SOFTWARE ENGINEERING$_1$

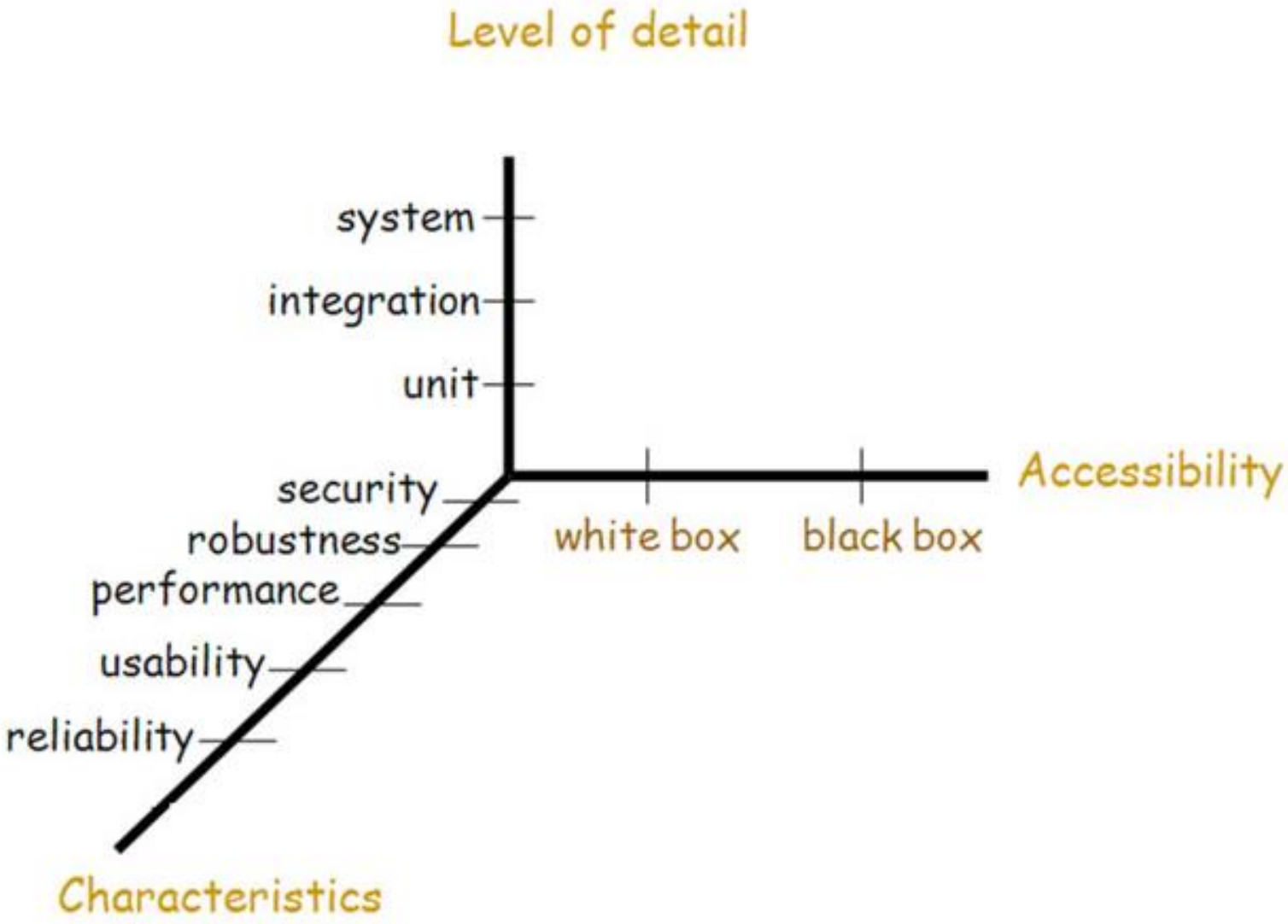
Control Flow Testing

Dr. Mohamad Kassab, mk9508@nyu.edu

LEARNING OUTCOMES – SUBJECT MASTERY

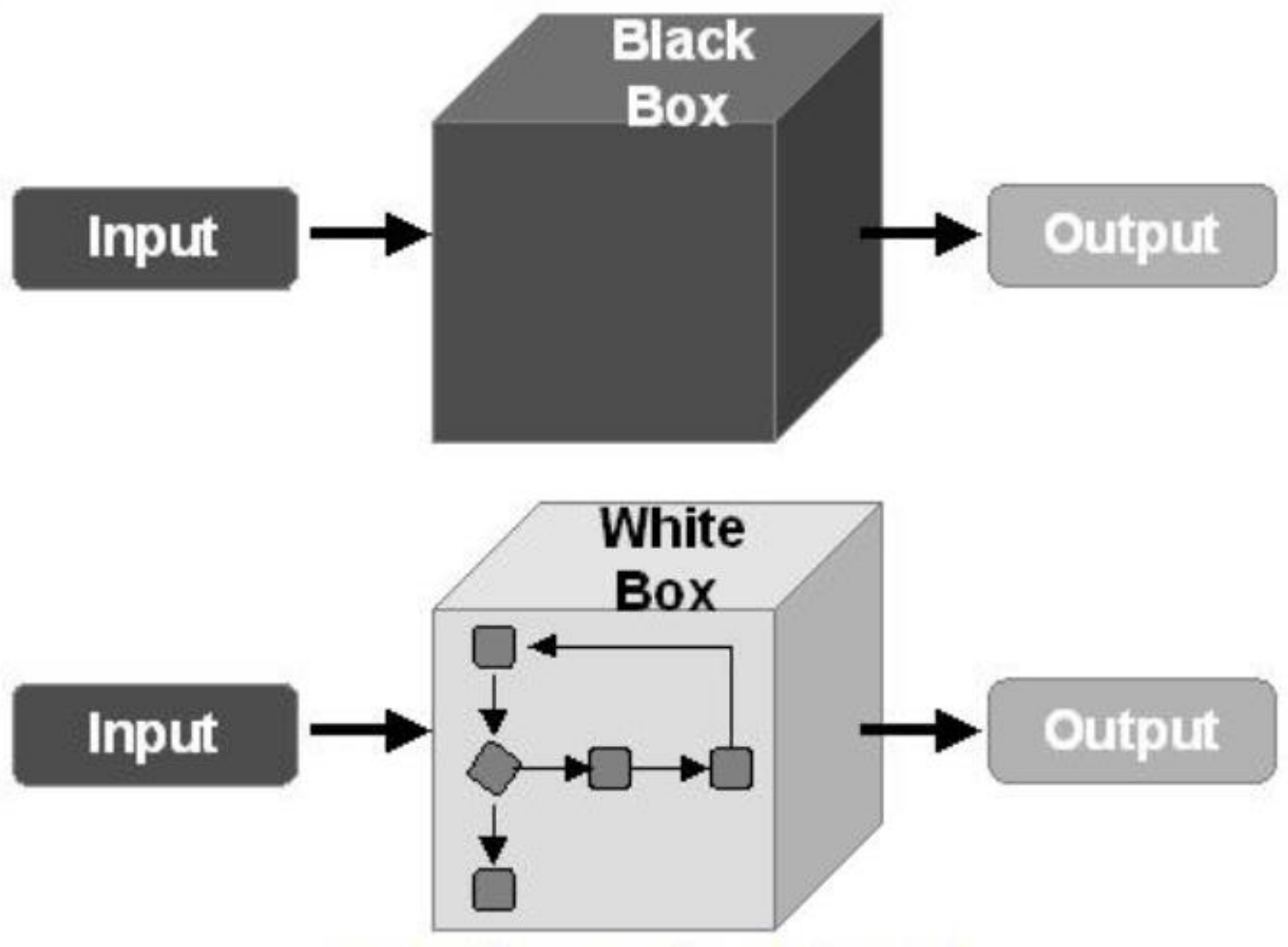
1. Develop practical skills in White Box (Structural) Testing, including the analysis of software internals for effective testing.

2. Master Control Flow Testing as a key White Box technique, enabling systematic exploration of program paths for defect detection.



TESTING DIMENSIONS

BLACK BOX / WHITE BOX TESTING



WHITE BOX TESTING TECHNIQUES

Control Flow Testing

Data Flow Testing

WHAT IS CONTROL FLOW GRAPH?

- A graphical representation of a program's control flow structure.

- It illustrates the flow of control among the various statements, functions, and modules within a program.

CONTROL FLOW TESTING: BASIC IDEA

Two kinds of basic program statements:

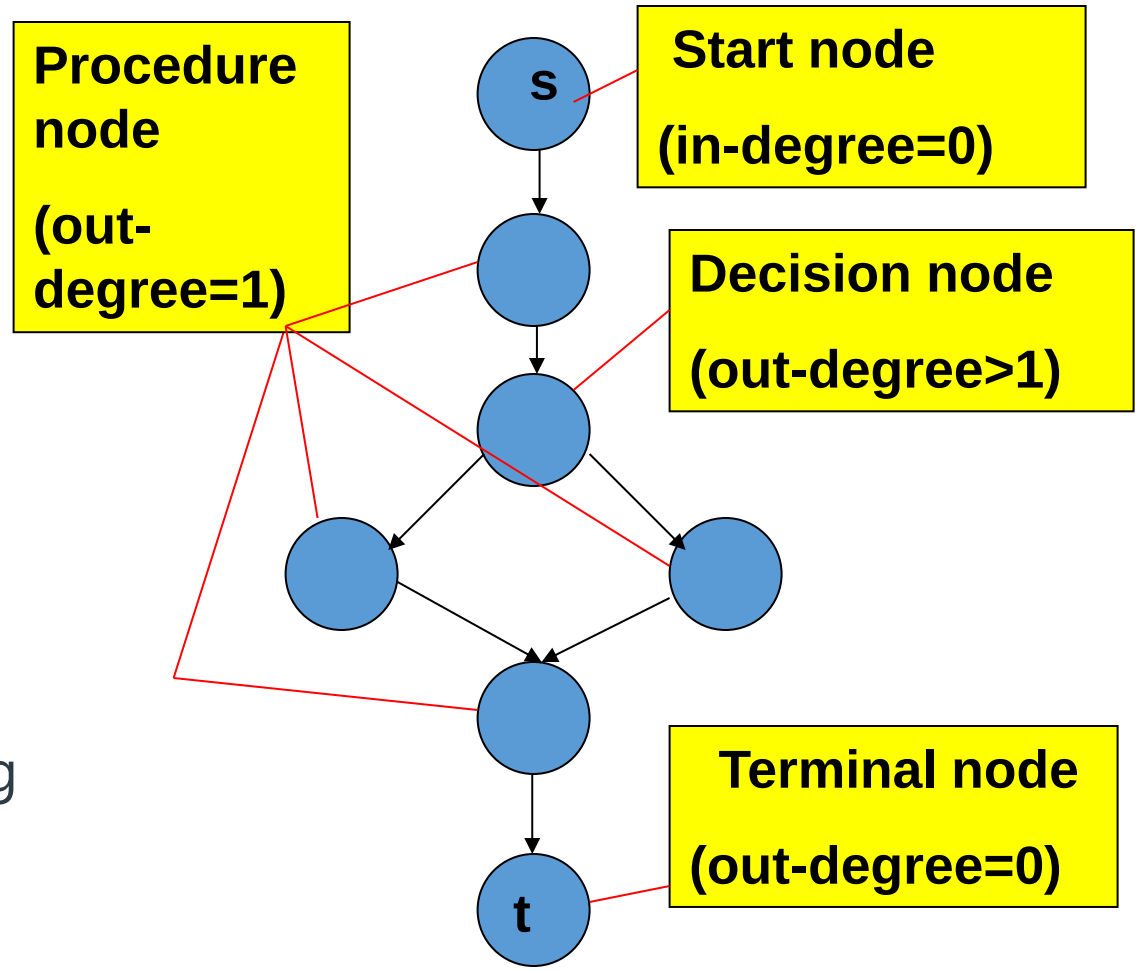
- Assignment statements (Ex. x = 2*y;)
- Conditional statements (Ex. if(), for(), while(), …)

