Software Testing and Quality Analysis

Chapter 3:

Structural Testing Methods: Control Flow Graph

Professor in charge:

Ms, M.Halassa

Academic year 2024/2025

Contents

- Test method families
  - Functional testing
  - Structural testing
- Types of structural testing
- Test Methods
  - Test From Control Flow Graph
  - The rules for transforming instructions into graphs
  - Paths in the control graph
  - Expression of control paths
  - Tests from Data Flow Graph

Test methods families

Two major families of test methods

- Functional test methods (or black box testing)
- Structural test methods (or white box testing),

Functional testing: Black-box testing

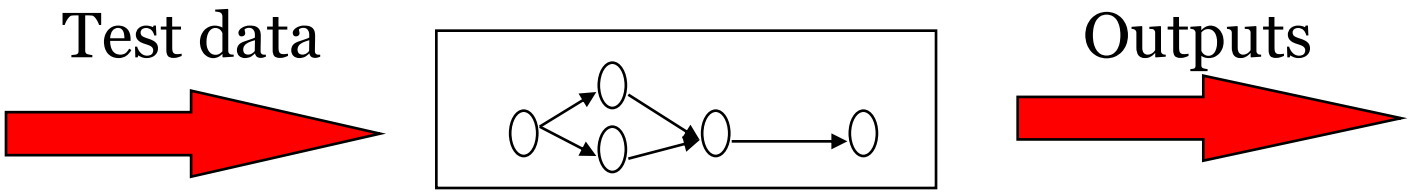
- The purpose of functional testing is to test what the program is supposed to do, not how it does it.
- The program structure is therefore unknown or ignored and the input data is selected according to the program specifications.

Functional testing: based on specification

Structural Testing:White Box Testing

- Tests are used to validate the structure of each module making up the software.
- Here we check what the program does and not what it is supposed to do.

Structural testing: based on program analysis



Test Methods

❖Uses the structure to derive test cases

- ***At the unit level***: uses the code (instructions, conditions, connection)
- ***At the integration level***: uses the call graph between modules
- ***At system level***: uses menus, processes

❖Completes the black box testing by studying the realization (not the specification)

❖Is associated with coverage criteria

- Coverage of all instructions
- Coverage of all conditions
- Coverage of all uses of a variable

Types of structural testing

2 methods used :

- Derivation of test sets from the **control flow graph:**
  - Follows the sequence of operations that are represented by a graph
  - Seeks to cover all instructions, all paths
- Derivation of test sets from **Data Flow Graph**
  - Follows the life of the variables of the program: definition, use, destruction
  - Seeks to cover all assignments, all uses -

1. Control flow testing

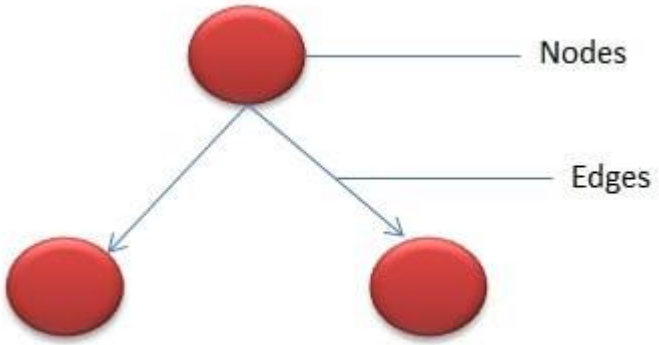
- Oldest structural testing technique.
- Basic technique for many other tests
- Detailed and complex.
- Used mainly for unit testing and component testing.
- Based on execution paths.
- Different types of coverage may be targeted.

1. Control flow testing

- Control-flow testing is done through control-flow graphs, to better understand what a code module does.
- Control-flow graphs give us a visual representation of the structure of the code.
- The Algorithm is:
  - First convert a section of the code into a control graph.
  - Then analyze the possible paths through the graph.

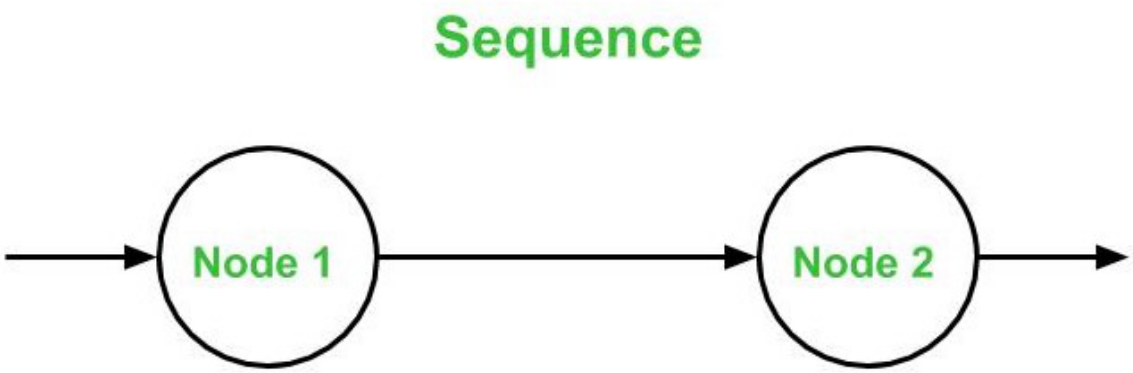
Control Flow Graph (CFG)

- A **CFG** control flow graph consists **of edges** and **nodes.**
- **Node**: instructions or a sequence of instructions (block)
- **Edge:** Transfer of control
- **Block**: a sequence of instructions such that if the first instruction is executed the others are also executed(no branching).
- A control graph makes it possible to build **the paths**
- Assume a start node and an end node
- A path is a sequence of nodes from start to end
- There can be **many possible paths and** some impossible paths.



