Software Testing and Reliability

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

Lecture 4

White-box Testing (control-flow coverage)

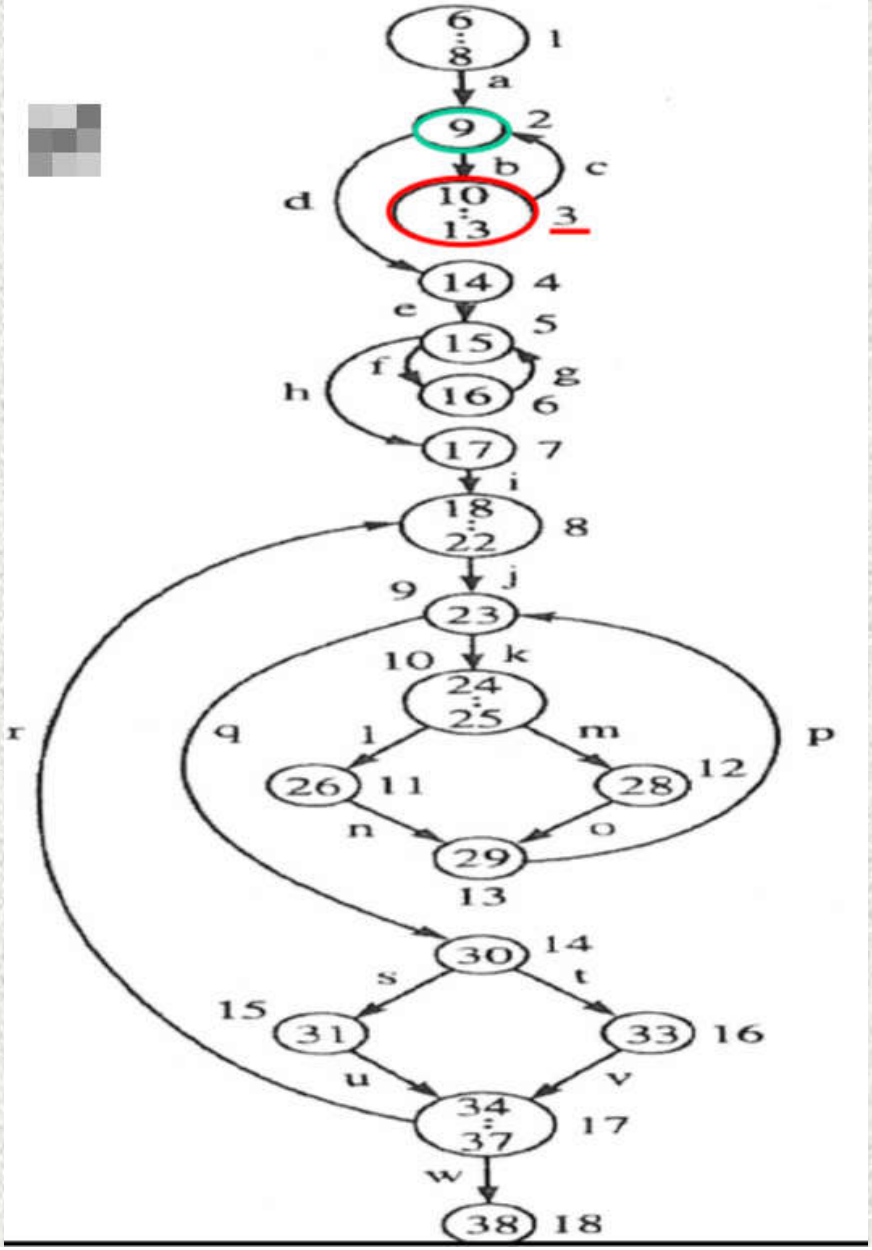
Program under Study

```
1   program example(input, output);
2   var a : array[1 . . 20] of char;
3       x,i: integer;
4       c, response: char;
5       found: boolean;
6   begin
7     writeln('Input an integer between 1 and 20');
8     readln(x);
9    while (x < 1) or (x > 20) do
10    begin
11      writeln('Input an integer between 1 and 20');
12      readln(x)
13    end;
14    writeln('input ',x,' characters');
15    for i := 1 to x do
16       read(a[i]);
17    readln;
18    repeat
19      writeln('Input character to be searched for: ');
```