Control flow

- Successive execution of program statements is viewed as flow of control.
- Conditional statements alter the default flow.

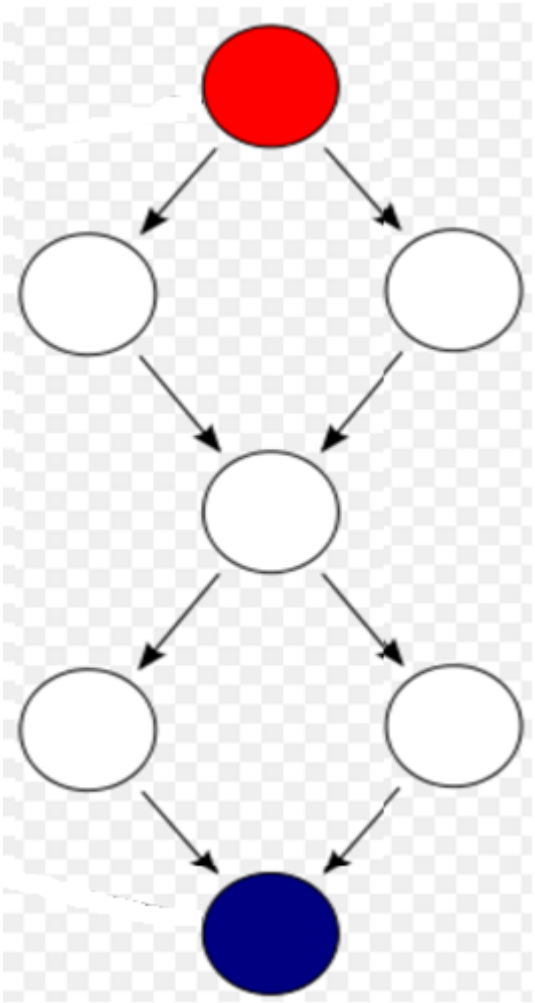
CONTROL FLOW TESTING: BASIC IDEA

- **Quadruple (E, N, s, t)**
  - N : set of nodes
  - s: Start node
  - t: Terminal node
  - E : set of edges
- **In-degree** (FAN-IN): number of edges arriving at node.
- **Out-degree** (FAN-OUT): number of edges leaving the node.
- **Path**: Sequence of consecutive edges



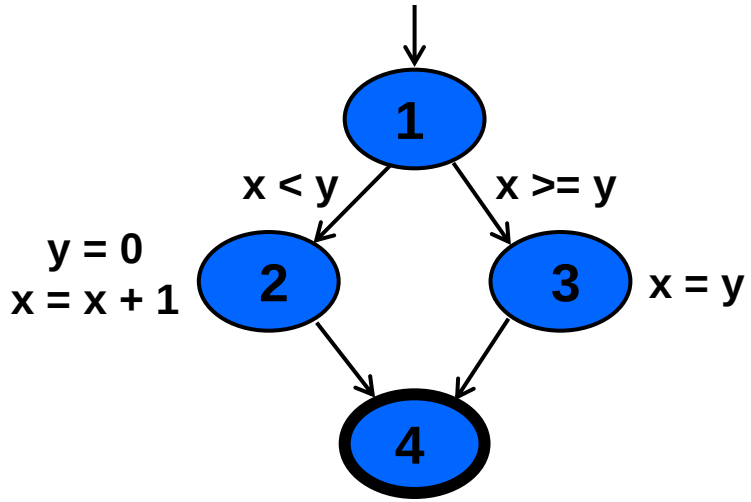
CONTROL FLOW GRAPH (CFG) EXAMPLES

```
if( c1() )
    f1();
else
    f2();

if( c2() )
    f3();
else
    f4();
```

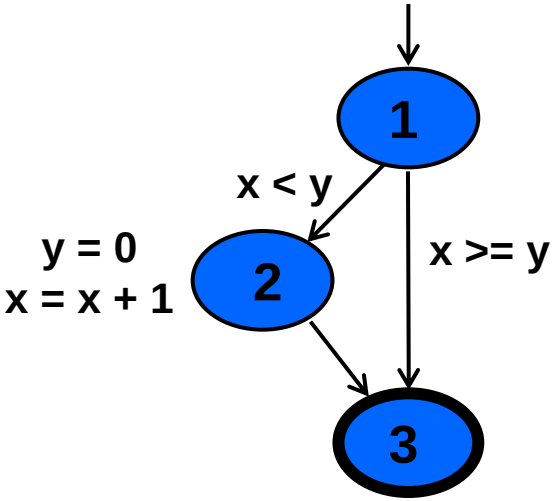


CONTROL FLOW GRAPH (CFG) EXAMPLES

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```

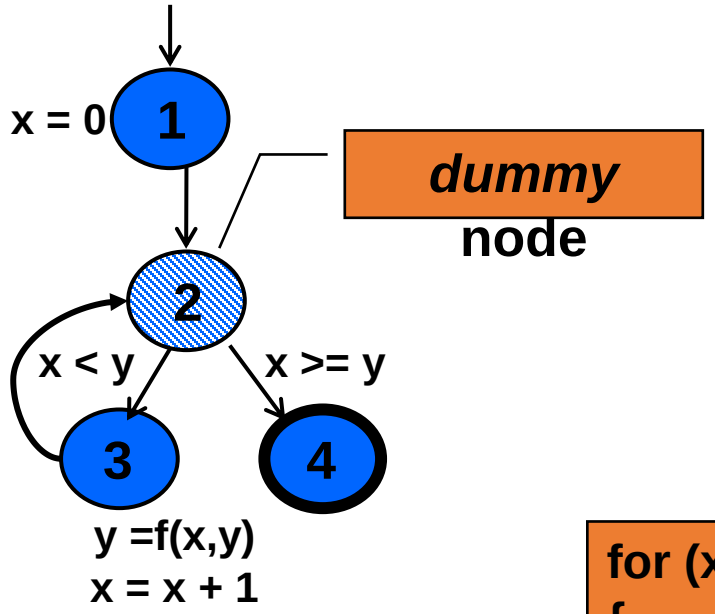



```
if (x < y)
{
   y = 0;
   x = x + 1;
}
```



CONTROL FLOW GRAPH (CFG) EXAMPLES

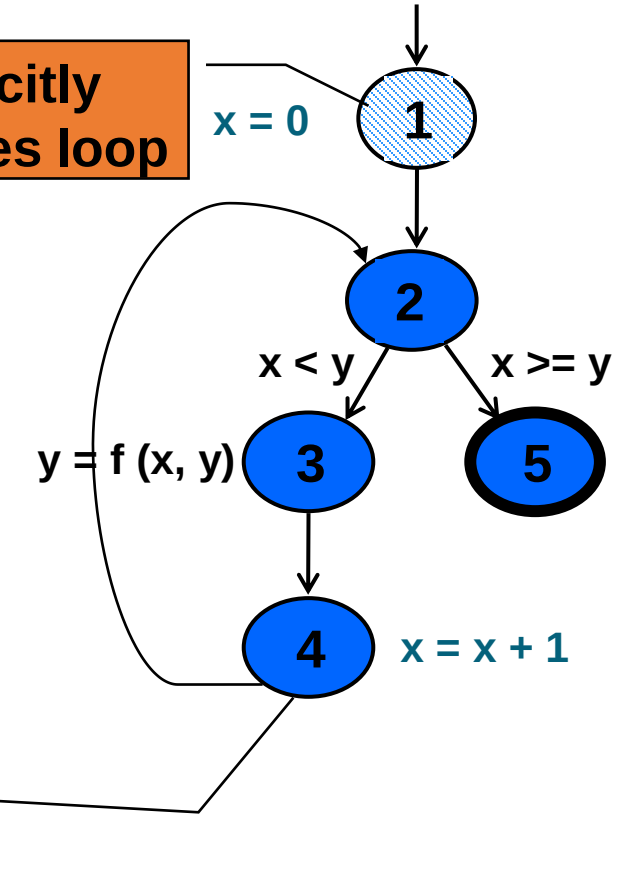
```
x = 0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
```