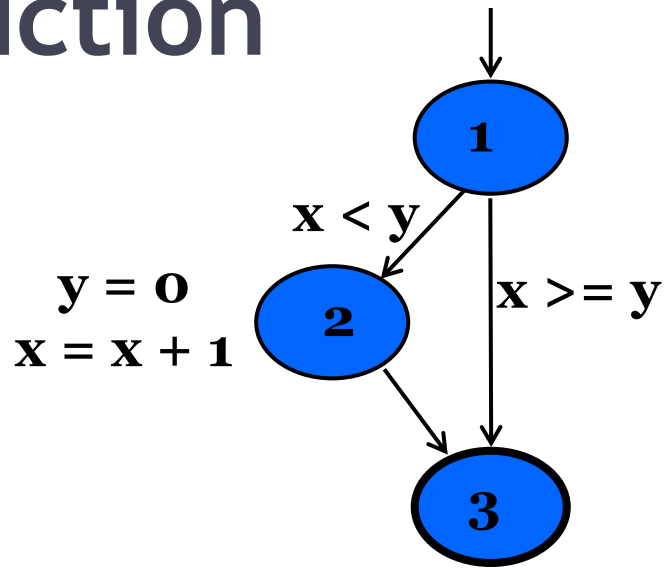
The rules for transforming instructions into graphs

CFG: Sequential Statements

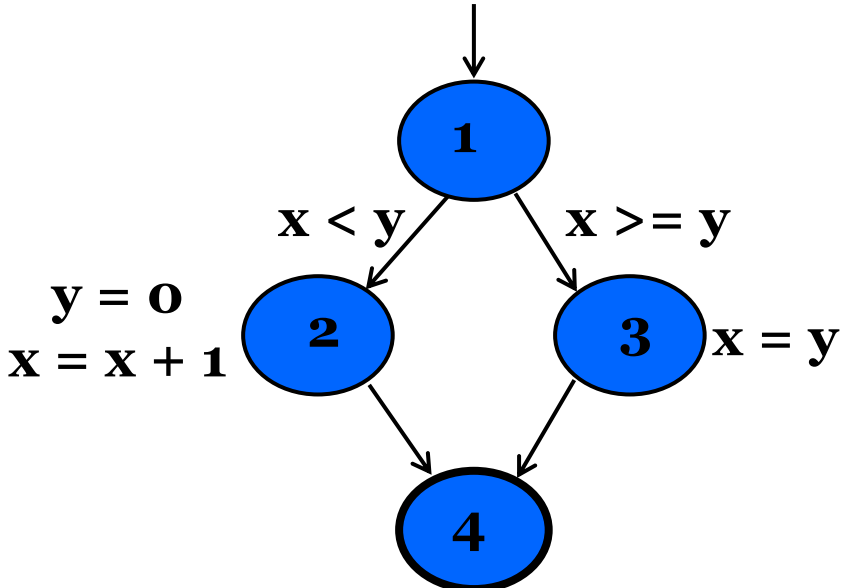


CFG: The if instruction

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

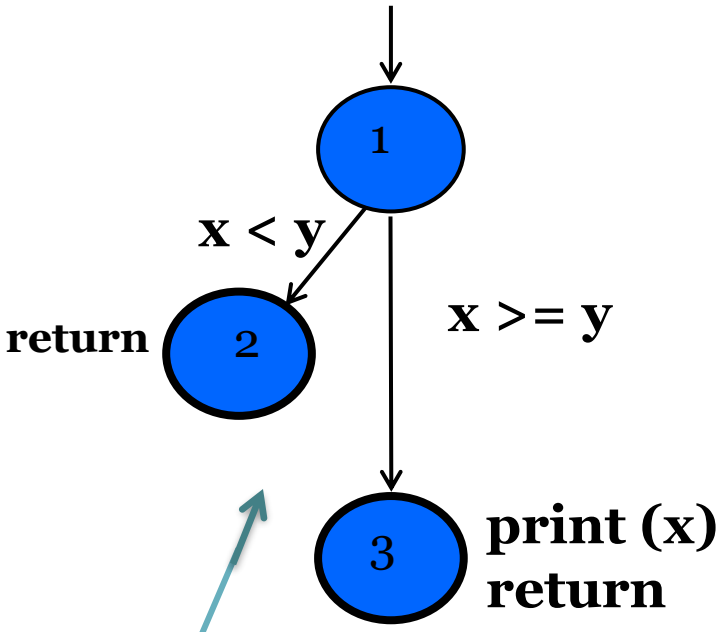



```
if (x < y)
{    y = 0;
   x = x + 1;}
else
{    x = y;
}
```