Program under Study (continued)

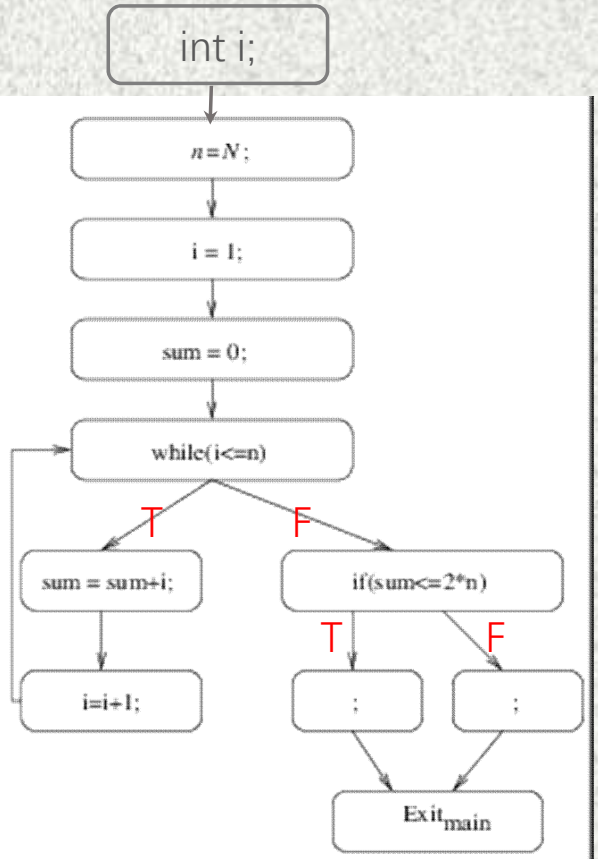
```
20      readln(c);
21      found := FALSE;
22      i := 1;
23      while (not(found)) and (i <= x) do
24      begin
25        if a[i] = c then
26          found := TRUE
27      else
28          i := i +1
29      end;
30      if found then
31        writeln('Character ',c,' appears at position',i)
32      else
33        writeln('Character ',c,' does not occur in the string');
34      writeln;
35      writeln('Search for another character? [y/n]');
36      readln(response);
37    until (response = 'n') or (response = 'N');
38  end.
```

Directed Graph of the Program



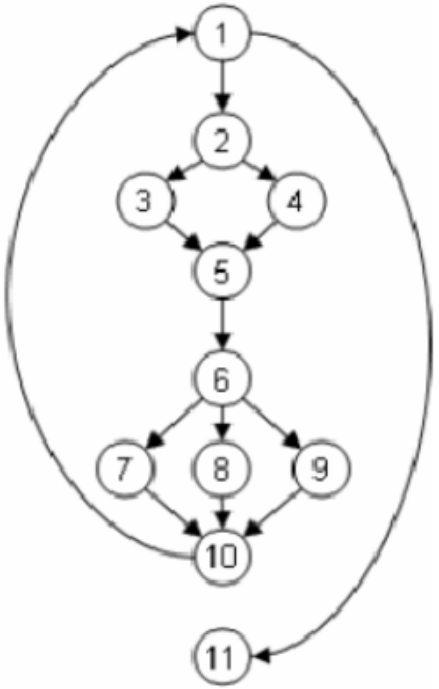
Control flow graph (example)

```
int n, sum;
main(){
  int i;
  n = N;
  i = 1;
  sum = 0;
  while ( i<=n ){
    sum = sum + i;
    i = i + 1;
  }
  if ( sum < 2*n ){
   ERROR: ;
  } else {
   ;
  }
}
```



Control flow graph (example)