dummy **node**

implicitly initializes loop

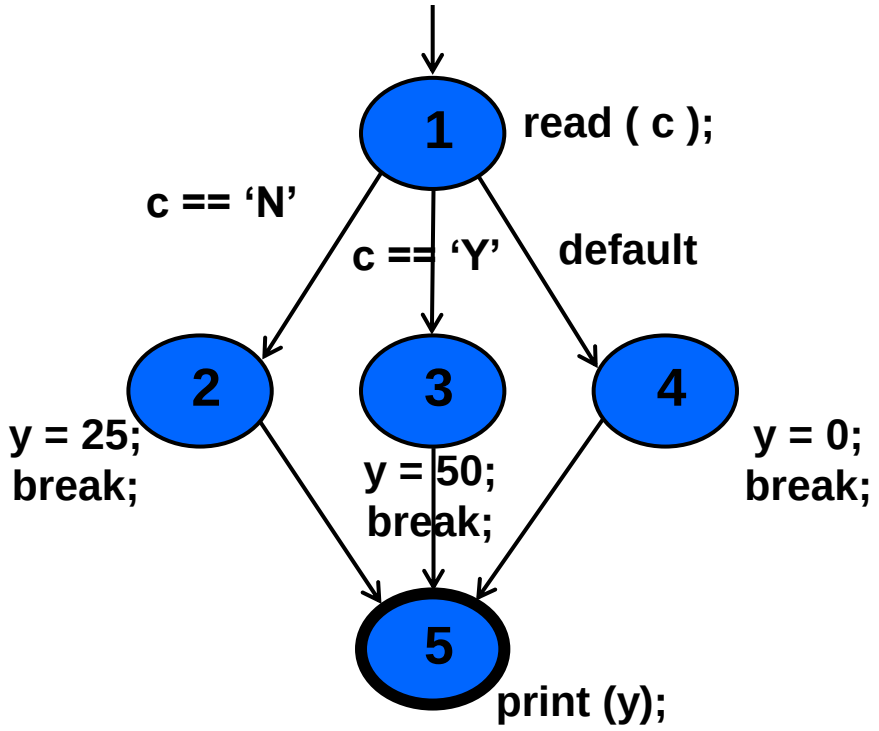
```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```



implicitly increments loop

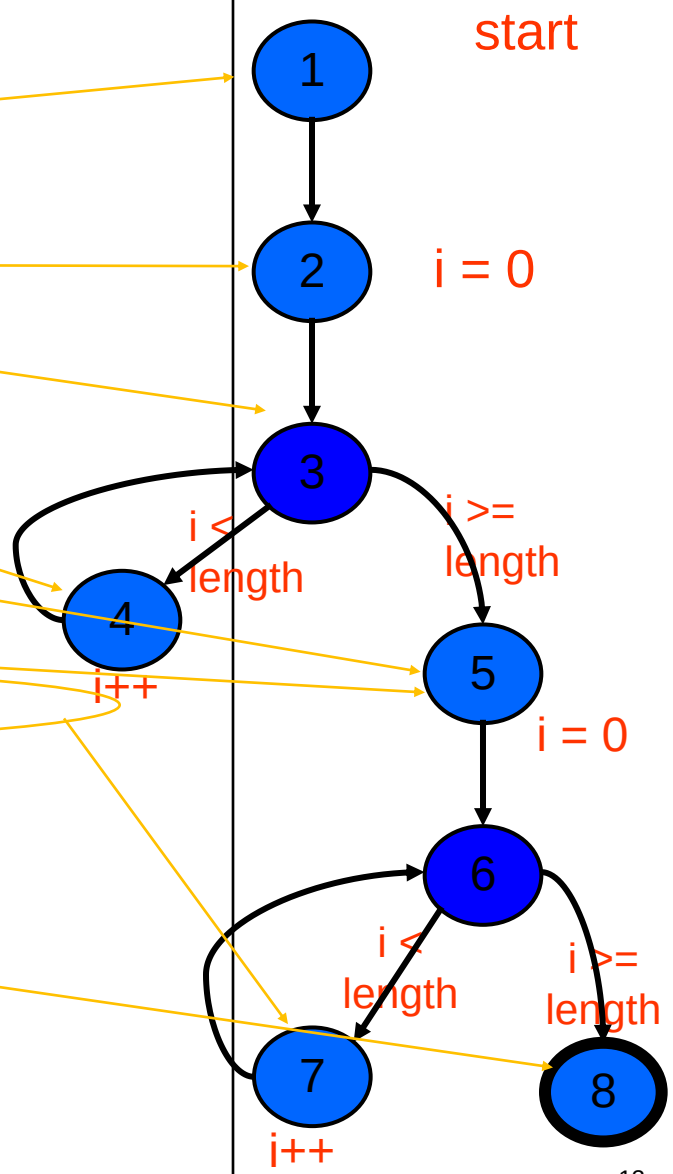
CONTROL FLOW GRAPH (CFG) EXAMPLES

```
read ( c) ;
switch ( c )
{
   case ‘N’:
      y = 25;
      break;
   case ‘Y’:
      y = 50;
      break;
   default:
      y = 0;
      break;
}
print (y);
```



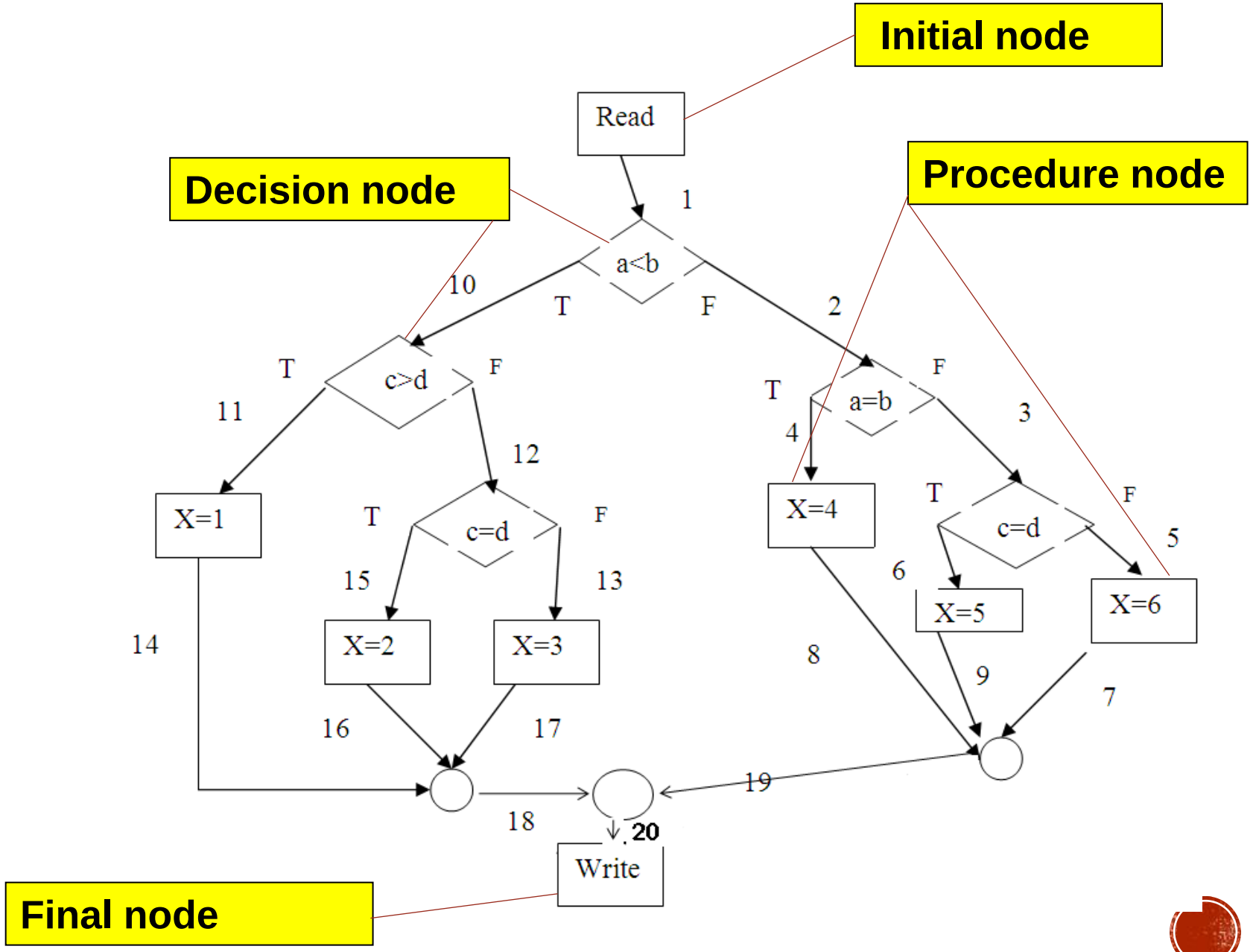
EXAMPLE: CFG FOR COMPUTESTATS

```
public static void computeStats (int [ ] numbers)
{
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers [ i ];
    }
    med   = numbers [ length / 2 ];
    mean = sum / (double) length;

    varsum = 0;
    for (int i = 0; i < length; i++)
    {
            varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
    }
    var = varsum / ( length - 1.0 );
    sd  = Math.sqrt ( var );

    System.out.println ("length:                " + length);
    System.out.println ("mean:                  " + mean);
    System.out.println ("median:                " + med);
    System.out.println ("variance:              " + var);
    System.out.println ("standard deviation: " + sd);
}
```



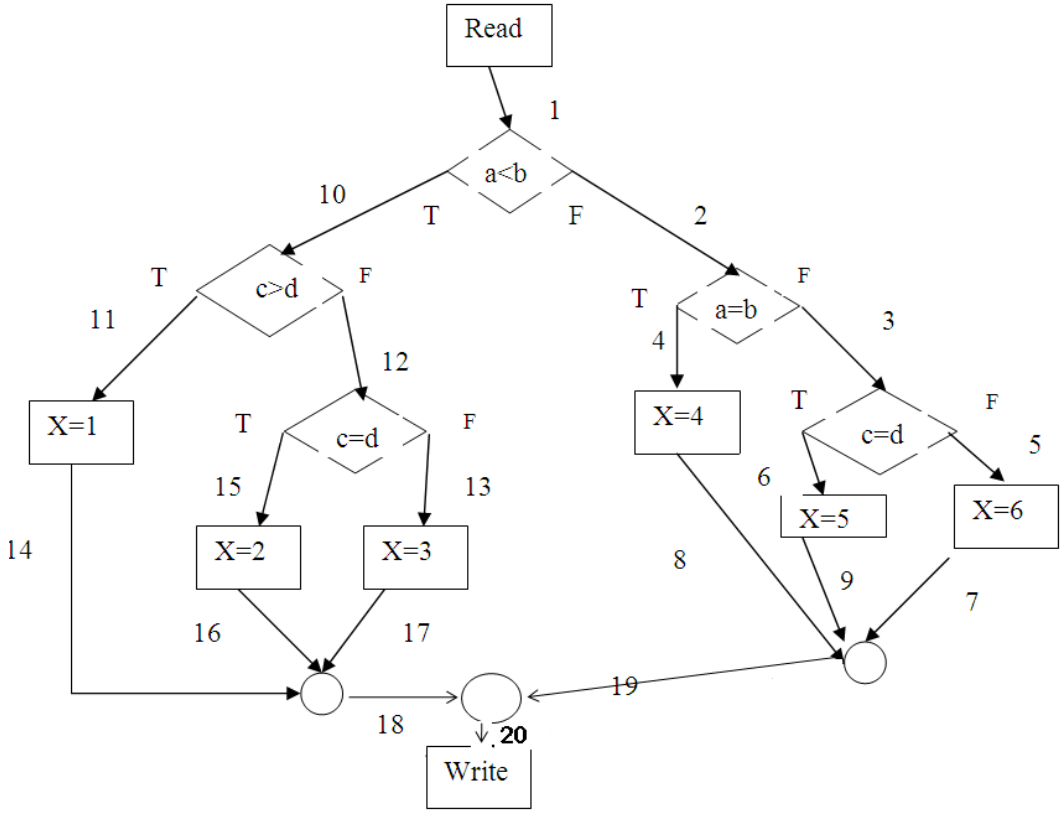
DRAW A CFG !