CFG: The return instruction

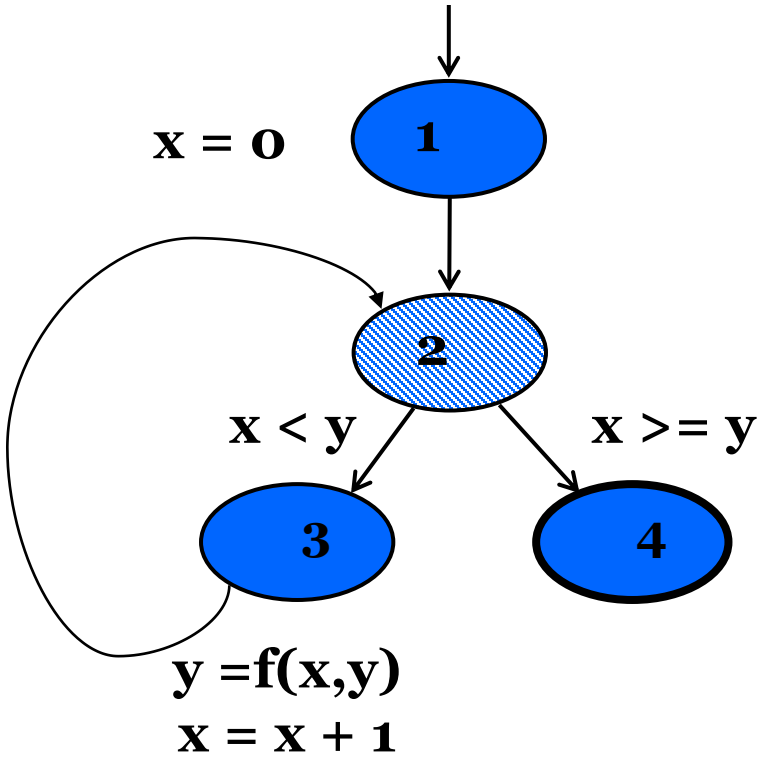
```
if (x < y)
{
   return;
}
print (x);
return;
```



No **arc** between **nodes** 2 and 3.
The **return** node must be separate.