Node	Statement
(1)	`while(x<100){`
(2)	`  if (a[x] % 2 == 0) {`
(3)	`      parity =  0;`
	`   }`
	`   else {`
(4)	`      parity =  1;`
(5)	`   }`
(6)	`   switch(parity){`
	`      case 0:`
(7)	`        println( "a[" + i + "] is even");`
	`      case 1:`
(8)	`        println( "a[" + i + "] is odd");`
	`      default:`
(9)	`        println( "Unexpected error");`
	`    }`
(10)	`   x++;`
	`}`
(11)	`p = true;`



Node, Edge and Branch

- Node
  - -- A block of statements
  - -- For example,
    - -- Each node is labelled as a number, that is, 1, 2, 3, …in the previous slide
    - -- Node 3 is associated with 10-13th lines of code
- Edge –link between nodes
- Branch
  - -- An edge, associated with the true or false branch of a decision node(or called predicate)
  - -- For example,
    - -- Node 2 (9th line of code) is a decision node, which has two out-going edges (b and d), -- b and d are called the branches of Node 2.
    - -- Each edge is labelled as a character, that is, a, b, c, … in the previous slide

Basic Concepts

- Relational expression

  -- An expression that returns true or false

- Simple predicate –only consisting of a single relational expression

  -- Linear simple predicate

  -- a_1x_1 + ….+ a_nx_n ROP k,

  where x_i is a variable, and ai and k are constants

- Compound predicate

Basic Concepts (continued)

- Consider the following code

```
public void methodZ()
{
    INPUT X and Y
    Define Z = X+Y
    If (X> 0)     { /*do Task A*/ }
    If (Y>0)      { /*do Task B*/ }
    If (Z < 0)    { /*do Task C*/ }
}
```

Convert (Z < 0) to (X+Y <0). This process is called "*predicate interpretation*" to ensure that the predicate is made up by the input variables rather than the temporary variables.

Basic Concepts (continued)

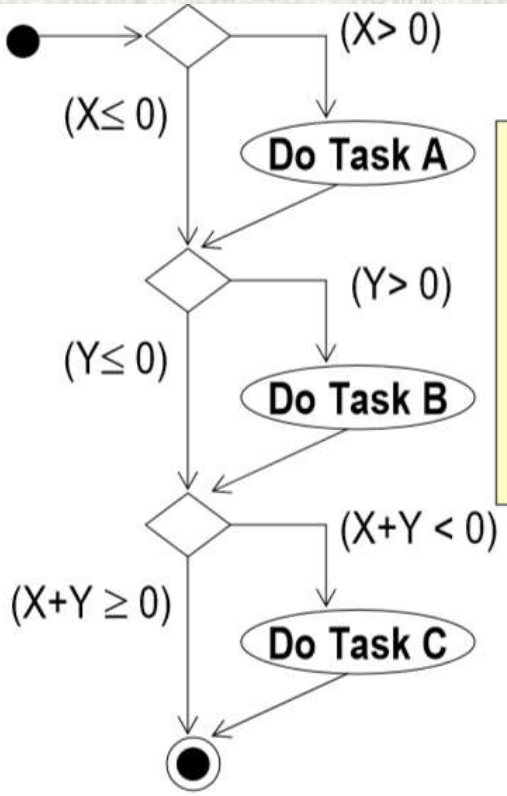
- Path condition

  -- Conjunction of predicate interpretations along a path

- Some path conditions can never be satisfied. These paths are called **infeasible paths**.

  -- In this case, no test cases can be generated for an infeasible path

Basic Concepts (continued)



Path 1 (X> 0) (Y>0)(X+Y < 0)	**Infeasible**
Path 2(X> 0) (Y>0)(X+Y ≥ 0)	Feasible
Path 3(X> 0) (Y≤0)(X+Y < 0)	Feasible
Path 4(X> 0) (Y≤0)(X+Y ≥ 0)	Feasible
Path 5(X≤ 0) (Y>0)(X+Y < 0)	Feasible
Path 6(X≤ 0) (Y>0)(X+Y ≥ 0)	Feasible
Path 7(X≤ 0) (Y≤0)(X+Y < 0)	Feasible
Path 8(X≤ 0) (Y≤0)(X+Y ≥ 0)	Feasible

Basic Concepts (continued)

- Executable path

  -- If a path can be executed by at least one input, the path is called executable path

