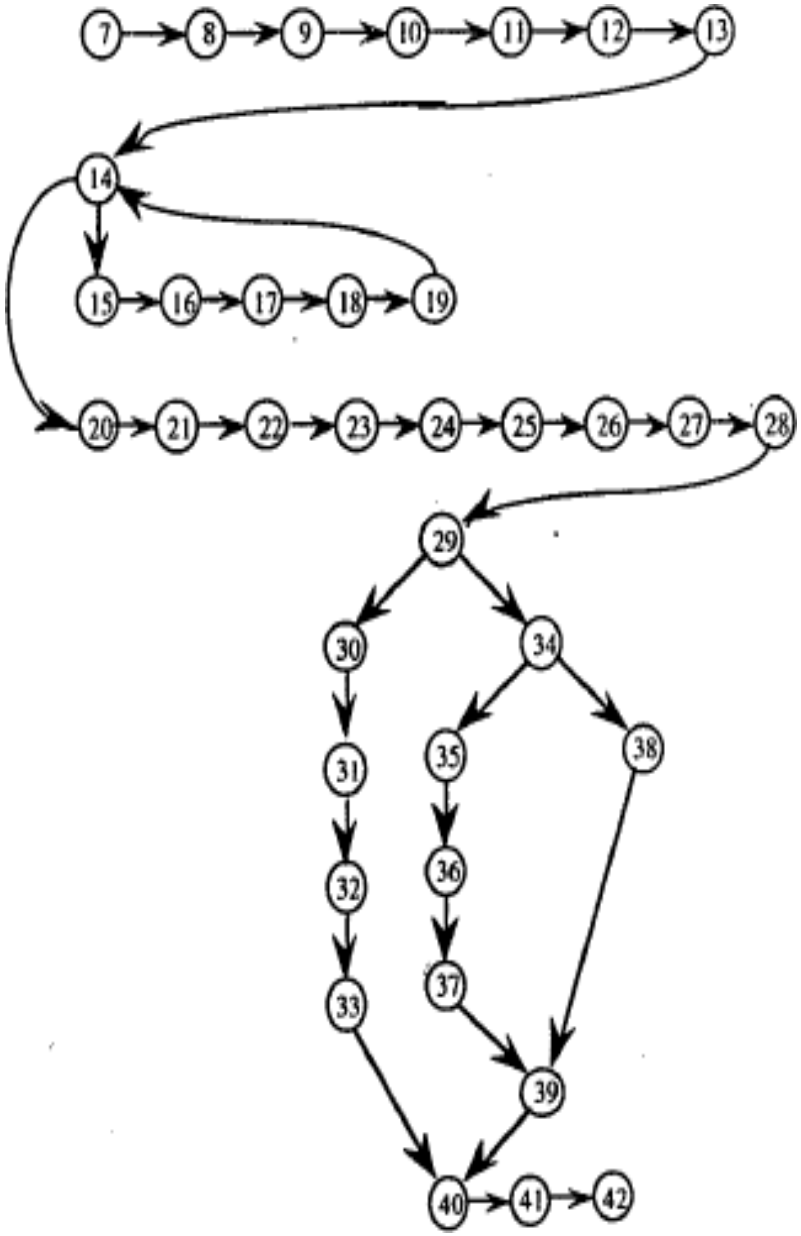
DATA FLOW TESTING STEPS

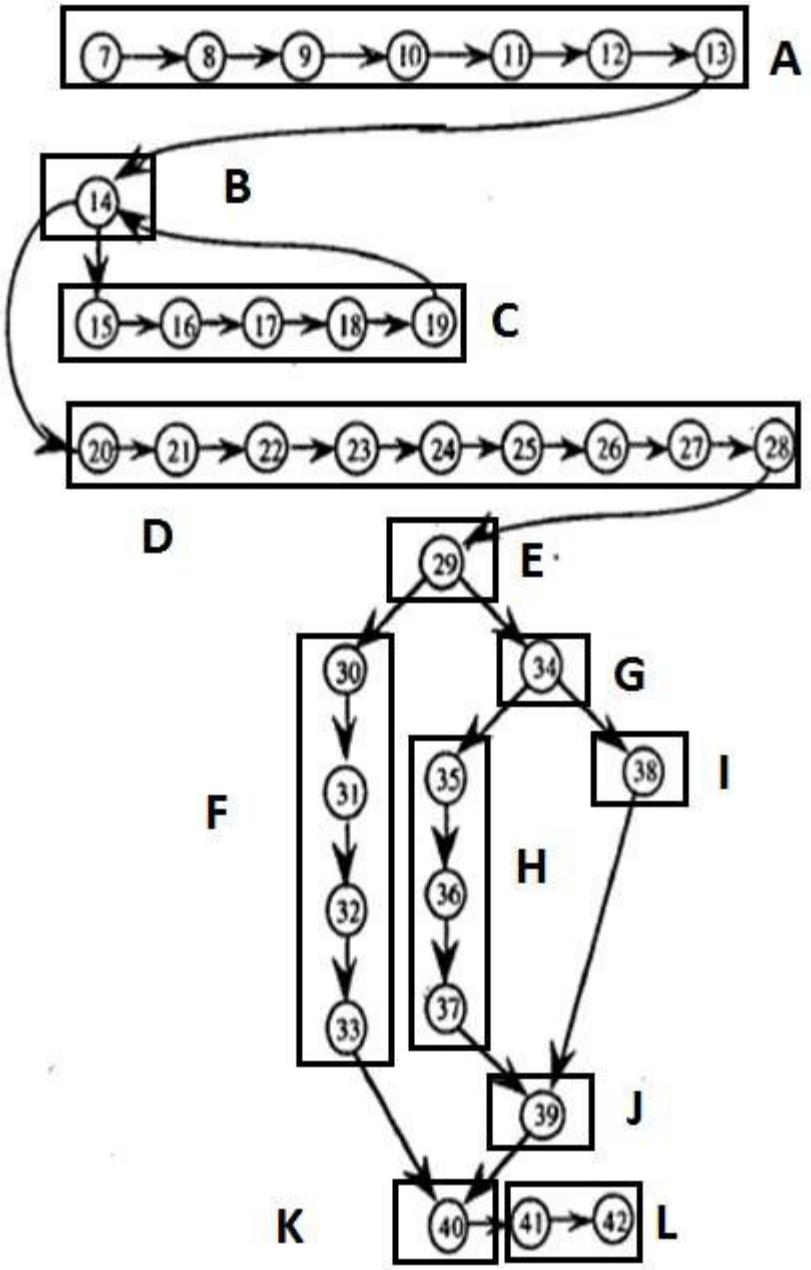
- Given a code (program or pseudo-code), the steps are:
  1. Number the lines and draw the program graph
  2. Draw the DD Path Graph
  3. List the variables
  4. List occurrences and assign a category to each variable
  5. Identify du-pairs and their use (p- or c-)
  6. Define test cases, depending on the required coverage