```
read(a, b, c, d);
if a < b then
    if c > d then
        x := 1
    else
        if c = d then
            x := 2
        else
            x := 3
else
    if a = b then
        x := 4
    else
        if c = d then
            x := 5
        else
            x := 6;
write(x);
```

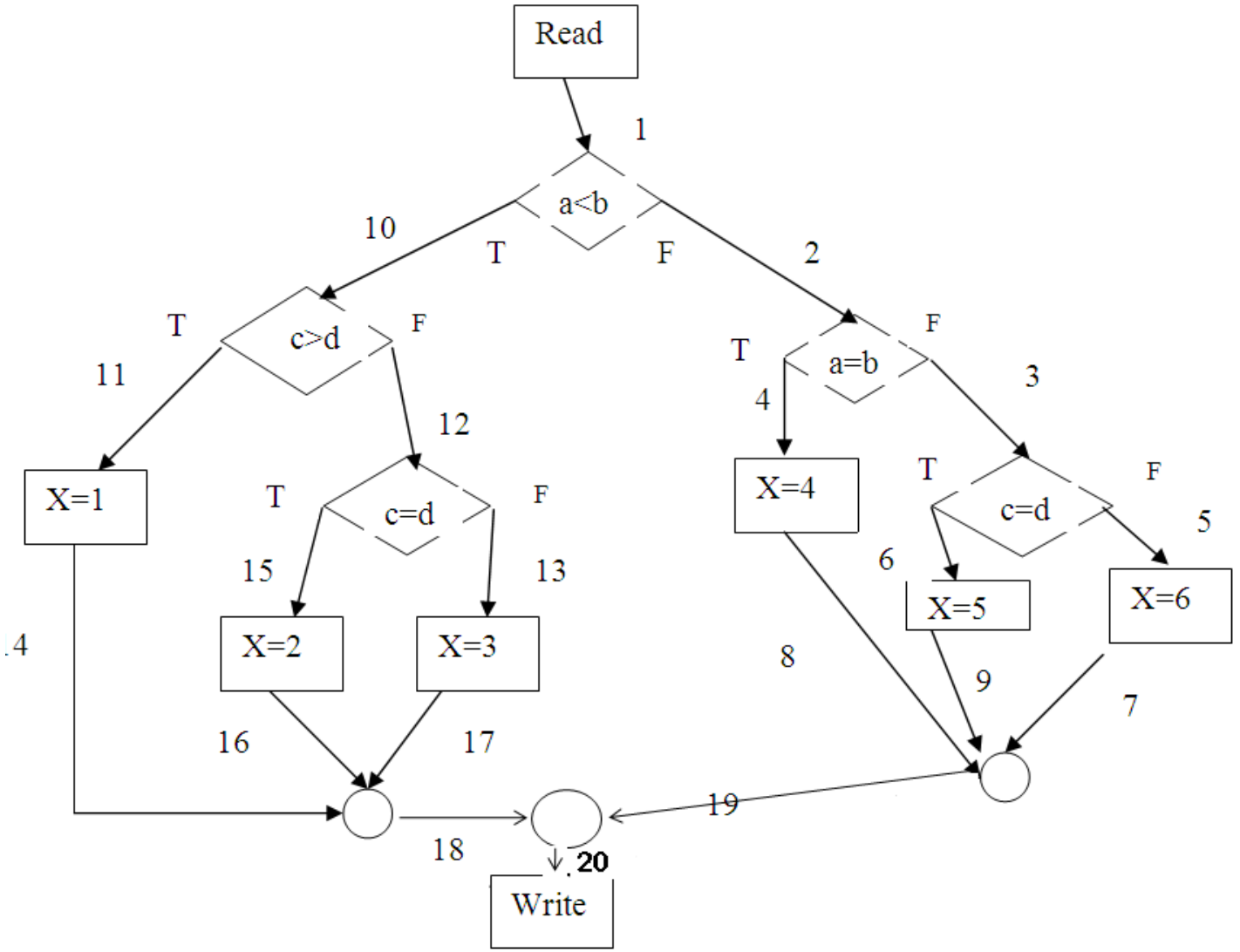


PROGRAM PATH

- 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.



- Tools: Automatically generate test inputs from program paths.



Exercise

- What is a possible input for the above program to make it execute the path

(1,10,12,13,17,18,20)

- What is the expected output?

PATHS IN A CONTROL FLOW GRAPH

- A few paths:

a.

P1: {1,2,3,5,7,19}
P2: {1,2,3,6,9,19}
P3: {1,2,4,8,19}
P4: {1,10,11,14,18}
P5: {1,10,12,13,17,18}
P6: {1,10,12,15,16,18}

b.

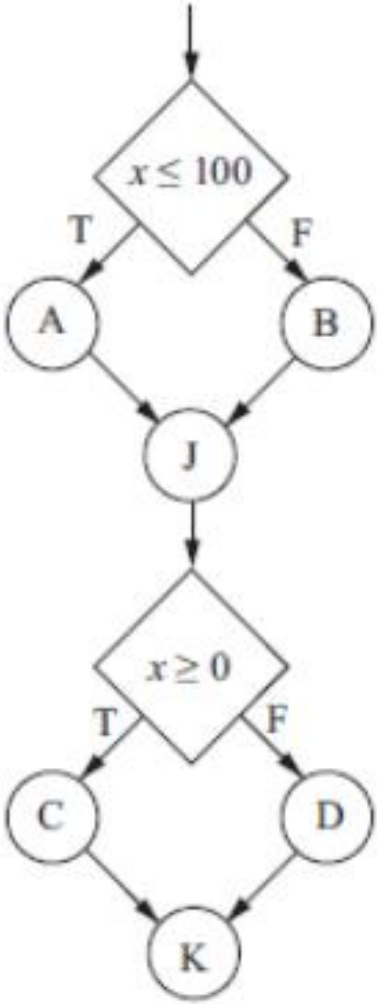
T1: {a=15, b=5, c=17,d=7}
T2: {a=15, b=5, c=9,d=9}
T3: {a=5, b=5, c=any, d=any}
T4: {a=5, b=7, c=9,d=7}
T5: {a=5, b=7, c=9,d=10}
T6: {a=5, b=7, c=9,d=9}

FEASIBLE VS. INFEASIBLE PATHS

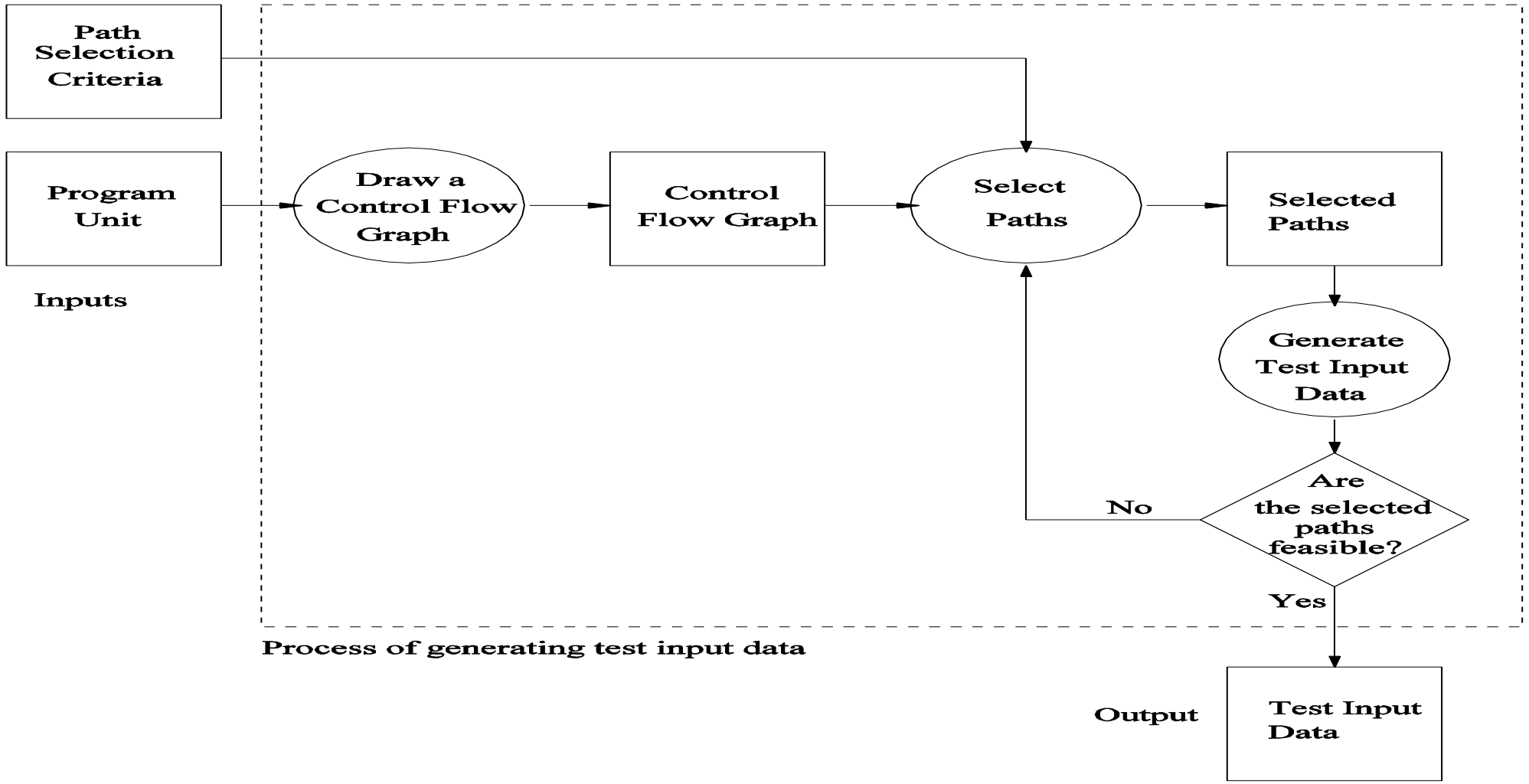
- Two kinds of paths:

- **Executable path**: There exists input so that the path is executed.
- **Infeasible path**: There is no input to execute the path.

- Consider the control flow graph shown where: A, B, J, C, D and K are computation nodes that do not affect the value of x.



OUTLINE OF CONTROL FLOW TESTING



The process of generating test input data for control flow testing.

PATH SELECTION CRITERIA

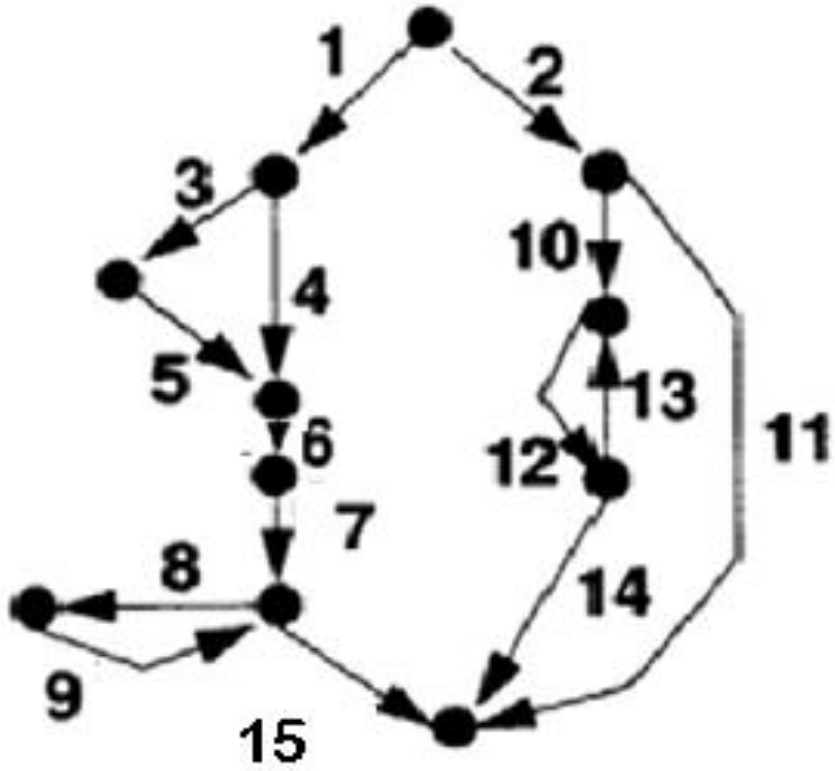
- Program paths are selectively executed.