- Input subdomain

  -- A set of inputs satisfying a path condition

Coverage Testing

- Intuition?
  - -- If a certain part of the program is not covered, we cannot know whether there are faults inside that part.
- Coverage criteria
  - -- Control-flow coverage
  - -- Data-flow coverage

Procedure of coverage testing

1. Given a coverage criterion, list out all the items to be covered

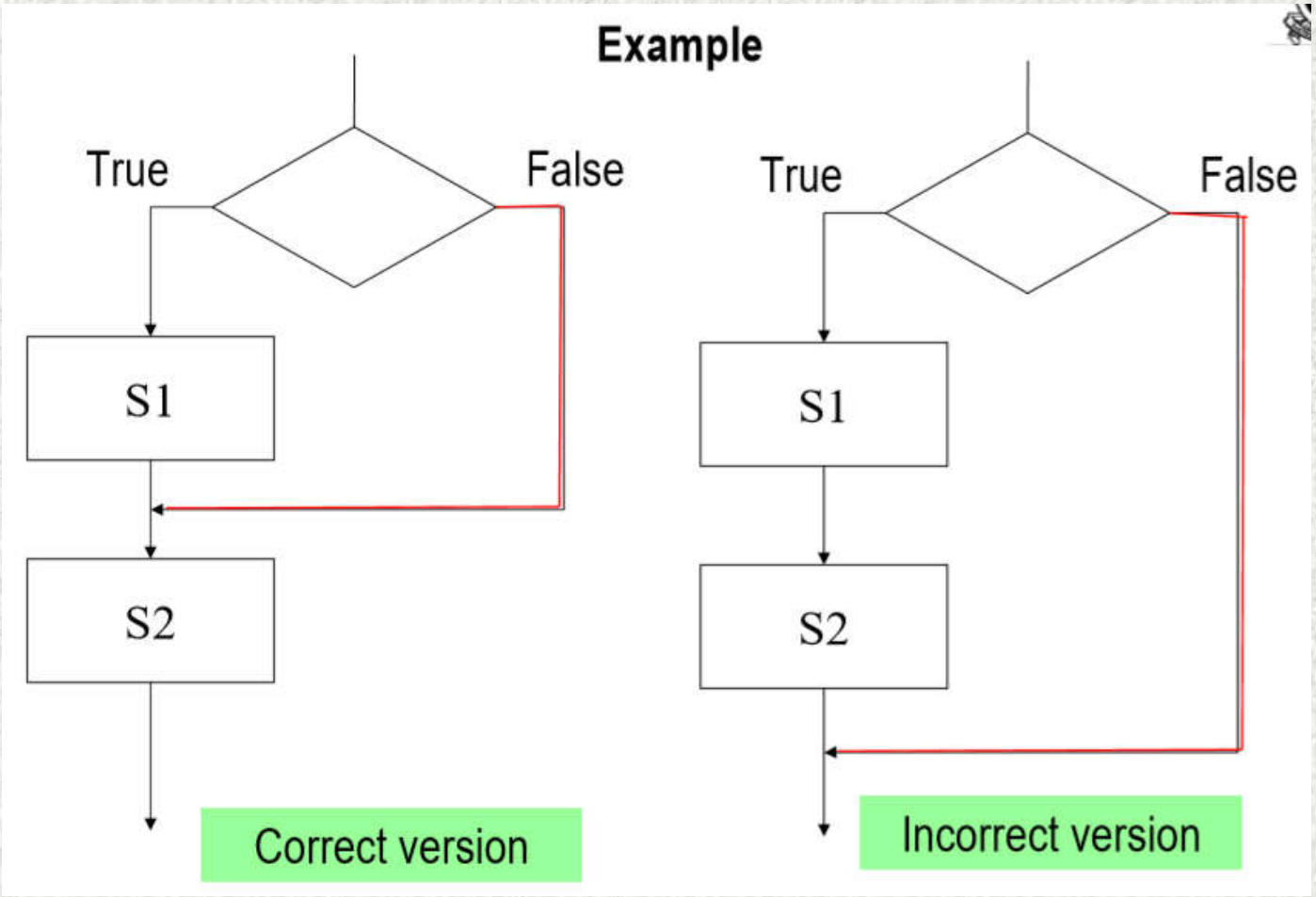
2. Find test cases to execute each item at least once