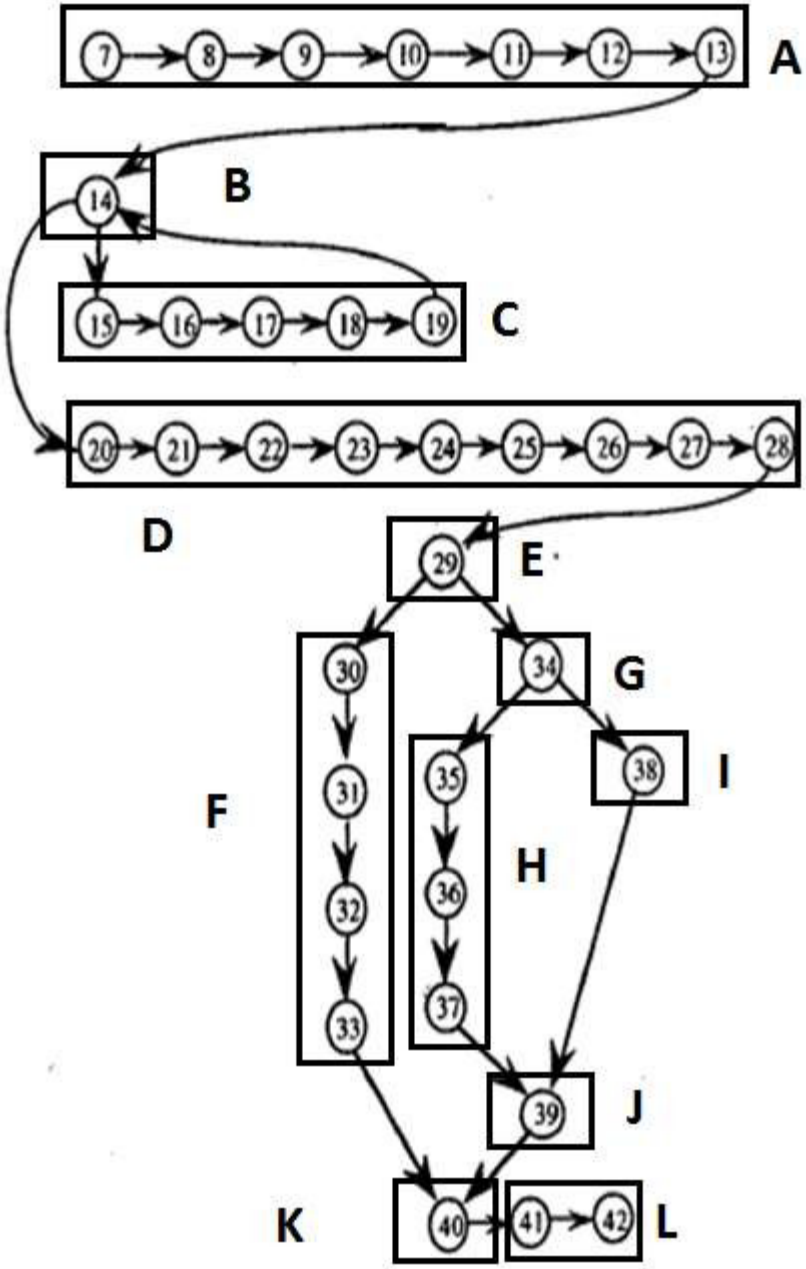
```
1   Program Commission (INPUT,OUTPUT)
2   Dim locks, stocks, barrels As Integer
3   Dim lockPrice, stockPrice, barrelPrice As Real
4   Dim totalLocks, totalStocks, totalBarrels As Integer
5   Dim lockSales, stockSales, barrelSales As Real
6   Dim sales, commission As Real
7   lockPrice = 45.0
8   stockPrice = 30.0
9   barrelPrice = 25.0
10  totalBarrels = 0
11  totalStocks = 0
12  totalBarrels = 0
13  Input(locks)
14  While NOT(locks = −1)  "locks = −1 signals end of data
15    Input(stocks, barrels)
16    totalLocks = totalLocks + locks
17    totalStocks = totalStocks + stocks
18    totalBarrels = totalBarrels + barrels
19    Input(locks)
20  EndWhile
21  Output("Locks sold:," totalLocks)
22  Output("Stocks sold:," totalStocks)
23  Output("Barrels sold:," totalBarrels)
24  lockSales = lockPrice*totalLocks
25  stockSales = stockPrice*totalStocks
26  barrelsSales = barrelPrice * totalBarrels
27  sales = lockSales + stockSales + barrelSales
28  Output("Total sales: ", sales)
29  If (sales > 1800.0)
30    Then
31      commission = 0.10 * 1000.0
32      commission = commission + 0.15 * 800.0
33      commission = commission + 0.20*(sales−1800.0)
34    Else If (sales > 1000.0)
35      Then
36      commission = 0.10 * 1000.0
37      commission = commission + 0.15*(sales−1000.0)
38      Else
39      commission = 0.10 * sales
40    EndIf
41  EndIf
42  Output("Commission is $", commission)
43  End Commission
```



Step 2: Derive DD Path Graph

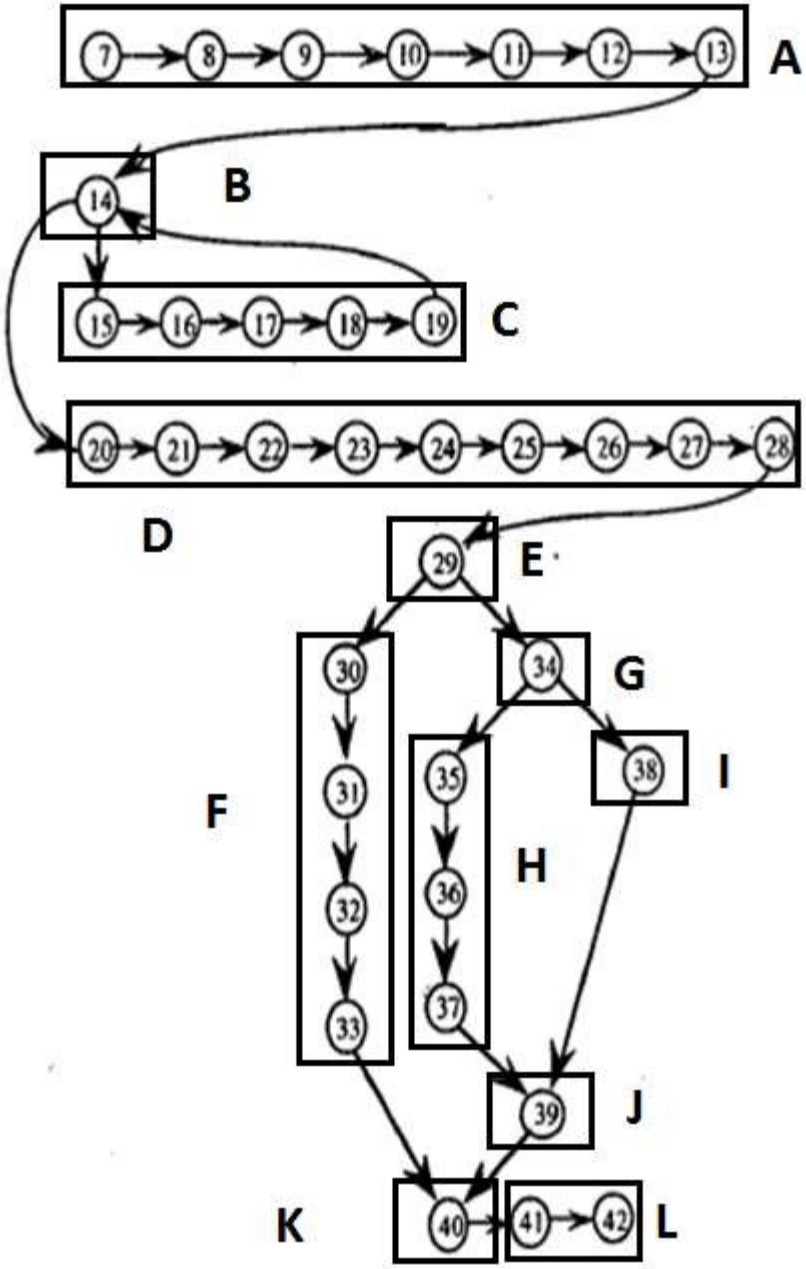

7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

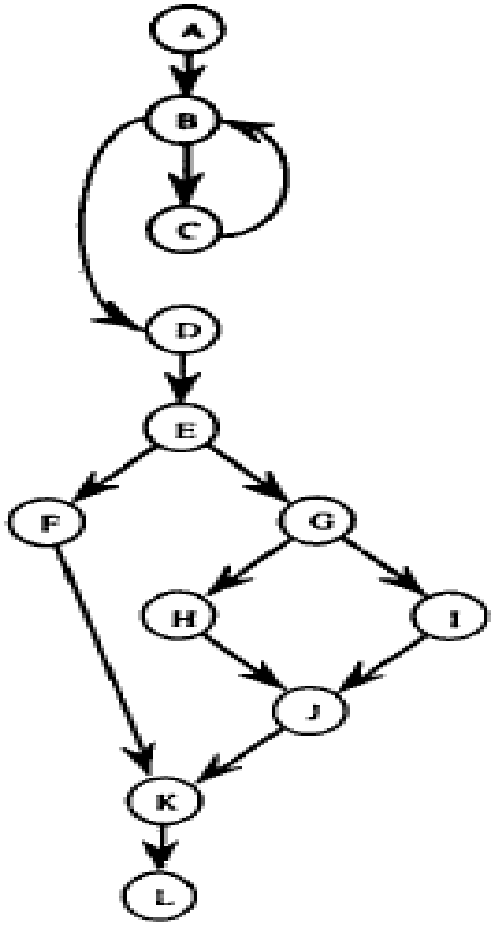

A
B
C
D
E
F
G
H
I
J
K
L



DD-Paths in Figure

DD-Path	*Nodes*
A	7, 8, 9, 10, 11, 12, 13
B	14
C	15, 16, 17, 18, 19
D	20, 21, 22, 23, 24, 25, 26, 27, 28
E	29
F	30, 31, 32, 33
G	34
H	35, 36, 37
I	38
J	39
K	40
L	41, 42


7
8
9
10
11
12
13
A
14
B
15
16
17
18
19
C
20
21
22
23
24
25
26
27
28
D
29
E
30
31
32
33
F
34
G
35
36
37
H
38
I
39
J
40
41
42
K
L