- Question: **What paths do I select for testing?**
- The concept of *path selection criteria* is used to answer the question.
- Advantages of selecting paths based on defined criteria:
  - Ensure that all program constructs are executed at least once.
  - Repeated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

STATEMENT COVERAGE CRITERION

- Statement coverage means executing individual program statements and observing the output.
- 100% statement coverage means all the statements have been executed at least once.
  - Cover all assignment statements.
  - Cover all conditional statements.
- Less than 100% statement coverage is unacceptable.

STATEMENT COVERAGE: EXAMPLE

- Minimum number of test cases: **2**
- Paths: <2 10 12 14>, <1 3 5 6 7 8 9 15>



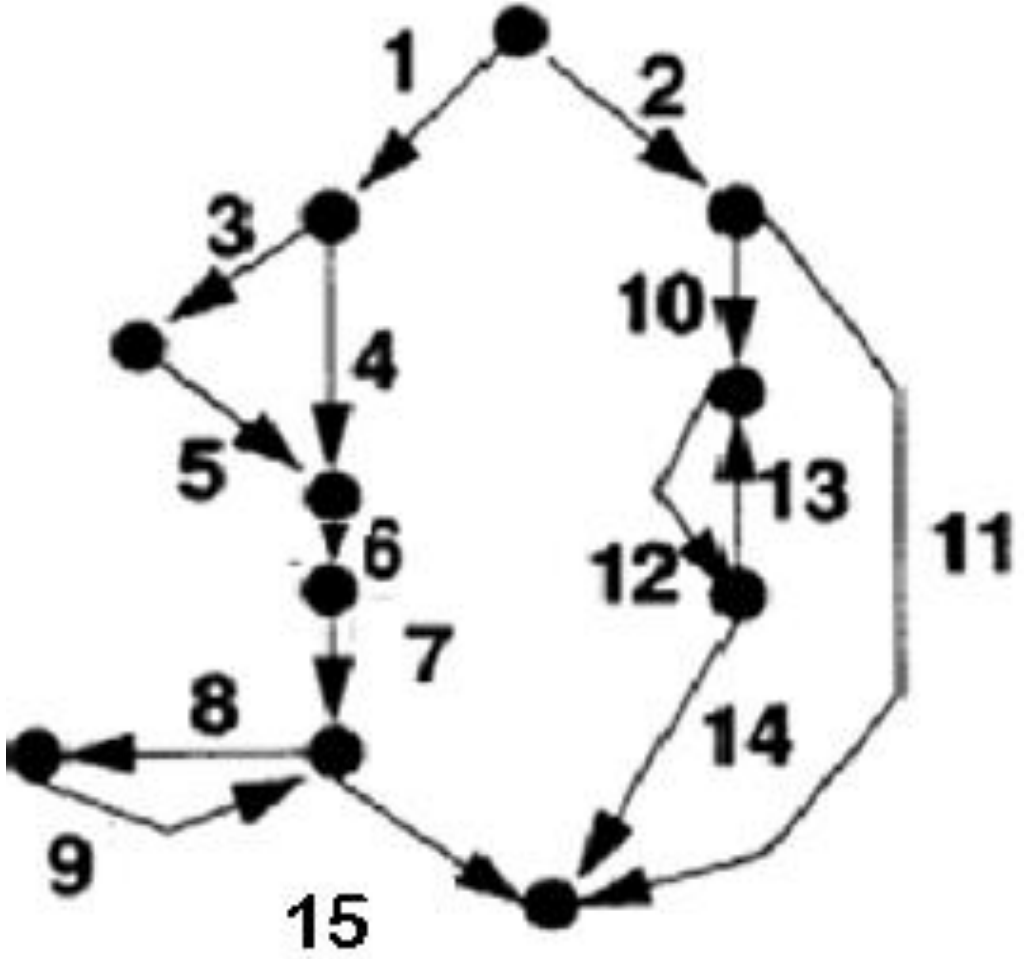
BRANCH COVERAGE CRITERION

- A branch is an outgoing edge from a node in a CFG.
- A condition node has two outgoing branches corresponding to the True and False values of the condition.
- Covering a branch means executing a path that contains the branch.
- Find a set of paths such that every edge lies on at least one path
- 100% branch coverage means selecting a set of paths such that each branch is included on some path.

BRANCH COVERAGE: EXAMPLE

- Minimum number of test cases: 4

- Paths: <2 11>, <2 10 12 13 12 14>, <1 3 5 6 7 15>, <1 4 6 7 8 9 15>



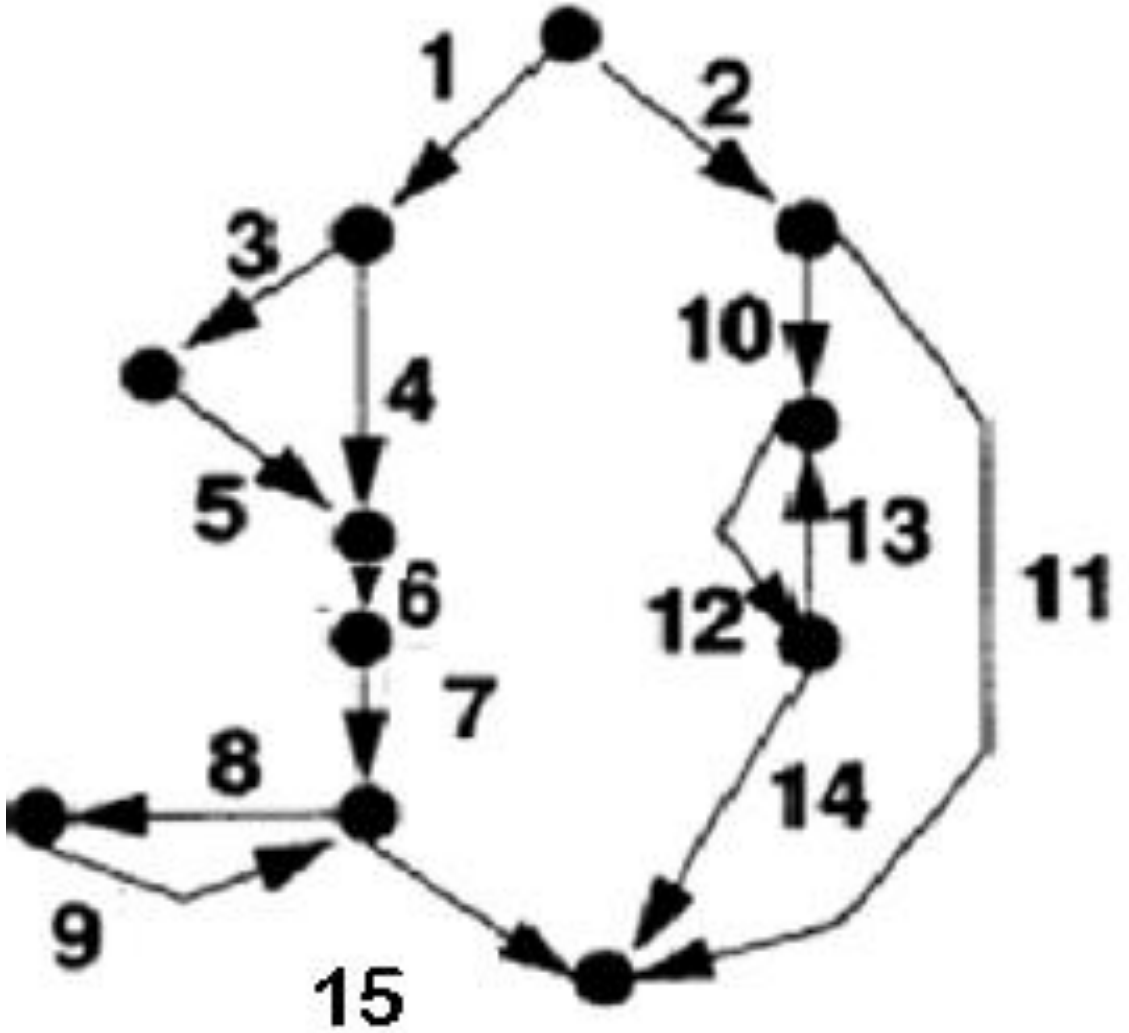
SIMPLE PATH COVERAGE

- **Strategy:**
  - every simple path (which *does not contain the same edge more than once*) is executed once