Control-flow Coverage

- Statement coverage
  - -- Execute every statement at least once
    - -- For example, cover all nodes (1, 2, …, 18) in Slide 4

- Branch coverage
  - -- Execute every branch at least once
    - -- For example, cover all branches (b, d, f, h, k, q, l, m, s, t, wand r) in Slide 4
      - -- Imply coverage of all edges
  - -- **Imply statement coverage**

Control-flow Coverage (continued)



Control-flow Coverage (continued)

In this case,

- statement coverage testing **only** covers the true branch
- branch coverage testing **covers** all statements and all branches
- Branch coverage can detect the fault, but the statement coverage cannot

Control-flow Coverage (continued)

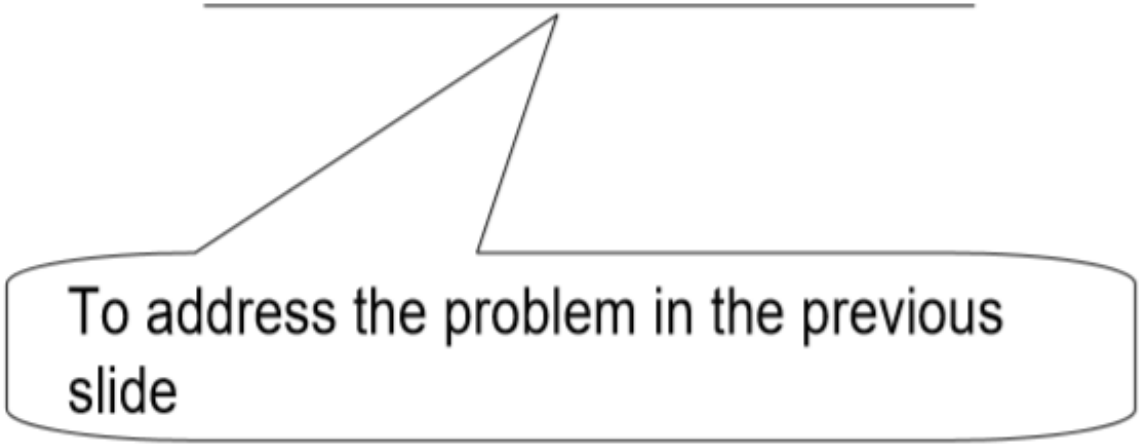
- Condition coverage
  - Execute all possible outcomes (TRUE or FALSE) of every condition in a decision at least once
  - **Do not imply branch coverage**
  - Example: WHILE (x < 10 OR x > 200) DO

Once reaching 100% condition coverage, other test cases will be ignored. As a result, the FALSE branch will not be tested

Test input value	Condition 1 x < 10	Condition 2 x > 200	Branch x < 10 OR x > 200
x < 10	TRUE √	FALSE √	TRUE
x > 200	FALSE √	TRUE √	TRUE
200 >= x >=10	FALSE	FALSE	FALSE

Control-flow Coverage (continued)

- Decision-condition coverage
  - Execute all possible outcomes of every condition in a decision at least once
  - Execute all possible outcomes of a decision at least once



Control-flow Coverage (continued)

- Multiple-condition coverage

  -- Execute all possible combinations of condition outcomes in a decision

    -- Remark: Some combinations may never be satisfied, for example, we cannot have (x < 10 = TRUE) and (x > 200 = TRUE) both condition outcomes at the same time

  -- **Imply decision-condition coverage**

Control-flow Coverage (continued)

- Path coverage
  - -- Execute every path at least once
  - -- **Imply all the coverage criteria**
  - -- Problem
    - -- When the software contains loops, it has infinite number of paths – then 100% path coverage becomes infeasible.
    - -- Solution
      - -- Group paths into equivalence classes (see the next slide)

Equivalence classes of paths

- Two paths are considered equivalent if they differ only in the number of iterations
- Two classes
  - -- **Zero** loop traversal
  - -- **At least one** loop traversals