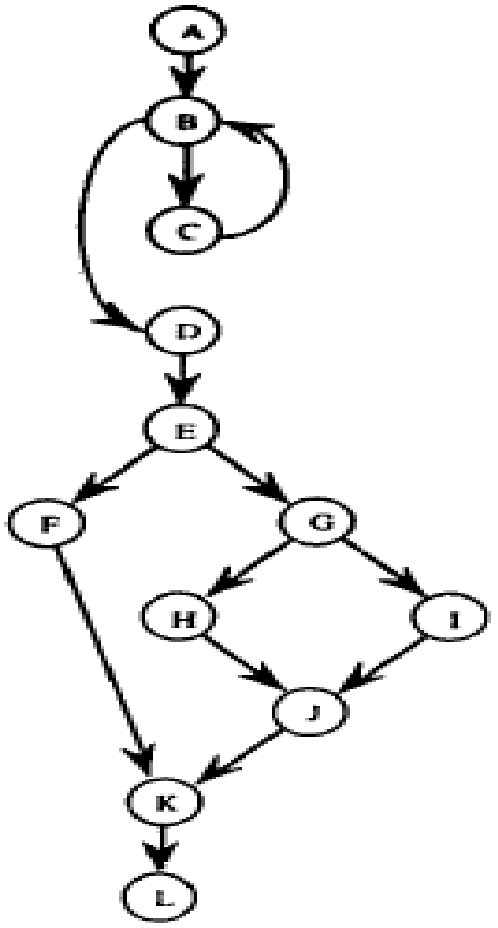

A
B
C
D
E
F
G
H
I
J
K
L

DD – PATH FROM PROGRAM GRAPH

Table 10.1 DD-Paths in Figure 10.1

DD-Path	Nodes
A	7, 8, 9, 10, 11, 12, 13
B	14
C	15, 16, 17, 18, 19
D	20, 21, 22, 23, 24, 25, 26, 27, 28
E	29
F	30, 31, 32, 33
G	34
H	35, 36, 37
I	38
J	39
K	40
L	41, 42

DD-PATH GRAPH OF THE COMMISSION PROGRAM



STEP 3: LIST THE VARIABLES USED

lock
stock
barrels

lockSales
stockSales
barrelSales

lockPrice
stockPrice
barrelPrice

sales
commission

totalLocks
totalStocks
totalBarrels

Step 4:

List occurrences and assign a category to each variable

DEFINE/USE NODES FOR VARIABLES

```
1  Program Commission (INPUT,OUTPUT)
2  Dim locks, stocks, barrels As Integer
3  Dim lockPrice, stockPrice, barrelPrice As Real
4  Dim totalLocks, totalStocks, totalBarrels As Integer
5  Dim lockSales, stockSales, barrelSales As Real
6  Dim sales, commission As Real
7  lockPrice = 45.0
8  stockPrice = 30.0
9  barrelPrice = 25.0
10 totalBarrels = 0
11 totalStocks = 0
12 totalBarrels = 0
13 Input(locks)
14 While NOT(locks = −1)  "locks = −1 signals end of data
15   Input(stocks, barrels)
16   totalLocks = totalLocks + locks
17   totalStocks = totalStocks + stocks
18   totalBarrels = totalBarrels + barrels
19   Input(locks)
20 EndWhile
21 Output("Locks sold:," totalLocks)
22 Output("Stocks sold:," totalStocks)
23 Output("Barrels sold:," totalBarrels)
24 lockSales = lockPrice*totalLocks
25 stockSales = stockPrice*totalStocks
26 barrelsSales = barrelPrice * totalBarrels
27 sales = lockSales + stockSales + barrelSales
28 Output("Total sales: ", sales)
29 If (sales > 1800.0)
30   Then
31     commission = 0.10 * 1000.0
32     commission = commission + 0.15 * 800.0
33     commission = commission + 0.20*(sales−1800.0)
34   Else If (sales > 1000.0)
35     Then
36     commission = 0.10 * 1000.0
37     commission = commission + 0.15*(sales−1000.0)
38     Else
39     commission = 0.10 * sales
40   EndIf
41 EndIf
42 Output("Commission is $", commission)
43 End Commission
```

```
7  lockPrice = 45.0
8  stockPrice = 30.0
9  barrelPrice = 25.0
10 totalBarrels = 0
11 totalStocks = 0
12 totalBarrels = 0
13 Input(locks)
14 While NOT(locks = –1)  "locks = –1 signals end of data
15   Input(stocks, barrels)
16   totalLocks = totalLocks + locks
17   totalStocks = totalStocks + stocks
18   totalBarrels = totalBarrels + barrels
19   Input(locks)
20 EndWhile
21 Output("Locks sold:," totalLocks)
22 Output("Stocks sold:," totalStocks)
23 Output("Barrels sold:," totalBarrels)
24 lockSales = lockPrice*totalLocks
25 stockSales = stockPrice*totalStocks
26 barrelsSales = barrelPrice * totalBarrels
27 sales = lockSales + stockSales + barrelSales
28 Output("Total sales: ", sales)
29 If (sales > 1800.0)
30   Then
31     commission = 0.10 * 1000.0
32     commission = commission + 0.15 * 800.0
33     commission = commission + 0.20*(sales–1800.0)
34   Else If (sales > 1000.0)
35     Then
36     commission = 0.10 * 1000.0
37     commission = commission + 0.15*(sales–1000.0)
38     Else
39     commission = 0.10 * sales
40   EndIf
41 EndIf
42 Output("Commission is $", commission)
43 End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

DEFINE/USE NODE FOR THE VARIABLES IN THE COMMISSION PROGRAM

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                  commission = 0.10 * 1000.0
37                  commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)
24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)
24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)   'loop condition uses -1 to indicate end of data
15          totalBarrels = totalBarrels + barrels
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)    'Loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While locks <> -1      'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                  commission = 0.10 * 1000.0
37                  commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)
24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice*totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)     'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                      commission = 0.10 * 1000.0
37                      commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              EndIf
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real
4       Dim totalLocks, totalStocks, totalBarrels As Integer
5       Dim lockSales, stockSales, barrelSales As Real
6       Dim sales, commission As Real

7       lockPrice = 45.0
8       stockPrice = 30.0
9       barrelPrice = 25.0
10      totalLocks = 0
11      totalStocks = 0
12      totalBarrels = 0

13      Input(locks)
14      While NOT(locks = -1)      'loop condition uses -1 to indicate end of data
15          Input(stocks, barrels)
16          totalLocks = totalLocks + locks
17          totalStocks = totalStocks + stocks
18          totalBarrels = totalBarrels + barrels
19          Input(locks)
20      EndWhile

21      Output("Locks sold: ", totalLocks)
22      Output("Stocks sold: ", totalStocks)
23      Output("Barrels sold: ", totalBarrels)

24      lockSales = lockPrice*totalLocks
25      stockSales = stockPrice*totalStocks
26      barrelSales = barrelPrice * totalBarrels
27      sales = lockSales + stockSales + barrelSales
28      Output("Total sales: ", sales)

29      If (sales > 1800.0)
30          Then
31              commission = 0.10 * 1000.0
32              commission = commission + 0.15 * 800.0
33          commission = commission + 0.20*(sales-1800.0)
34      Else If (sales > 1000.0)
35          Then
36                  commission = 0.10 * 1000.0
37                  commission = commission + 0.15*(sales-1000.0)
38              Else commission = 0.10 * sales
39              Endif
40      EndIf

41      Output("Commission is $", commission)
42      End Commission
```

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

Variable	Defined at Node	Used at Node
Lock price	7	24
Stock price	8	25
Barrel price	9	26
Total Locks	10,16	16,21,24
Total Stocks	11,17	17,22,25
Total Barrels	12,18	18,23,26
Locks	13,19	14,16
Stocks	15	17
Barrels	15	18
LockSales	24	27
StockSales	25	27
BarrelsSales	26	27
Sales	27	28,29,33,34,37,39
commission	31,32,33,36,37,39	32,33,37,42

STEP 5
IDENTIFY DU-PAIR AND THEIR USE (P-USE AND C-USE)

- A ***definition-clear*** (***sub***) ***path*** with respect to a variable v (denoted dc-path) is a definition-use(sub) path in PATHS(P) with initial and final nodes DEF(v,m) & USE(v,n) such that no other node in the (sub) path is a defining node of v

(e.g. v does not get reassigned in the path.)