CFG: while loops

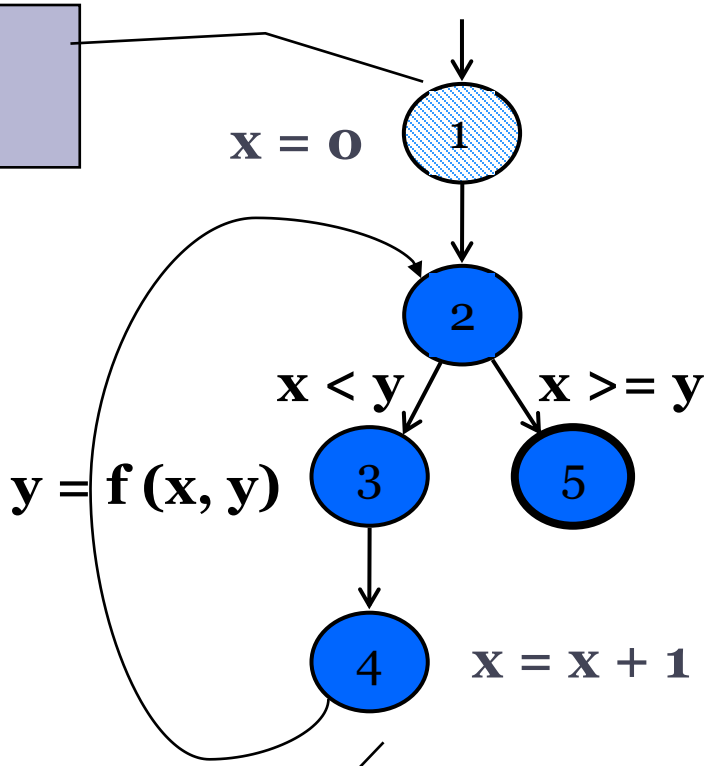
- ```
  if (x < y)
  {
     return;
  }
  print (x);
  return;
  ```


  ```

CFG:"for" loops

- **initialisation**

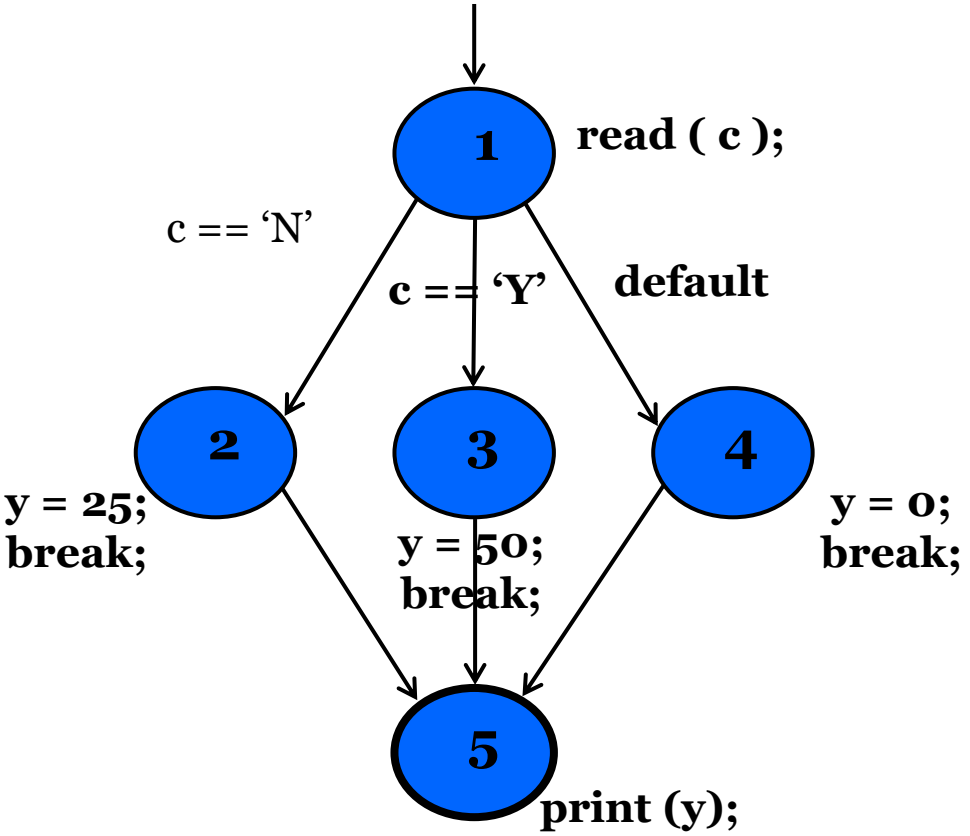
```
for (x = 0; x < y;
x++)
{
   y = f (x, y);
}
```



Implicitly: Loop increment

CFG: the case structure (switch)

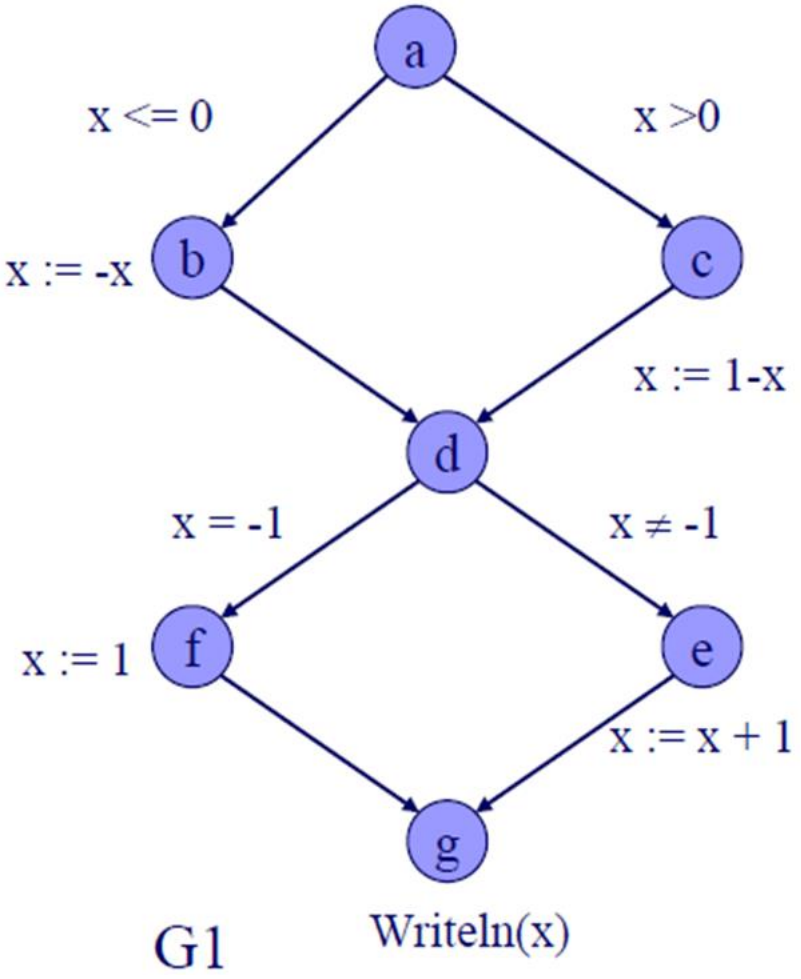
```
read ( c) ;
switch ( c )
{  case 'N':
   y = 25;
      break;
   case 'Y':
      y = 50;
      break;
   default:
      y = 0;
      break;}
print (y);
```



Control Flow Graph

- The program P1 has the following control flow graph:

```
if x <= 0 then x := -x
else x := 1 - x;
if x = -1 then x=1
else x := x+1;
writeln(x)
```



Paths in the control graph

- The graph G1 is a control graph that admits an input -the node a -, an output -the node g.
  - the path *[a, c, d, e, g]* ***is a control or test path,***
  - path *[b, d, f, g]* ***is not a control path.***

- The graph G1 comprises 4 control paths:
  - β1 = *[a, b, d, f, g]*
  - β2 = *[a, b, d, e, g]*
  - β3 = *[a, c, d, f, g]*
  - β4 = *[a, c, d, e, g]*

Expression of control graph

- The graph G1 can be expressed in algebraic form in the following form:

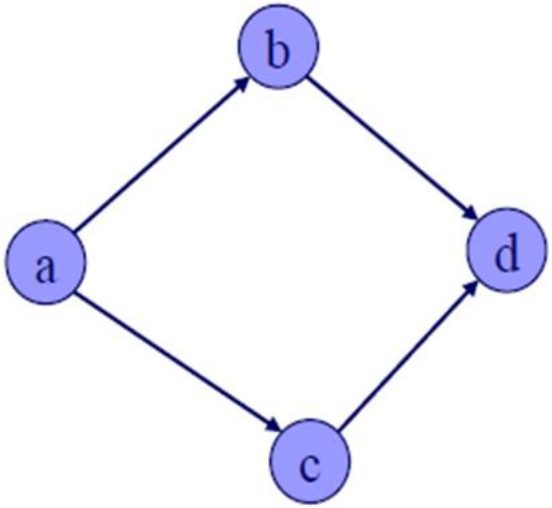
G1 = abdfg+ abdeg+ acdfg+ acdeg
the + sign designates the logical "or" between paths.

- Simplification of the expression of paths

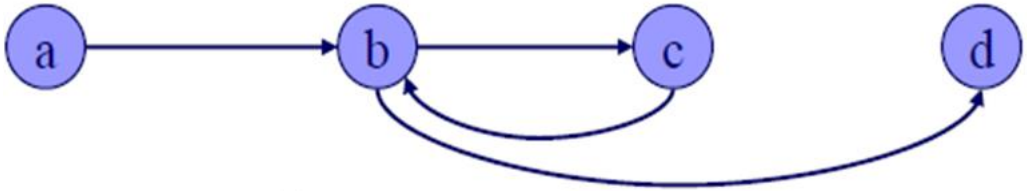
G1 = a (bdf+ bde+ cdf+ cde) g
G1 = a (b + c) d (e + f) g

Calculation of the expression of the control graph

sequentiel : ab



alternative :
a (b + c) d

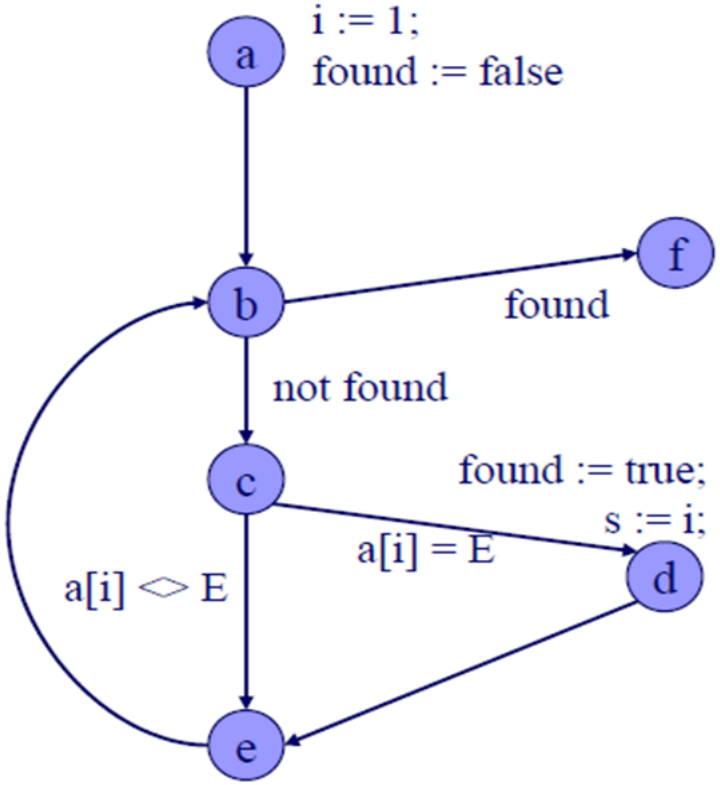


iterative : ab (cb)* d

Control graph with loops

- The program P2 has the following control flow graph:

```
i := 1;
found := false;
while (not found) do
      begin
      if (a[i] = E) then
         begin
         found := true;
         s := i;
         end;
      i := i + 1;
      end;
```



G2 = ab [c (1 + d) eb]* f

G2

Path Expressions - Exercise

- Here is the following program P4:

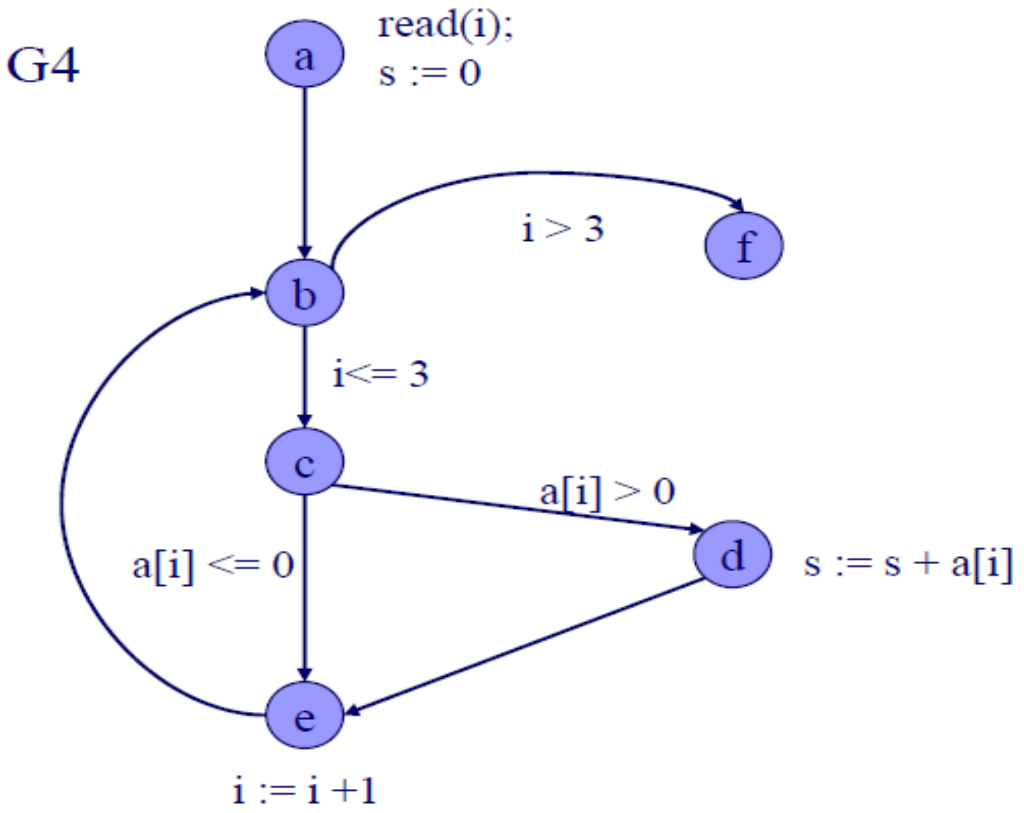
```
read(i);
s := 0;
while (i <= 3) do
      begin
      if a[i] > 0 then s := s + a[i];
      i := i + 1;
      end
end;
```

Question:

- Establish **the control flow graph** of this program
- **Provide path expression**

Path Expressions - Solution

Program Control Flow Graph of p4



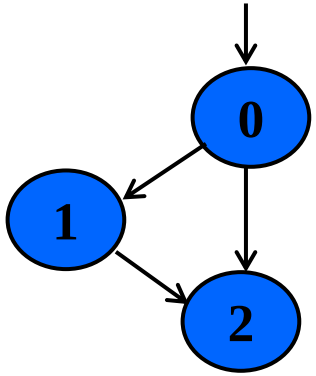
G4 = ab [c (1 + d) eb]* f

Structural Test Methods: All-nodes criterion (statement coverage)

- Statement coverage consists of choosing a test case from all those allowing each (accessible) instruction of the code to be executed.
- This involves selecting test data until all instructions in the code have been executed.
- ***Aim***: to raise awareness of all the control paths that allow us to visit all the **nodes of the control graph.**

Structural Test Methods: All-nodes criterion (statement coverage)

- Example:



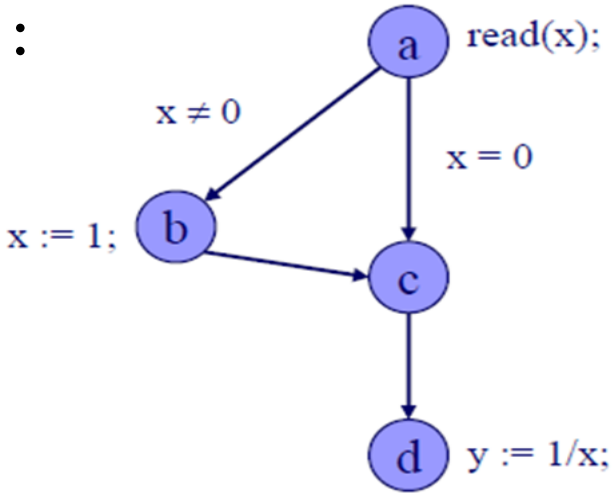
Node Coverage : TR = { 0, 1, 2 }
Test Path = [0 1 2]

- Statement coverage is satisfied by the path [0,1,2].

Limits of the statement coverage

- There is the following program :