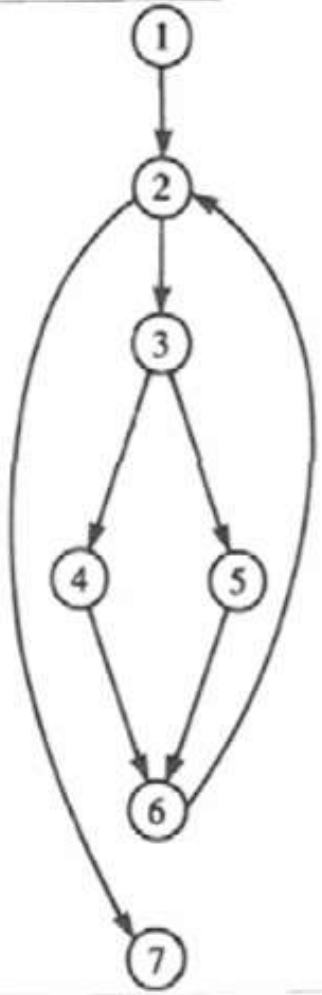
Coverage for equivalence classes of paths

Some notations

- · (dot)
  -- represents sequences
- **+** (plus)
  -- represents selections
- ***** (star)
  -- represents iterations
- **0** (zero)
  -- represents NULL

Coverage for equivalence classes of paths (continued)

```
      .
1     .
2     while (condition1) do
      begin
3        if (condition2) then
            .
4           .
         else
            .
5           .
6     end
      .
7     .
```



while loop example

$1 \cdot 2 \cdot (3 \cdot (4+5) \cdot 6 \cdot 2)^* \cdot 7$

$1 \cdot 2 \cdot ((3 \cdot (4+5) \cdot 6 \cdot 2)+0) \cdot 7$

Translate each digit into 1

$1 \cdot 1 \cdot ((1 \cdot (1+1) \cdot 1 \cdot 1)+1) \cdot 1$

3 paths

to be covered

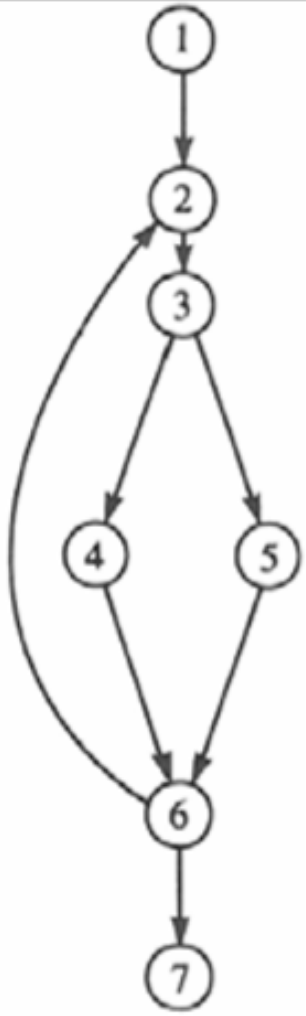
Coverage for equivalence classes of paths (continued)

3 paths for the above while loop example

1. 1–2–7
2. 1–2–3–4–6–2–7
3. 1–2–3–5–6–2–7

Coverage for equivalence classes of paths (continued)

```
      .
1     .
2     repeat
3        if (condition2) then
               .
4              .
         else
               .
5              .
6     until (condition1)
      .
7     .
```



repeat loop example

$1 \cdot 2 \cdot 3 \cdot (4+5) \cdot 6(\cdot 2 \cdot 3 \cdot (4+5) \cdot 6)^* \cdot 7$

⇩

$1 \cdot 2 \cdot 3 \cdot (4+5) \cdot 6((\cdot 2 \cdot 3 \cdot (4+5) \cdot 6)+0) \cdot 7$

⇩

$1 \cdot 1 \cdot 1 \cdot (1+1) \cdot 1((\cdot 1 \cdot 1 \cdot (1+1) \cdot 1)+1) \cdot 1$

⇩

6 paths

to be covered

Coverage for equivalence classes of paths (continued)

6 paths for the above repeat loop example

1. 1–2–3–4–6–7
2. 1–2–3–5–6–7
3. 1–2–3–4–6–2–3–4–6–7
4. 1–2–3–4–6–2–3–5–6–7
5. 1–2–3–5–6–2–3–4–6–7
6. 1–2–3–5–6–2–3–5–6–7

How do we know that an item has been executed?