Selected Define/Use paths

Variable	Path (Beginning,End Nodes)	Definition-Clear?
lockprice	7,24	Yes
stockprice	8,25	Yes
barrelprice	9,26	Yes
totalStocks	11,17	Yes
totalStocks	11,22	No
totalStocks	11,25	No
totalStocks	17,17	Yes
totalStocks	17,22	No
totalStocks	17,25	No
locks	13,14	Yes
locks	13,16	Yes
locks	19,14	Yes
locks	19,16	Yes
sales	27,28	Yes
sales	27,29	Yes
sales	27,33	Yes
sales	27,34	Yes
sales	27,37	Yes
sales	27,39	Yes

IDENTIFYING DU-PATHS IN COMMISSION PROBLEM

- Table shows the du-paths in the commission problem
- They are named by their beginning and ending nodes (from program graph of the commission program)
- The third column in the table indicates whether the du-paths are definition-clear
- The while loop (node sequence <14, 15, 16, 17, 18, 19, 20>) inputs and accumulates values for totalLocks, totalStocks, totalBarrels
- The initial value definition for totalStocks occurs at node 11, and it is first used at node 17
- Thus the path (11,17), which consists of the node sequence <11, 12, 13, 14, 15, 16, 17> is definition clear
- The path (11, 17, 22) which consists of the node sequence <11, 12, 13,(14,15,16,17,18,19,20)*,21,22> is not definition clear because values of totalStocks are defined at node 11 and possible several times at node 17.
- (The asterisk after while loop is the kleene star notation used in both formal logic and regular expression to denote zero or more repetitions)

du- PATHS FOR STOCKS

Variable *stocks*:

- We have DEF(stocks,15) & USE (stocks,17)
  - Hence path <15,17> is a du-path with respect to stocks
  - p0 = <25,27> is also decision-clear, a dc-path

A Simple Path Segment is a path segment in which at most one node is visited twice. – E.g., (7,4,5,6,7) is simple. • Therefore, a simple path may or may not be loop-free.

du-PATHS FOR LOCKS

Variable ***locks***:

- We have DEF(locks,13), DEF(locks,19), USE(locks,14) & USE(locks,16): 4 du-paths(dc too!)
  - p1 = <13,14>
  - p2 = <13,14,15,16>
  - p3 = <19,20,14>
  - p4 = < 19,20,14,15,16>
  - We could extend p1 and p3 to include node 21:
  - p1` = <13,14,21> & p3` = <19,20,14,21>
    - Then p1`,p2,p3` & p4 form a very complete set of test cases for the while loop
      - » Bypass the loop
      - » Begin the loop
      - » Repeat the loop
      - » Exit the loop

du-PATHS FOR LOCKS cont…

- Du-paths:
  - P1 and P2 refer to priming value of locks, which is read at node 13;
- Locks has a predicate use in the while statement (node 14) and if condition is true (as in path p2), a computation use at statement 16
- Other two du-paths start near the end of the while loop and occur when the loops repeats
- These 4 paths provide the loop coverage:
  - Bypass the loop,
  - Begin the loop
  - Repeat the loop and
  - Exit the loop
  - All these du-paths are definition-clear

du-PATHS FOR totalLocks

- The du-paths for the totalLocks will lead us to typical test cases for computations
- With two defining nodes:
  - (DEF(totalLocks, 10) and DEF(totalLocks, 16))
- With three usage nodes:
  - (USE(totalLocks,16),(USE(totalLocks,21), (USE(totalLocks, 24)))
  - We might expect 6 du-paths
- Path p5=<10,11,12,13,14,15,16> is a du-path in which the initial value of totalLocks(0) has computation use. This path is definition-clear
- Thenext path is problematic: P6=<10,11,12,13,14,15,16,17,18,19,20,14,21>

du-PATHS for totalLocks cont…

- Path P6 ignores the possible repetition of the while loop. We could highlight this by nothing that the subpath <16,17,18,19,20,14,15> might be traversed several times. We still have a du-path which is not definition clear
- If a problem occurs with the value of totalLocks at node 21 (the output statement), we should look at the intervening DEF(totalLocks, 16) node
- The next path contains p6; we can show this by using a path name in place of its corresponding node sequence:
  - P7 = <10,11,12,13,14,151,6,17,18,19,20,14,21,22,23,24>
  - P7 = <p6,22,23,24>
- Du-path P7 is not definition-clear because it includes node 16
- Subpaths that begin with node 16 (an assignment statement) are interesting.
- The first <16,16> seems degenerated
- The remaining two du-paths are both subpaths of P7;
  - P8 = <16,17,18,19,20,14,21>
  - P9 = <16,17,18,19,20,14,21,22,23,24>
  - Both are definition-clear, and both have the loop iteration

Variable ***totallocks***:

- We have:
  - DEF(totallocks,10) & DEF(totallocks,16)
  - USE(totallocks,16), USE(totallocks,21) & USE(totallocks,24)
  - We could expect 6 du-path, why?
  - There are only 5:
    - p5 = <10,11,12,13,14,15,16> (also a dc-path)
    - p6 = <10,11,12,13,14,15,16,17,18,19,20,21>
      - The subpath: <16,17,18.19.20,14,15> might be traversed several times
      - This is not a dc-path, if there is a problem with the value of totallocks at node 21, we should look at the defining node, 16
    - p7 = < **10,11,12,13,14,15,16,17,18,** 19,20,21,22,23,24>
      - p7 = <p6, 22,23,24 >, not a dc-path
    - <16,16> is degenerate, disregard it
    - p8 = < 16,17,18,19,20,21> this is a dc-path
    - p9 = < 16,17,18,19,20,21,23,24> this is a dc-path
  - p8 & p9 have the same loop iteration problem as p6

Du-PATH FOR VARIABLE SALES

Variable ***sales***:

- DEF(sales,27), USE(sales,28), USE(sales,29), USE(sales,33), USE(sales,34) , USE(sales,37) , USE(sales,38)
  - p10 = <27,28>
  - p11 = <27,28,29>
  - p12 = < 27,28,29, 30,31,32,33>
    - » This is a dc-path covering p10 & p11
    - » If we test with p12, we will cover p10 & p11 too
  - p13 = < 27,28,29,34>
  - p14 = < 27,28,29,34,35,36,37>
  - p15 = < 27,28,29,38>

Du-PATH FOR THE VARIABLE COMMISSION

Variable ***commission***:

- Since 31,32 & 33 could be replaced by:
  - Commission := 220 + 0.2 * (sales - 1800), then 33 could be considered the defining node
- Same for 36,37 could be considered the defining node

– DEF(commission, 33), DEF(commission, 37), DEF(commission, 38), USE(commission, 41)
  - p16 = <33,41> (a dc-path)
  - p17 = <37,41> (a dc-path)
  - p18 = <38,41> (a dc-path)

Define/Use paths for Commission

Variable	Path(Beginning,End) Nodes	Feasible?	Definition-Clear?
commission	31,32	Yes	Yes
commission	31,33	Yes	No
commission	31,37	No	n/a
commission	31,42	Yes	No
commission	32,32	Yes	Yes
commission	32,33	Yes	Yes
commission	32,37	No	n/a
commission	32,42	Yes	No
commission	33,32	No	n/a
commission	33,33	Yes	Yes
commission	33,37	No	n/a
commission	33,42	Yes	Yes
commission	36,32	No	n/a
commission	36,33	No	n/a
commission	36,37	Yes	Yes

commission	36,42	Yes	No
commission	37,32	No	n/a
commission	37,33	No	n/a
commission	37,37	Yes	Yes
commission	37,42	Yes	Yes
commission	39,32	No	n/a
commission	39,33	No	n/a
commission	39,37	No	n/a
commission	39,42	Yes	Yes

DU-PATH TEST COVERAGE METRICS

SAMPLE QUESTIONS

- Describe definition-use pairs with a suitable example.
- Illustrate the concept of define/use testing with suitable code.
- Explain about du-path test coverage metrics with data flow diagram.
- Define the various ***data flow testing criteria***.
- With suitable example, explain use testing and slice based testing.
- Explain about slice based testing in a data flow testing.

- The metrics – a set of criteria, essentially
  - allow the tester to select sets of paths through the program,
  - where
    - the number of paths selected is ***always finite***, and
    - chosen in a ***systematic*** and
    - ***intelligent manner*** in order to help us ***uncover errors***

The objective of finding the du-paths in a program is to define a set of test coverage metrics (known as the Rapps-Weyuker data flow metrics)

9 CRITERIA'S

1. **All-Nodes** - corresponds to '**statement coverage**', is satisfied if every node is covered by the set of paths.
2. **All-Edges** - corresponds to '**branch coverage**', is satisfied if every edge (branch) of the program graph is covered.
3. **All-Paths** - corresponds to the concept of '**path coverage**', is satisfied if every path of the program graph is covered in the set.

- In addition to these metrics, six new metrics were defined:
  4. All-Defs Criterion
  5. All-Uses Criterion
  6. All-P-Uses/Some C-Uses Criterion
  7. All-C-Uses/Some P-Uses Criterion
  8. All-DU-Paths Criterion
  9. All-P-Use Criterion

DU-PATH TEST COVERAGE METRICS

- The first three test coverage metrics are the same as those of the structural test coverage metrics: all paths, all edges and all nodes.
- The others presume that define and usage nodes have been identified for all program variables & that du-path have been identified with respect to each variable

DU-PATH TEST COVERAGE METRICS - DEFINITIONS

- In the following definitions,
  - T (Test) → Set of (sub) paths in the program graph G(P) of a program P, with set V of variables
  - We assume that the define/use paths are all feasible

Definition

The set T satisfies the *All-Defs* criterion for the program P iff for every variable $v \in V$, T contains definition-clear paths from every defining node of v to a use of v.