```
read(x);
...
if (x <> 0) then x := 1;
...
y  := 1/x;
```



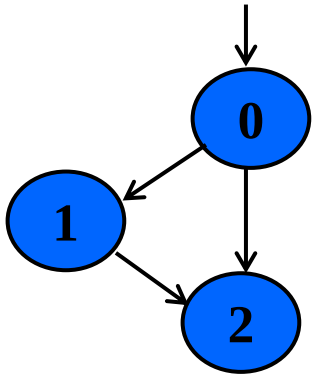
All-nodes coverage is satisfied by the path [abcd] without any error being detected.

Division by zero !

Structural Test Methods: All-edges criterion (branch coverage)

- If we try to cover all the **nodes** without covering all the **adges**, we risk not detecting certain defects on the uncovered adges..., In this case, it is the decisions that are the subject of the tests,

- ***Goal:*** we must visit all **the adges of the control graph (**cover all program decision-making at least once)



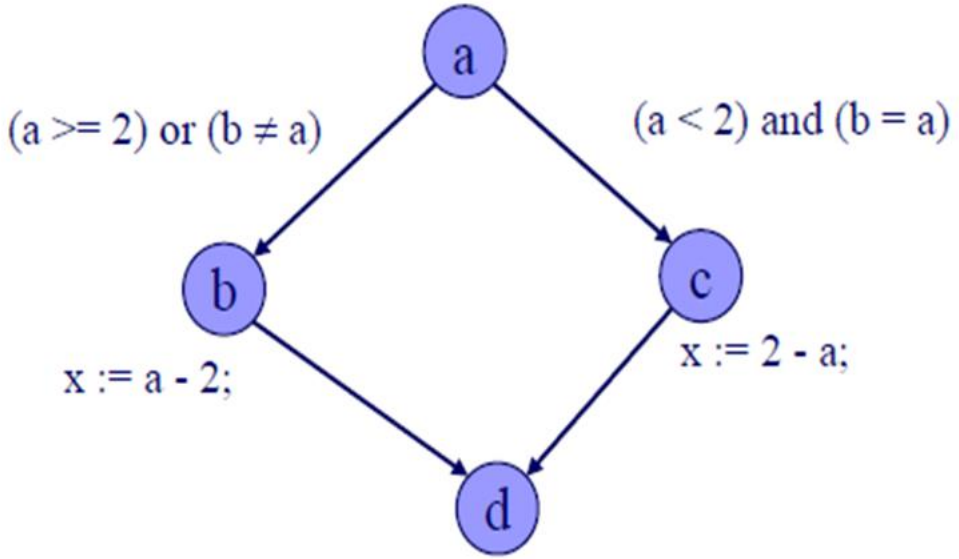
Edge Coverage : adges = { (0,1), (0, 2), (1, 2) }

Test Paths = [0 1 2]
[0 2]

Case of multiple condition coverage (1)

Example:

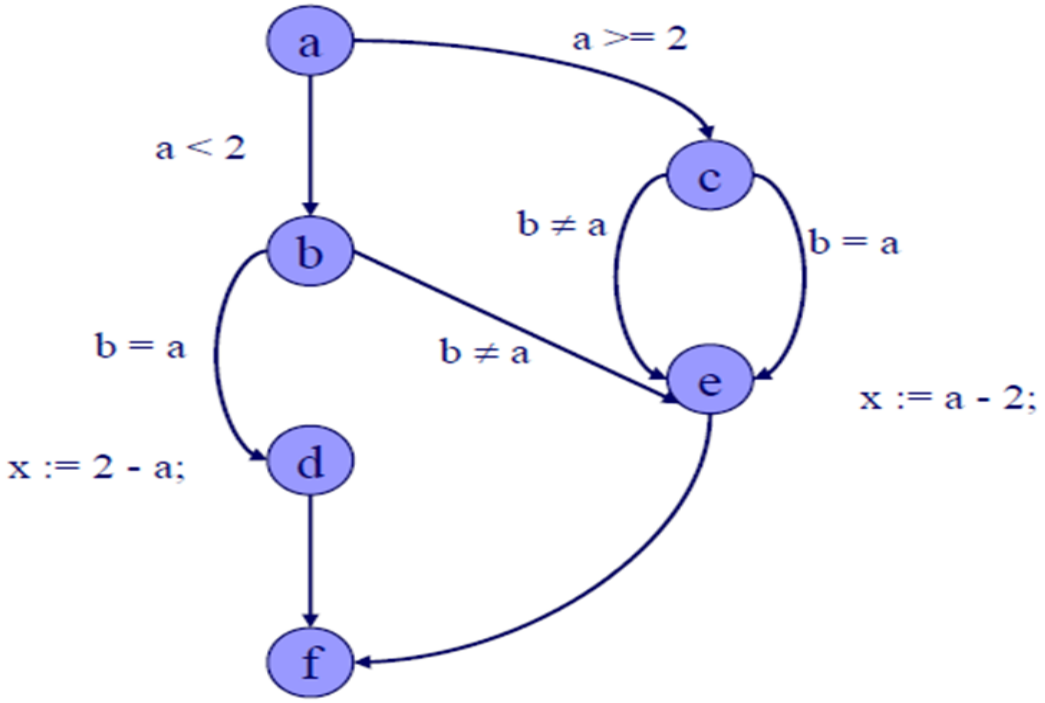
```
if ((a < 2) and (b = a))
then x := 2 - a
else x := a - 2
```



The test case TC1 = {a=b=1}, TC2 = {a=b=3},
Satisfy the criteria all edges without covering all possible decisions –
e.g. TC3 = {a=3, b=2}.

Case of multiple condition coverage (2)

- **Compound Condition Coverage** requires that all combinations of condition values at every branch statement will have been covered, and that every entry point will have been taken, at least once, so multiple conditions must be decomposed



Test cases

- TC1 = {a=b=1}
- TC2 = {a=1, b=0}
- TC3 = {a=3, b=2}
- TC4 = {a=b=3}

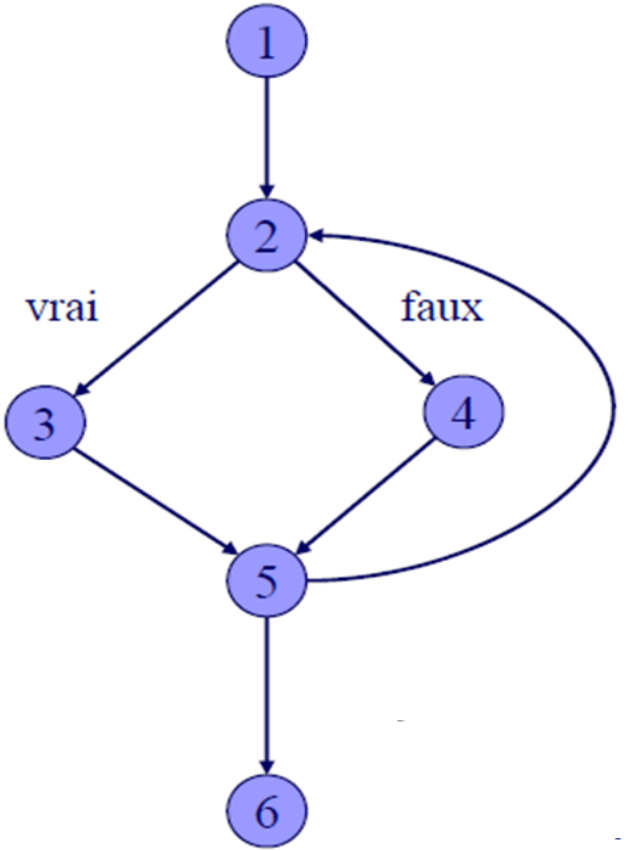
Loop Coverage

The basic minimum number of test cases tends to be two tests:
- Zero times through the loop
- One time through the loop

It's preferred to test the loop:
- Zero times
- One time
- The maximum number of times it is expected to cycle (if you know how many times that is likely to be).

"Exhaustive testing is impossible."

Loop Coverage



Path with testing the loop 0 times

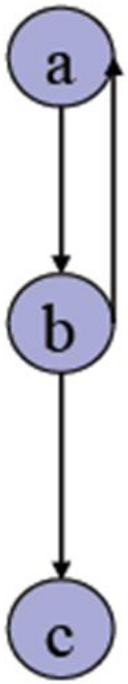
- [1,2,3,5,6]
- [1,2,4,5,6]

Path with testing the loop 1 times