SIMPLE PATH COVERAGE: EXAMPLE

- Minimum number of test cases: 6

- Paths: <2 11>, <2 10 12 14>, <1 3 5 6 7 15>, < 1 4 6 7 15>, <1 3 5 6 7 8 9 15>, <1 4 6 7 8 9 15>



ALL PATH COVERAGE

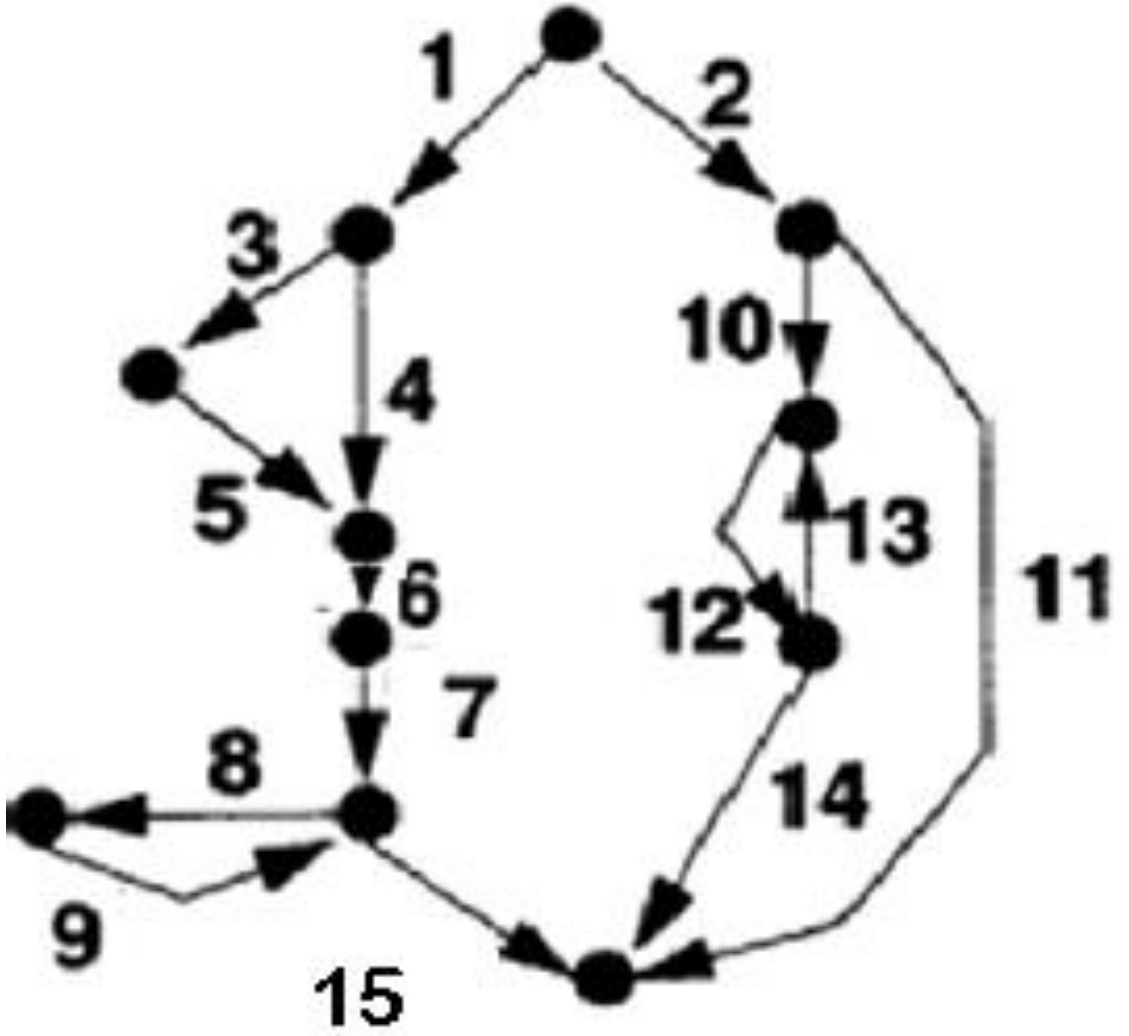
- **Strategy:**
  - every possible program path is executed at least once
  - In terms of flowgraph: find all DD-paths through the flowgraph

.

ALL PATH COVERAGE: EXAMPLE

- Minimum number of test cases: Infinite

- Paths: <2 11>, <2 10 12 14>, <2 10 12 (13 12)^n 14>, <1 3 5 6 7 15>, < 1 4 6 7 15>, <1 3 5 6 7 (8 9)^n 15>, <1 4 6 7 (8 9)^n 15> (any n>0)



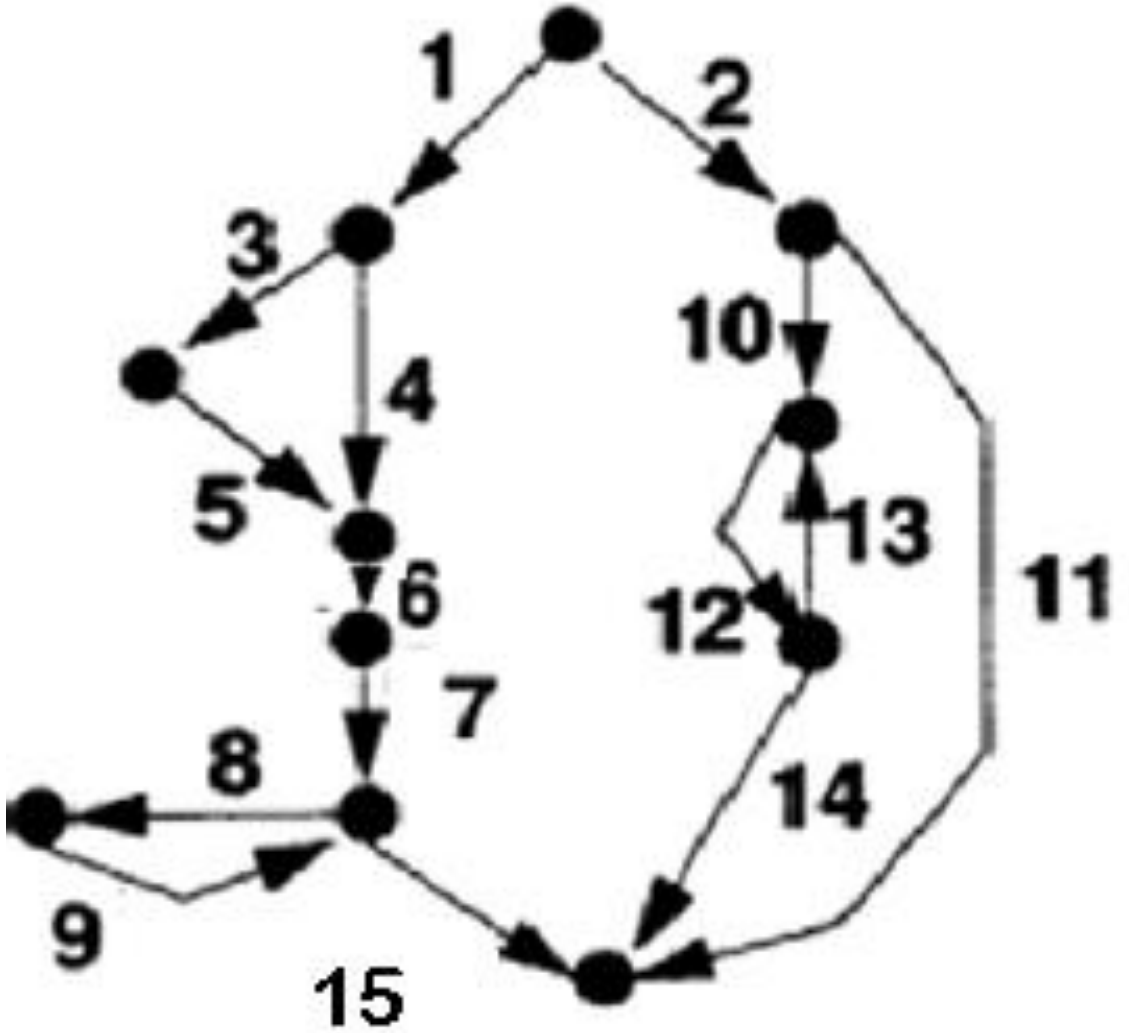
VISIT-EACH-LOOP COVERAGE

- **Strategy:**
  - skip the loop entirely
  - only one pass through the loop

Visit-Each-Loop Coverage: Example

- Minimum number of test cases: **7**

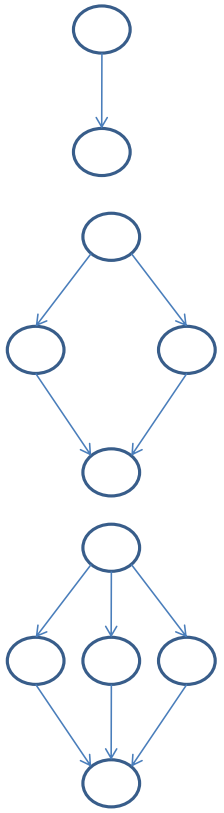
- Paths: <2 11>, <2 10 12 14>, <2 10 12 13 12 14>, <1 3 5 6 7 15>, < 1 4 6 7 15>, <1 3 5 6 7 8 9 15>, <1 4 6 7 8 9 15>



LINEARLY INDEPENDENT PATHS COVERAGE

Two or more paths are considered independent if one cannot be derived from the other. In other words, each path represents a unique combination of decisions and conditions within the program.

McCabe's Cyclomatic Complexity



linear sequence of two or more instructions
e=1, *n=2* therefore *C=1-2+2=1*

If statement
e=4, *n=4* therefore *C=4-4+2=2*

Three way case statement
e=6, *n=5* therefore *C=6-5+2=3*

McCabe Example

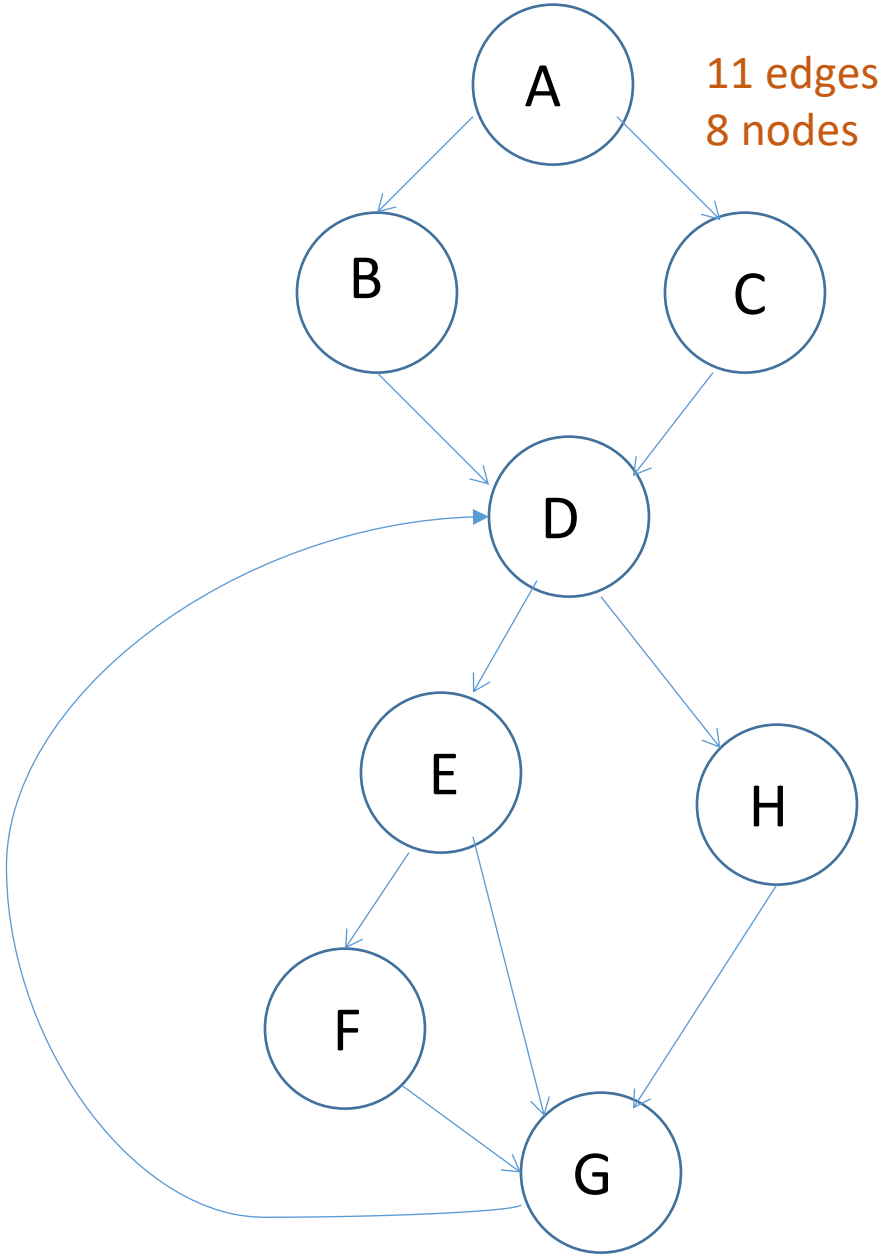
A program fragment : all variables are int