Definition

The set T satisfies the *All-Uses* criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every use of v, and to the successor node of each USE(v,n).

Definition

The set T satisfies the *All-P-Uses /Some C-Uses* criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every predicate use of v, and if a definition of v has no P-uses, there is a definition-clear path to at least one computation use.

Definition

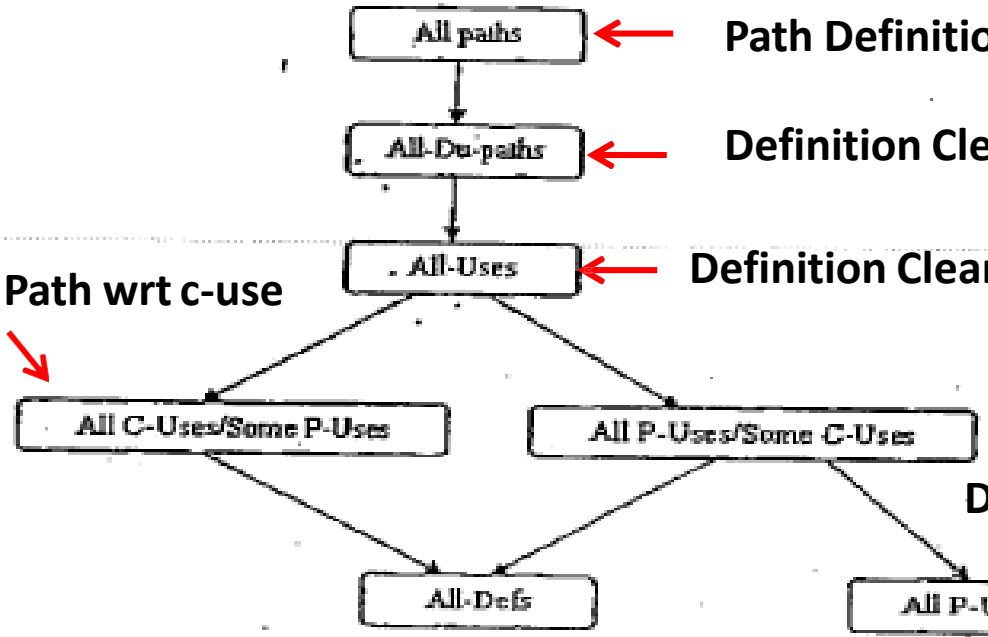
The set T satisfies the *All-C-Uses /Some P-Uses* criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every computation use of v, and if a definition of v has no C-uses, there is a definition-clear path to at least one predicate use.

Definition

The set T satisfies the *All-DU-paths* criterion for the program P iff for every variable $v \in V$, T contains definition-clear paths from every defining node of v to every use of v, and to the successor node of each

These test coverage metrics have several set-theory based relationships, which are referred to as subsumption. These relationships are shown in Figurenbelow.

Rapps-Weyuker hierarchy of dataflow coverage metrics.



Path Definition

Definition Clear Path

Definition Clear Path wrt USE(v,n)

Definition Clear Path wrt c-use

DC Path wrt p-use

All DEF(v,n) with dc-path

Branch Coverage

Statement Coverage

PROGRAM SLICING

Program Slices

- A concept related to dataflow analysis (def-use path) is to look at a slice of program that is related to some "variable of interest" and trace the program statements that are in the program slice. (Program slice was first discussed by Mark Weiser in the 1980's)
  - Tracing program statements that contribute or affect the value of the variable of interest at some point of the program; { e.g. (v,node) } going "backward" to include all those statements that have ***affected*** the variable's value is considered a slicing of the program with respect to that variable.
  - The designated program slice then becomes a path that one would consider for testing.
- This is also a popular, perhaps subconscious, ***debugging technique*** used by most of us.

SLICE BASED TESTING

- The second type of data flow testing

  - A program slice is a set of program statements that contribute to, or affect a value for a variable at some point in the program

- The idea of slicing is to divide a program into components that have some useful meaning

- **DEFINITION**

– Given a program P and a set V of variables in P, a slice on the variable set V at statement n, written S(V, n) is the set of all statements in **P prior to node n that contribute to the values of variables in V at node n**

SLICE BASED TESTING Cont…

- **DEFINITION**

– Given a program P, and a program graph G(P) in which statements and statement fragments are numbered, and a set V of variables in P, the ***slice on the variable set V at statement fragment n***, written S(V, n), is the set node numbers of all statement fragments in **P prior to n that contribute to the values of variables in V at statement fragment n.**

- "prior to": a slice captures the execution time behavior of a program with respect to the variable(s) in the slice
- "contribute": some declarative statements have an effect on the value of a variable (e.g. const, type)
  - We will exclude those non-executable statements

SLICE-BASED TESTING DEFINITION

- Five forms of ***usage nodes***
  - P-use (used in a predicate (decision))
  - C-use (used in computation)
  - **O-use (used for output, e.g. printf())**
  - **L-use (used for location, e.g. pointers, subscripts)**
  - **I-use (iteration, e.g. internal counters)**

- Two forms of ***definition nodes***
  - I-def (defined by input, e.g. scanf())
  - A-def (defined by assignment)

- For now, we presume that the slice S(V,n) is a slice on one variable, that is, the set V consists of a single variable, v

GUIDELINES FOR CHOOSING SLICES

- If statement fragment n in S(V, n) is a ***defining node*** for v, then ***n is included in the slice.***

- If statement fragment n is a ***usage node***, then it is ***included in the slice.***

- If a statement is both a ***defining*** and a ***usage node***, then it is ***included in the slice***.

- In a static slice, ***P-uses and C-uses*** of other variables (not the v in the slice set V) are included to the ***extend that their execution affects the value*** of the variable v.

- If the value of v is the same whether a statement fragment is included or excluded, ***exclude*** the statement fragment.

- ***O-use, L-use, and I-use*** nodes are **excluded from slices**

- Eventually we will develop a DAG (directed acyclic graph) of slices in which nodes are slices and edges correspond to the subset relationship

- **Guidelines for choosing slices**:
  - If the value of v is the same whether the statement fragment is included or excluded, exclude the fragment.
  - If statement fragment n is:
    - a defining node for v, include n in the slice
    - a usage node for v, exclude n from the slice
  - O-use, L-use & I-use nodes are excluded from slices