- [1,2,3,5,2,3,5,6]
- [1,2,3,5,2,4,5,6]
- [1,2,4,5,2,3,5,6]
- [1,2,4,5,2,4,5,6]

Coverage of all i-paths

Coverage of all possible paths passing from 0 to i times in each loop of the control graph.



The test cases cover the
[ABC], [ABABC] and [ABABABC] paths
Satisfies the criteria all 2-paths

Independent paths

- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

- When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

Structural test methods :independent : Path coverage

- Design test cases such that: all linearly independent paths in the program are executed at least once.

- When the all-path criteria is met, it implies:
  - the node coverage is met.
  - the branch coverage is met.

paths Coverage

- Path Coverage requires that all program paths will have been traversed at least once.
- when n the maximum number of iterations possible, for the criteria all i-paths, it will execute all possible paths of the control graph flow

Cyclomatic complexity

❖**Cyclomatic** complexity is a measure of the logical complexity of a program.

❖In the context of testing**, cyclomatic complexity** indicates the maximum number of independent complete paths, which gives the number **of test cases** needed to ensure good code coverage.
❖Minimum number of test cases = The number of independent paths for a graph=
Nb adges – Nb nodes + 2

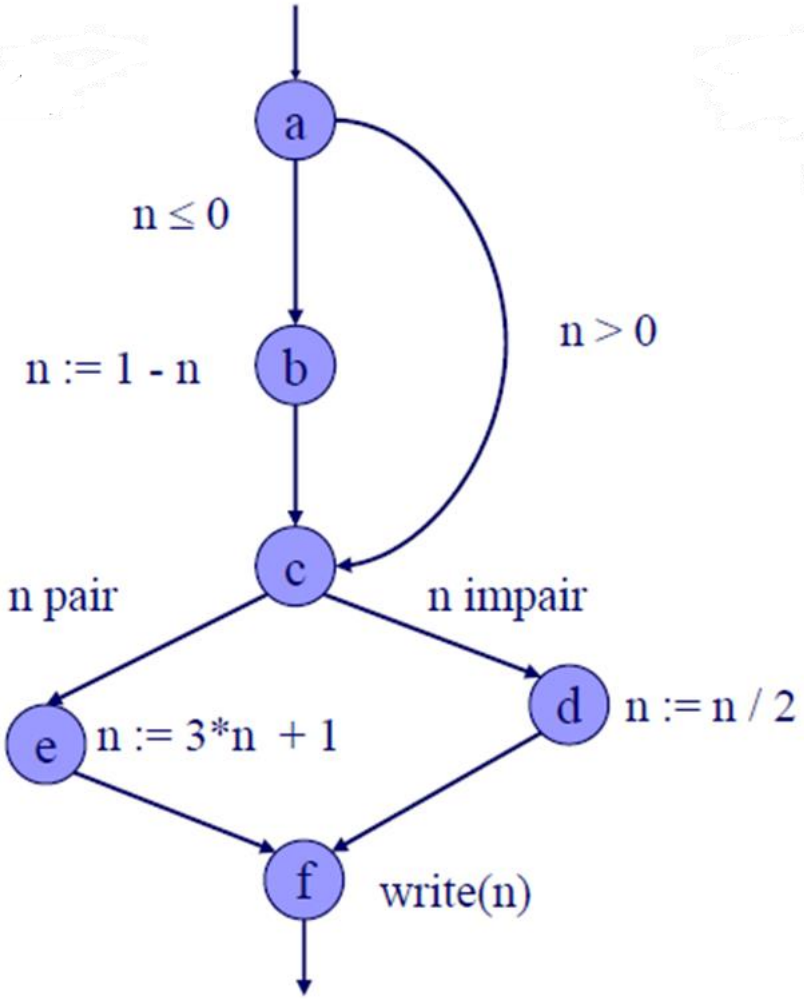
Structural Test Methods - Exercise

```
If n ≤ 0 then n := 1-n
     end;
If n pair
              then n := n / 2
              else n := 3*n + 1
              end ;
Write(n);
```

- Using the code draw the corresponding control flow graph
- Determine test cases satisfied the following criterion:

- **All-Nodes** coverage.
- **All-Edges** coverage ;
- **Independen**t paths coverage

Structural Test Methods - Solusion



- All-Nodes
  - » $n = 0, n = -1$
- All-Edges
  - » $n = 2, n = -2$
- Independent paths
  - » $n = -1, n = -2, n = 1, n = 2$