```
1.  i=1;
2   if  (a< b)
3               a=a+b;
4   else
5               b=a+c;
6   while (i<20)
7   {
8   if (b<c)
9   {
10              if (b>c)
11                              b=b+a;
12   }
13              else
14                              c=b+a;
15              i++;
16  }
```

I added line numbers for ease of analysis

Condense line numbers together as follows:

A=1,2;
B=3;
C=4,5;
D=6,7,8;
E=9,10;
F=11;
G=12,15,16, and
H=13,14



Noted: You can draw a graph that assigns a node to each syntactic element of the program (including { and }, which had 16 nodes and 19 edges

This is a perfectly correct approach, and you are less likely to make mistakes this way.

For efficiency, however, I chose to condense line numbers.)

Example, Continued

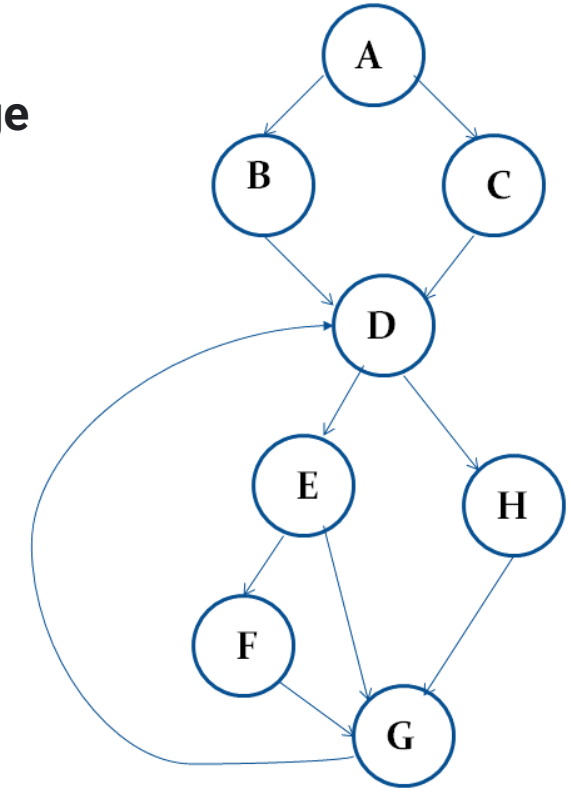
- From the graph we can see that e=11 and n=8.
- Now with c=e-n+2 we have c=11-8+2 = 5.
- As a check, you can count the number of simple branches (if and while statements without complex predicates) and add one to it. There are 4 simple branches so c=4 + 1 = 5.

Linearly Independent Paths

The cyclomatic complexity of a section of source code is the maximum number of linearly independent paths within it—where "linearly independent" means that **each path has at least one edge that is not in one of the other paths**.

List the independent code paths in this code.

One set of 5 independent paths is:
1: ABDEFGD…
2: ABDEG…
3: ABDHG…
4: ACDE…
5: ACDHG…

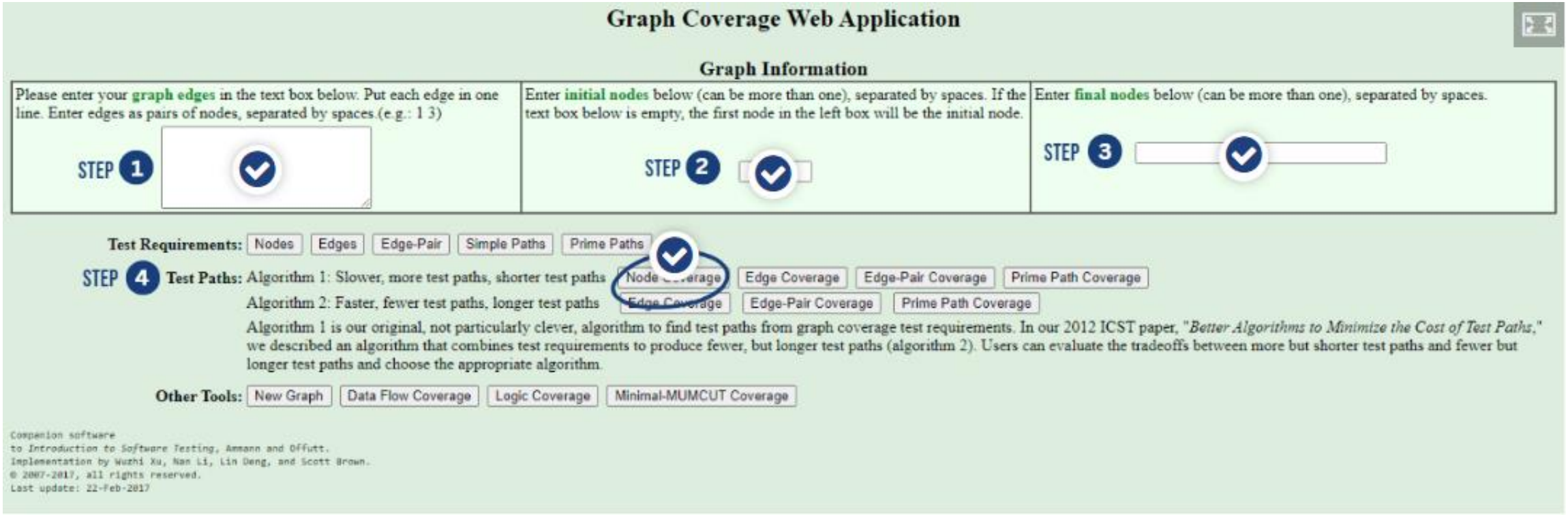


Note that path 4 covers both ACDEFGD…and ACDEG …because DEFGD and DEG are in paths 1 and 2. Now, converting A-G to their corresponding line numbers gives the test paths in terms of the code.

Note: you can get a different set that is also correct – for a code graph vector space there can be more than one basis set.

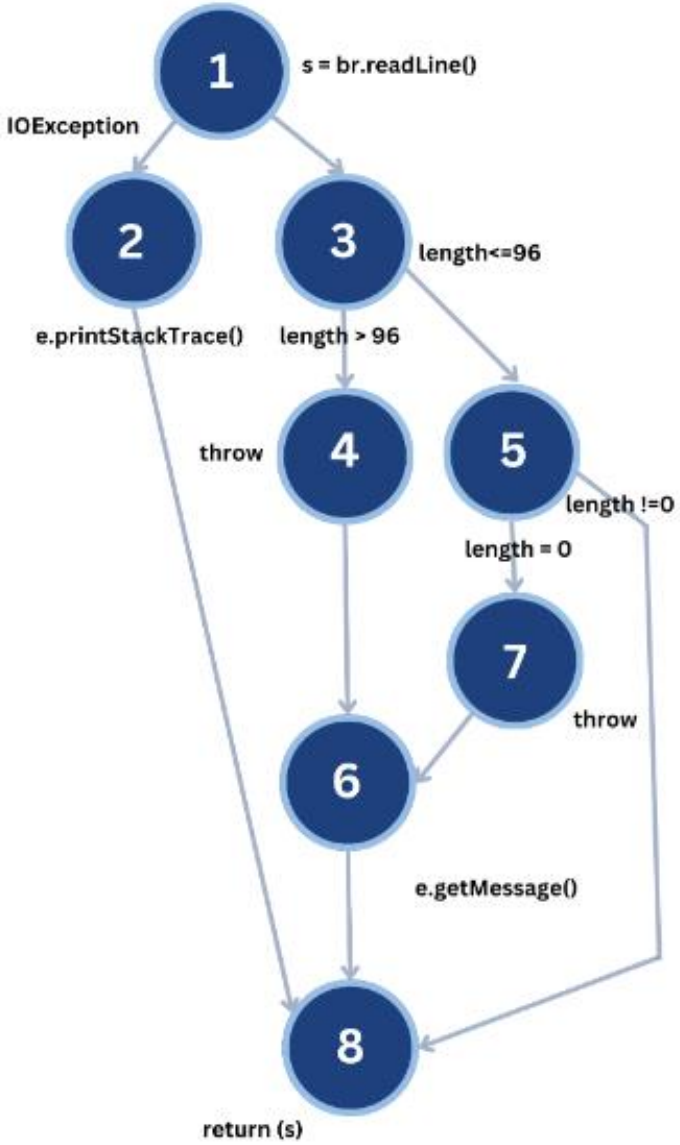
Graph Coverage Web Application

- Several powerful tools are available to assist in graph testing.
- For example, one tool is provided by the authors of the book for free through the web at: https://cs.gmu.edu:8443/offutt/coverage/GraphCoverageLinks to an external site.



GRAPH COVERAGE WEB APPLICATION