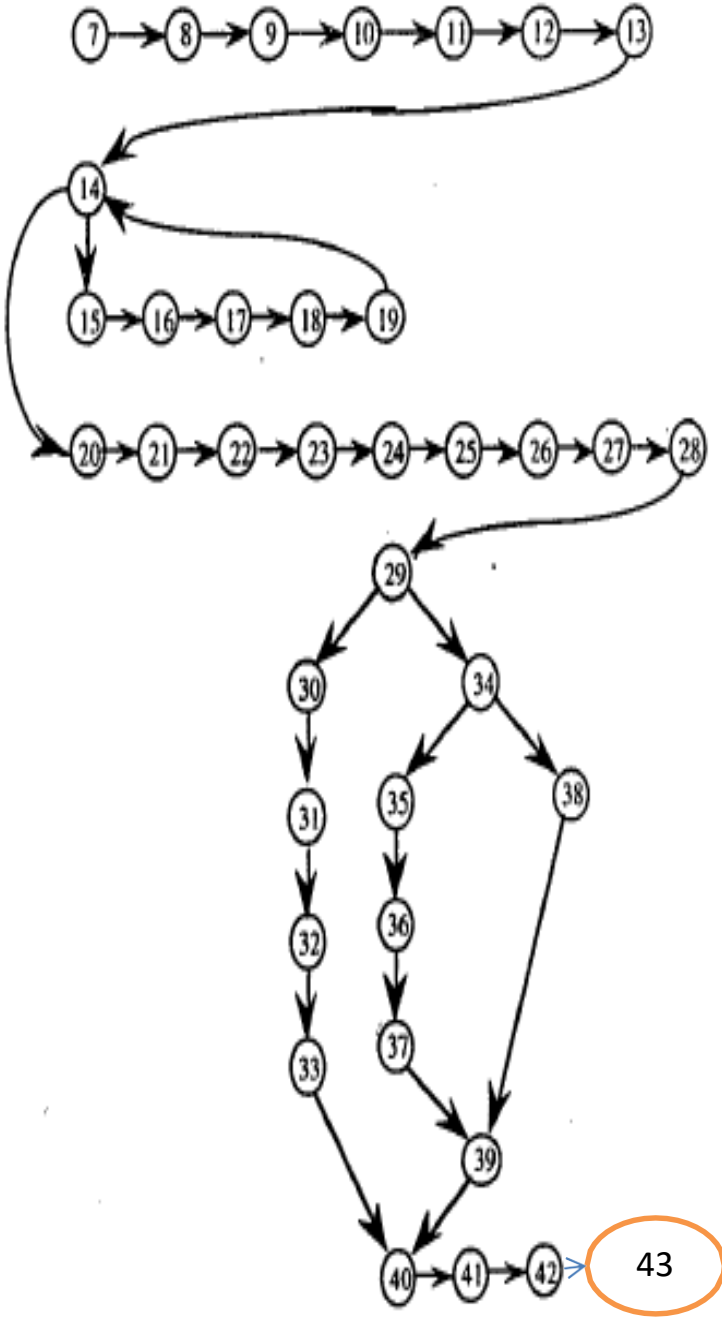
DEFINE/USE EXAMPLE
THE COMMISSION PROBL

STEP 1: NUMBERING THE LINES

```
1   Program Commission (INPUT,OUTPUT)

2       Dim locks, stocks, barrels As Integer
3       Dim lockPrice, stockPrice, barrelPrice As Real

4.  Dim totalLocks, totalStocks,totalBarrels as Integer
5.  Dim lockSales, stockSales, barrelSales As Real
6.  Dim sales, commission : REAL
7.  Lockprice = 45.0
8.  Stockprice = 30.0
9.  barrelPrice = 25.0
10. totalLocks = 0
11. totalStocks = 0
12. totalBarrels = 0
13. Input (locks)
14. While NOT (locks = -1)   //Input device uses -1 to indicate end of data
15.     Input (stocks, barrels)
16.     totalLocks = totalLocks + locks
17.     totalStocks = totalStocks + stocks
18.     totalBarrels = totalBarrels + barrels
19.     Input (locks)
20. EndWhile
21. Output ("Locks sold : ", totalLocks)
22. Output ("Stocks sold : ", totalStocks)
23. Output ("Barrels sold : ", totalBarrels)
24. lockSales = lockPrice * totalLocks
25. stockSales = stockPrice * totalStocks
26. barrelSales = barrelPrice * totalBarrels
27. sales = lockSales + stockSales + barrelSales
28. Output ("total sales : ", sales)
29. If (sales > 1800.0)
30.   Then
31.         Commission = 0.10 * 1000.0
32.         Commission = commission + 0.15 * 800.0
33.         Commission = commission + 0.20 * (sales-1800.0)
34.   Else If (sales > 1000.0)
35.       Then
36.         Commission = 0.10 * 1000.0
37.         Commission = commission + 0.15 * (sales – 1000.0)
38.   Else
39.         Commission =0.10 * sales
40.   EndIf
41. EndIf
42. Output ("commission is $ ",'commission)
43. End commission
```



EXAMPLE – COMMISSION PROBLEM

- **SLICE ON LOCK VARIABLE**

In the program fragment

```
13. Input(locks)
14. While NOT(locks = -1)
15.     Input(stocks, barrels)
16.     totalLocks = totalLocks + locks
17.     totalStocks = totalStocks + stocks
18.     totalBarrels = totalBarrels + barrels
19.     Input(locks)
20.EndWhile
```

There are these slices on locks (notice that statements 15, 17, and 18 do not appear):

S1: S(locks, 13) = {13} DEFINING NODE I-DEF

S2: S(locks, 14) = {13, 14, 19, 20}

S3: S(locks, 16) = {13, 14, 19, 20}

S4: S(locks, 19) = {19} DEFINING NODE I-DEF

SLICE ON STOCKS AND BARRELS

S_5: S(stocks,15) = {13,14,15,19,20}

S_6: S(stocks,17) = {13,14,15, 19,20}

S_7: S(barrels,15) = {13,14,15,19,20}

S_8: S(barrels,18) = {13,14,15, 19,20}

SLICE ON TOTAL LOCKS

S_9: S(totallocks,10) = {10} (A-def)

S_{10}: S(totallocks,16) = {10,13,14,16,19,20} (A-def & C-use)

S(totallocks,21)= {10,13,14,16,19,20} 21 is an O-Use of totallocks, excluded

S_{11}: S(totallocks,24) = {10,13,14,16,19,20} (24 is a C-use of total locks)

SLICE ON TOTAL STOCKS AND TOTALBARRELS

S_{12}: S(totalstocks,11) = {11} (A-def)

S_{13}: S(totalstocks,17) = {11,13,14,15,17,19,20} (A-def & C-use)

S_{14}: S(totalstocks,22) = {11,13,14,15,17,19,20}(22 is an O-Use of totalstocks)

S_{15}: S(totalbarrels,12) = {12}

S_{16}: S(totalbarrels,18) = {12,13,14,15,18,19,20} (A-def & C-use)

S_{17}: S(totalbarrels,23) = {12,13,14,15,18,19,20} (23 is an O-Use of totalbarrels)

SLICE ON VALUES DEFINED BY ASSIGNMENT STATEMENTS (A-defs)

S_{18}: S(lock_price,24) = {7} (A-def)

S_{19}: S(stock_price,25) = {8}

S_{20}: S(barrel_price,26) = {9}

S_{21}: S(lock_sales,24) = {7,10,13,14,16,19,20,24}

S_{22}: S(Stock_sales,25) = {8,11,13,14,15,17,19,20,25}

S_{23}: S(barrel_sales,26) = {9,12,13,14,15,18,19,20,26}

SLICE ON SALES AND COMMISSION

S_{24}: S(sales,27) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{25}: S(sales,28) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{26}: S(sales,29) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{27}: S(sales,33) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{28}: S(sales,34) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{29}: S(sales,37) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

S_{30}: S(sales,38) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

- $S_{24} = S_{10} \cup S_{13} \cup S_{16} \cup S_{21} \cup S_{22} \cup S_{23}$
- If the value of sales is wrong, we first look at how it is computed, and if this is OK, we check how the components are computed

S_{31}: S(commission,31) = {31}

S_{32}: S(commission,32) = {31,32}

S_{33}: S(commission,33) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,
26,27,29,30,31,32,33}

S_{34}: S(commission,36) = {36}

S_{35}: S(commission,37) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,
26,27,36,37}

S_{36}: S(commission,38) = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,
26,27,29,34,38}

S_{37}: S(commission,41) ={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,
26,27,29,30,31,32,33,34,35,36,37,38}

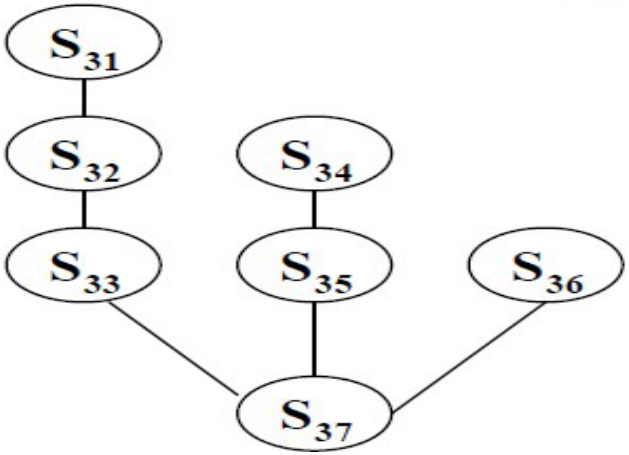
LATTICE OF SLICES

- **What is a slice?**
  - Slice is a set of statement fragment numbers, we can find slices that are subsets of other slices.

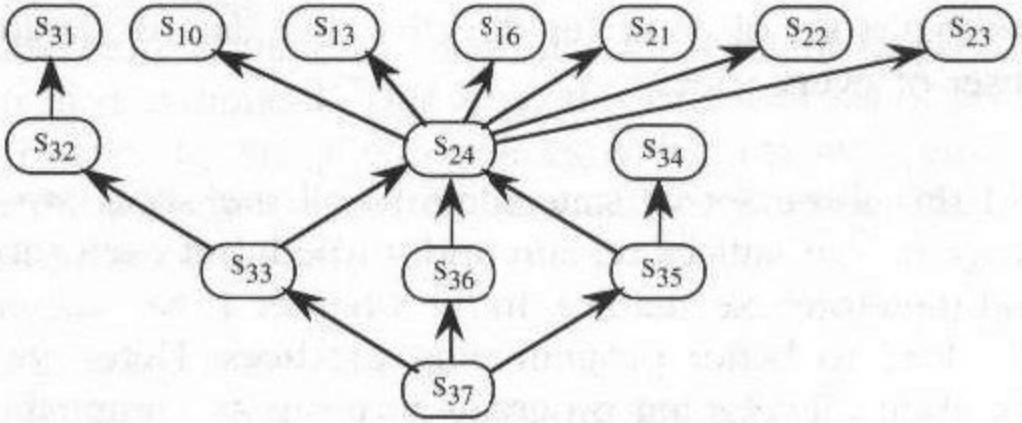
- **What is Lattice?**
  - Lattice is a DAG in which slices are nodes and an edge represents the proper subset relationship.

- This allows us to "**work backwards**" from points in a program, presumably where a fault is suspected.

Lattice of slices in the commission problem



The lattice is a DAG in which slices are nodes and an edge represents the proper subset relationship



Style & Technique

1. Never make a slice S(V,n) for which variables v of V do not appear in statement fragment n.
   - example: defining a slice on the ***locks*** variable at node 17: totalstocks := totalstocks + stocks
   - Defining such slices necessitates tracking the values of all variables at all points of the program
2. Make slices on one variable
3. Make slices for all A-def nodes

4. Make slices for P-use nodes
   - When a variable is used in a predicate, the slice on that variable at that decision statement how the predicate var got its value

5. Slices on non-P-use usage aren't very interesting

6. Consider making slices compilable
   - i.e. a set of compiler directives and declarative statements is a subset of every slice (const, var, etc.)
   - Every slice will be separately compilable (executable)
     - We could develop programs in terms of compilable slices
     - We then could code a slice and immediately test it
     - Then merge all slices into a solid program

```
4.  Dim totalLocks, totalStocks,totalBarrels as Integer
5.  Dim lockSales, stockSales, barrelSales As Real
6.  Dim sales, commission : REAL
7.  Lockprice = 45.0
8.  Stockprice = 30.0
9.  barrelPrice = 25.0
10. totalLocks = 0
11. totalStocks = 0
12. totalBarrels = 0
13. Input (locks)
14. While NOT (locks = -1)    //Input device uses -1 to indicate end of data
15.     Input (stocks, barrels)
16.     totalLocks = totalLocks + locks
17.     totalStocks = totalStocks + stocks
18.     totalBarrels = totalBarrels + barrels
19.     Input (locks)
20. EndWhile
21. Output ("Locks sold : ", totalLocks)
22. Output ("Stocks sold : ", totalStocks)
23. Output ("Barrels sold : ", totalBarrels)
24. lockSales = lockPrice * totalLocks
25. stockSales = stockPrice * totalStocks
26. barrelSales = barrelPrice * totalBarrels
27. sales = lockSales + stockSales + barrelSales
28. Output ("total sales : ", sales)
29. If (sales > 1800.0)
30.    Then
31.          Commission = 0.10 * 1000.0
32.          Commission = commission + 0.15 * 800.0
33.          Commission = commission + 0.20 * (sales-1800.0)
34.     Else If (sales > 1000.0)
35.        Then
36.           Commission = 0.10 * 1000.0
37.           Commission = commission + 0.15 * (sales – 1000.0)
38.     Else
39.           Commission =0.10 * sales
40.     EndIf
41. EndIf
42. Output ("commission is $ ",'commission)
43. End commission
```

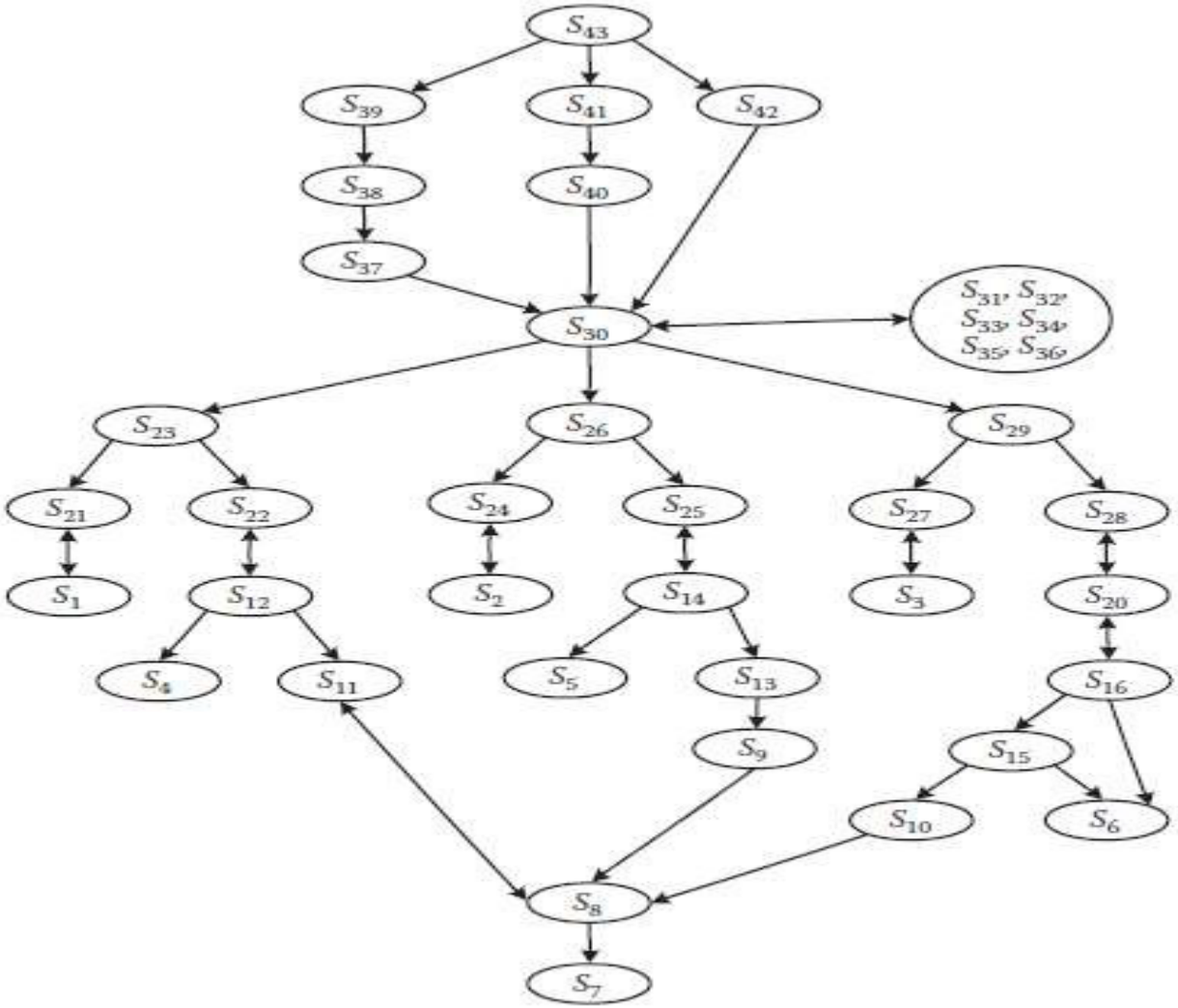
- **S1: S(lockPrice, 7) = {7}**
- **S2: S(stockPrice, 8) = {8}**
- **S3: S(barrelPrice, 9) = {9}**
- **S4: S(totalLocks, 10) = {10}**
- **S5: S(totalStocks, 11) = {11}**
- **S6: S(totalBarrels, 12) = {12}**
- **S7: S(locks, 13) = {13}**
- **S8: S(locks, 14) = {13, 14, 19, 20}**
- **S9: S(stocks, 15) = {13, 14, 15, 19, 20}**
- **S10: S(barrels, 15) = {13, 14, 15, 19, 20}**
- **S11: S(locks, 16) = {13, 14, 19, 20}**
- **S12: S(totalLocks, 16) = {10, 13, 14, 16, 19, 20}**
- **S13: S(stocks, 17) = {13, 14, 15, 19, 20}**
- **S14: S(totalStocks, 17) = {11, 13, 14, 15, 17, 19, 20}**
- **S15: S(barrels, 18) = {12, 13, 14, 15, 19, 20}**
- **S16: S(totalBarrels, 18) = {12, 13, 14, 15, 18, 19, 20}**
- **S17: S(locks, 19) = {13, 14, 19, 20}**
- **S18: S(totalLocks, 21) = {10, 13, 14, 16, 19, 20}**
- **S19: S(totalStocks, 22) = {11, 13, 14, 15, 17, 19, 20}**
- **S20: S(totalBarrels, 23) = {12, 13, 14, 15, 18, 19, 20}**

- **S21: S(lockPrice, 24) = {7}**
- **S22: S(totalLocks, 24) = {10, 13, 14, 16, 19, 20}**
- **S23: S(lockSales, 24) = {7, 10, 13, 14, 16, 19, 20, 24}**
- **S24: S(stockPrice, 25) = {8}**
- **S25: S(totalStocks, 25) = {11, 13, 14, 15, 17, 19, 20}**
- **S26: S(stockSales, 25) = {8, 11, 13, 14, 15, 17, 19, 20, 25}**
- **S27: S(barrelPrice, 26) = {9}**
- **S28: S(totalBarrels, 26) = {12, 13, 14, 15, 18, 19, 20}**
- **S29: S(barrelSales, 26) = {9, 12, 13, 14, 15, 18, 19, 20, 26}**
- **S30: S(sales, 27) = {7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27}**
- **S31: S(sales, 28) = {7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27}**