```
try {
    s = br.readLine();
    if (s.length() > 96)
        throw new Exception (“too long”);
    if (s.length() == 0)
        throw new Exception (“too short”);
} (catch IOException e) { e.printStackTrace();
} (catch Exception e) {
    e.getMessage();
}
return (s);
```



CONTROL FLOW TESTING TOOLS

- **EclEmma**: EclEmma is an Eclipse plug-in for Java code coverage analysis. While it primarily focuses on coverage analysis, it also provides features for generating JUnit test cases that aim to increase code coverage, including control flow paths.

- **JCrasher**: JCrasher is a Java unit testing tool that automatically generates test cases to exercise various control flow paths in Java programs. It uses a combination of random testing and control flow analysis techniques to create tests that aim to achieve high code coverage.

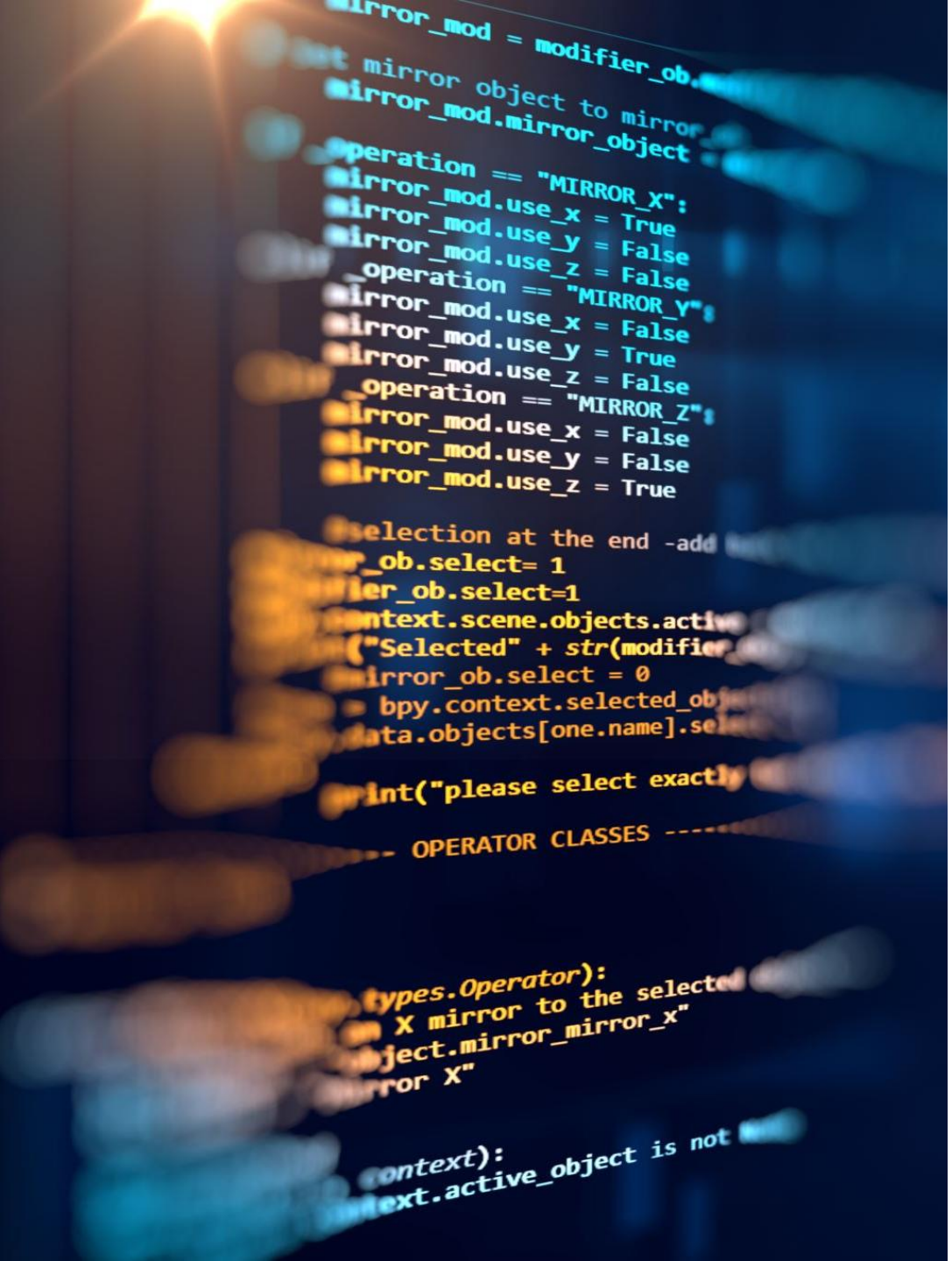
- **CUTE (C Unit Testing Easier)**: CUTE is a unit testing framework for C and C++ programs that can be used to automate control flow testing. It provides facilities for generating and executing test cases automatically to cover different control flow paths in C/C++ code.

CONTROL FLOW TESTING TOOLS

- **Testwell CTC++ (C/C++ Test Coverage for C++ and C)**: Testwell CTC++ is a code coverage tool for C and C++ programs that can be used to automate control flow testing. It provides features for generating test cases and executing them to achieve high coverage of control flow paths.

- **Hansel**: Hansel is a tool for automatic test case generation in Java. It uses symbolic execution techniques to explore different control flow paths in Java programs and generate test cases that aim to achieve high code coverage.

- **PathCrawler**: PathCrawler is a tool for automatic test case generation in Java. It uses symbolic execution and constraint solving techniques to explore different control flow paths in Java programs and generate test cases that aim to achieve high coverage.



WHICH COVERAGE CRITERION TO SELECT?

1. The Complexity of Software Artifact

2. Criticality of the Software Artifact

3. Resource Availability and Cost Considerations

4. Level of Trust

CODE COVERAGE REQUIREMENTS

- Low code coverage indicates inadequate testing, but 100% coverage is impossible in practice **and generally** not cost effective.

- Although 100% code coverage may appear like a best possible effort, even 100% code coverage is estimated to only expose about half the faults in a system.

- **Code coverage of 70-80% is a reasonable goal for system test of most projects with most coverage metrics.**

- **Empirical studies of real projects found that increasing code coverage above 70-80% is time consuming and therefore leads to a relatively slow bug detection rate**

MINIMUM ACCEPTABLE CODE COVERAGE STANDARDS

The aviation standard **DO-178B** requires 100% code coverage for safety critical system

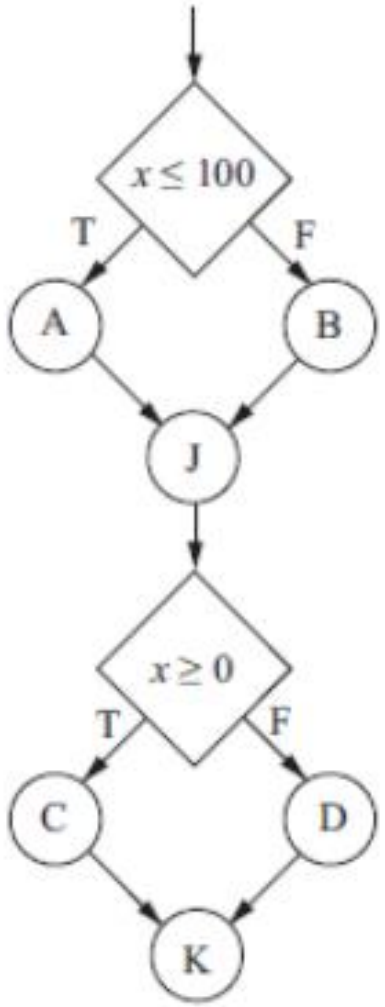
The standard **IEC 61508:2010** "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems" recommends 100% code coverage of several metrics, but the strenuousness of the recommendation relates to the criticality.

The **IEEE Standard for Software Unit Testing** section 3.1.2 specifies 100% statement coverage as a completeness requirement. Section A9 recommends 100% branch coverage for code that is critical or has inadequate requirements specification.

Knowledge Check

In the control flow graph shown, A, B, C, D and J are computation nodes that do not affect the value of variable x.

If there is no limit to the number of test case, how many different feasible execution paths can be tested?



Knowledge Check

- **Which of the following statements are true?**
  - 100% branch coverage guarantees 100% All paths coverage.
  - 100% statement coverage guarantees 100% branch coverage.
  - 100% statement coverage guarantees 100% All paths coverage.
  - 100% branch coverage guarantees 100% statement coverage.

Knowledge Check

You have been asked by your supervisor to "exhaustively test" the software for a mission critical system using All-Paths testing. What is the most appropriate response to give to the supervisor?

- It is impossible to do this kind of testing
- All-paths testing can be completed in a reasonable amount of time without even using automation.
- All-paths testing can be performed but will take forever.
- All-paths testing can be performed on small portions of the most critical code using automation and given enough time.

Summary

- Control flow is a fundamental concept in program execution.
- A program path is an instance of execution of a program unit.
- Select a set of paths by considering path **selection criteria**.
    - Statement coverage
    - Branch coverage
    - Predicate coverage
    - All paths
- From source code, derive a CFG (compilers are modified for this.)
- Select paths from a CFG based on path selection criteria.
- Extract path predicates from each path.
- Solve the path predicate expression to generate test input data.
- There are two kinds of paths.
    - feasible
    - infeasible

Questions