- **S37: S(commission, 31) = { 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29,30, 31}**
- **S38: S(commission, 32) = { 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29,30, 31, 32}**
- **S39: S(commission, 33) = { 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29,30, 31, 32, 33}**
- **S40: S(commission, 36) = { 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29,34, 35, 36}**
- **S41: S(commission, 37) = { 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29,34, 35, 36, 37}**
- **S42: S(commission, 39) = { 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29,34, 38, 39}**
- **S43: S(commission, 41) = { 7, 8, 9 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29, 30,31, 32, 33, 34, 35, 36, 37, 39}**

Slices are composed of sets of previous slices

- **S1: S(lockPrice, 7) = {7}**
- **S2: S(stockPrice, 8) = {8}**
- **S3: S(barrelPrice, 9) = {9}**
- **S4: S(totalLocks, 10) = {10}**
- **S5: S(totalStocks, 11) = {11}**
- **S6: S(totalBarrels, 12) = {12}**
- **S7: S(locks, 13) = {13}**
- **S8: S(locks, 14) = S7 ∪ {14, 19, 20}** **(S8: S(locks, 14) = {13, 14, 19, 20})**
- **S9: S(stocks, 15) = S8 ∪ {15}**
- **S10: S(barrels, 15) = S8**
- **S11: S(locks, 16) = S8**
- **S12: S(totalLocks, 16) = S4 ∪ S11 ∪ {16}**
- **S13: S(stocks, 17) = S9 = {13, 14, 19, 20}**



Full lattice on commission.

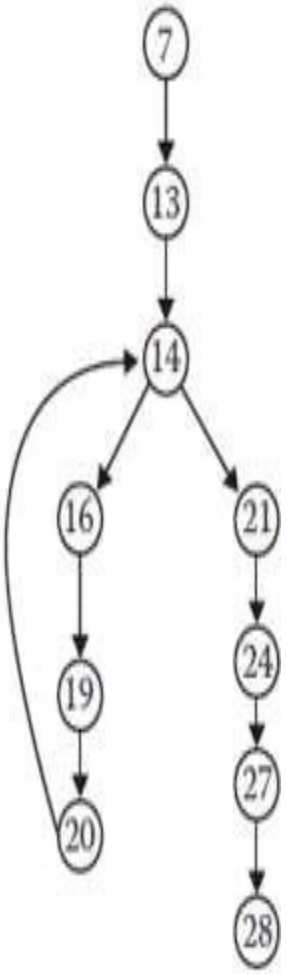
SLICE, SPLICE, SPLIT????

SLICE SPLICING

- The commission program is split into **four slices**
- Slice 1 contains the input while loop controlled by the locks variable.
- Slices 1, 2, and 3 each culminate in a value of sales, which is the starting point for Slice 4, which computes the commission bases on the value of sales.

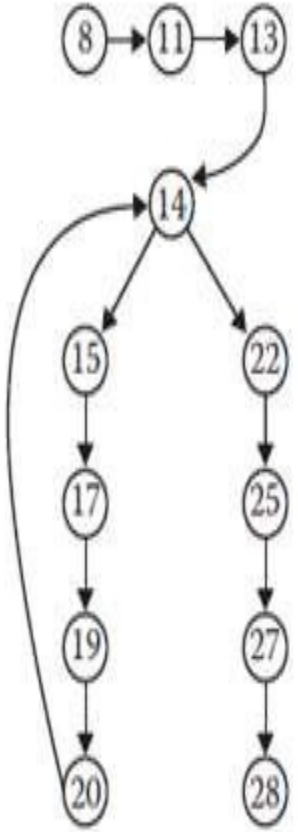
SLICE 1 – BASED ON LOCKS

```
1  Program Slice1 (INPUT,OUTPUT)
2  Dim locks As Integer
3  Dim lockPrice As Real
4  Dim totalLocks As Integer
5  Dim lockSales As Real
6  Dim sales As Real
7  lockPrice = 45.0
13 Input(locks)
'locks = –1 signals end of data
14 While NOT(locks = –1)
16 totalLocks = totalLocks + locks
19 Input(locks)
20 EndWhile
21 Output("Locks sold:", totalLocks)
24 lockSales = lockPrice*totalLocks
27 sales = lockSales
28 Output("Total sales: ", sales)
```



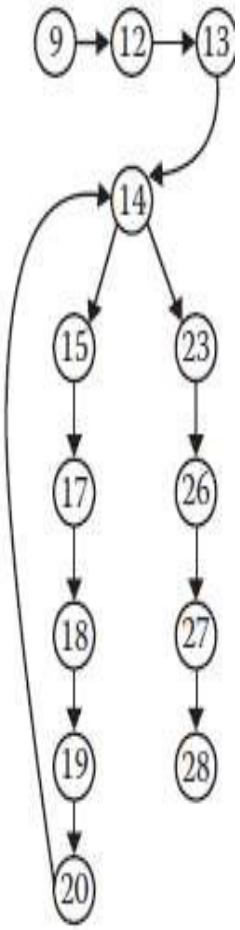
SLICE – 2 – BASED ON STOCKS

```
1  Program Slice2 (INPUT,OUTPUT)
2  Dim locks, stocks As Integer
3  Dim stockPrice As Real
4  Dim totalStocks As Integer
5  Dim stockSales As Real
6  Dim sales As Real
8  stockPrice = 30.0
11 totalStocks = 0
13 Input(locks)
'locks = –1 signals end of data
14 While NOT(locks = –1)
15 Input(stocks)
17   totalStocks = totalStocks + stocks
19   Input(locks)
20 EndWhile
22 Output("Stocks sold:," totalStocks)
25 stockSales = stockPrice*totalStockd
27 sales = stockSales
28 Output("Total sales:," sales)
```



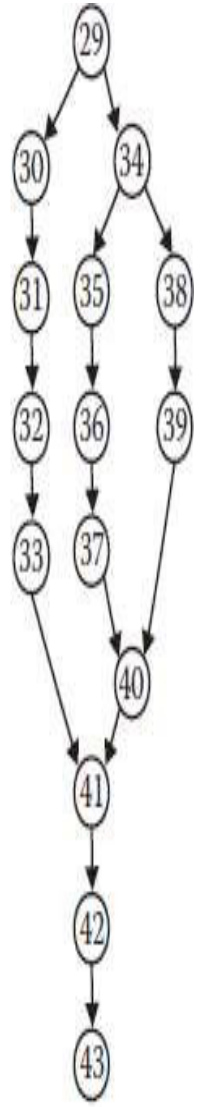
SLICE 3 – BASED ON BARRELS

```
1  Program Slice3 (INPUT,OUTPUT)
2  Dim locks, barrels As integer
3  Dim barrelPrice As Real
4  Dim totalBarrels As Integer
5  Dim barrelSales As Real
6  Dim sales As Real
9  barrelPrice = 25.0
12 totalBarrels = 0
13 Input(locks)
'locks = –1 signals end of data
14 While NOT(locks = –1)
15 Input(barrels)
18   totalBarrels = totalBarrels + barrels
19   Input(locks)
20 EndWhile
23 Output("Barrels sold:," totalBarrels)
26 barrelSales = barrelsPrice * totalBarrles
27 sales = barrelSales
28 Output("Total sales:," sales)
```



SLICE 4 – BASED ON BARRELS

```
1  Program Slice4 (INPUT,OUTPUT)
6  Dim sales, commission As Real
29 If (sales>1800.0)
30   Then
31    commission = 0.10 * 1000.0
32    commission = commission + 0.15*800.0
33    commission = commission + 0.20*(sales–1800.0)
34   Else If (sales > 1000.0)
35    Then
36    commission = 0.10 * 1000.0
37    commission = commission + 0.15*(sales–1000.0)
38    Else
39    commission = 0.10 * sales
40   EndIf
41 EndIf
42 Output("Commission is $," commission)
43 End Commission
```



PROGRAM SLICING TOOLS

Selected Program Slicing Tools

Tool/Product	*Language*	*Static/Dynamic?*
Kamkar	Pascal	Dynamic
Spyder	ANSI C	Dynamic
Unravel	ANSI C	Static
CodeSonar®	C, C++	Static
Indus/Kaveri	Java	Static
JSlice	Java	Dynamic
SeeSlice	C	Dynamic

Difference between Static Slicing and Dynamic Slicing

- Consider a small piece of a **program unit**, in which there is an **iteration block containing an if-else block**.
- There are a few statements in both the if and else blocks that have an effect on a variable.
- In the case of **static slicing**, since the whole program unit is looked at irrespective of a particular execution of the program, the affected statements **in both blocks would be included in the slice.**
- But, in the case of **dynamic slicing** we consider a particular execution of the program, wherein the **if block gets executed** and the affected statements in the else block do not get executed.
- So, that is why in this particular execution case, the dynamic **slice would contain only the statements in the if block**.