- By program instrumentation
  -- Insert some printing statements to trace the program execution

Can be done either after or in parallel with compilation

Applications

- To measure the thoroughness of a test
- To control the behaviour of a test
- To generate symbolic traces
- To be used in dynamic data flow analysis

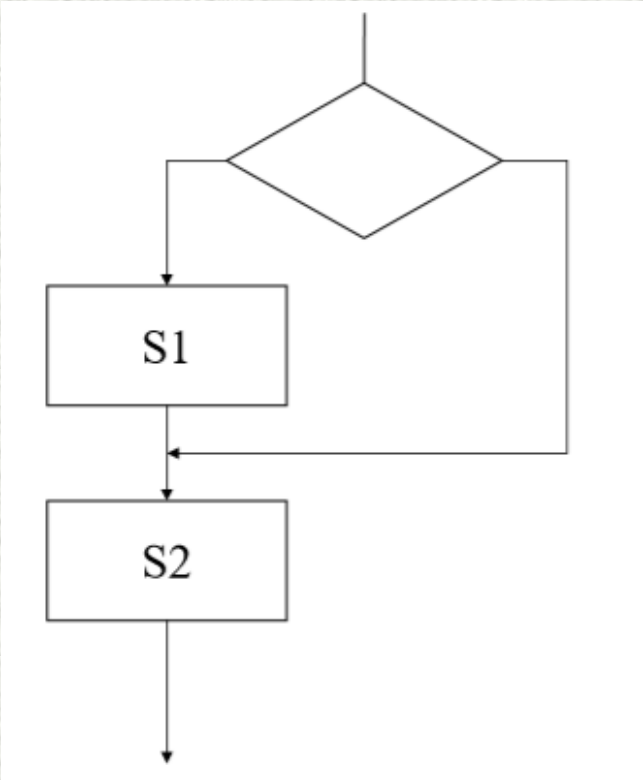
Example: Coverage Testing

- Intuition?
- Coverage criteria
  - Control-flow coverage
  - Data-flow coverage

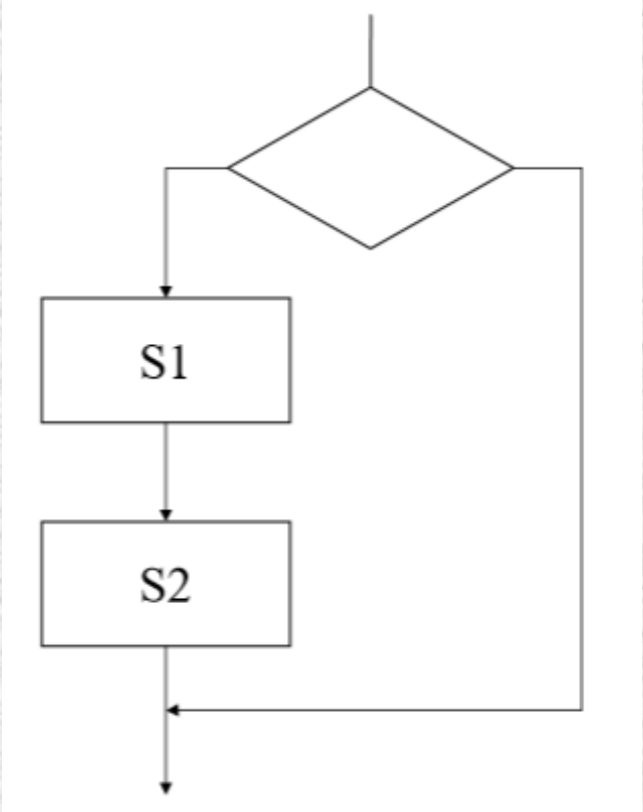
Control-flow Coverage

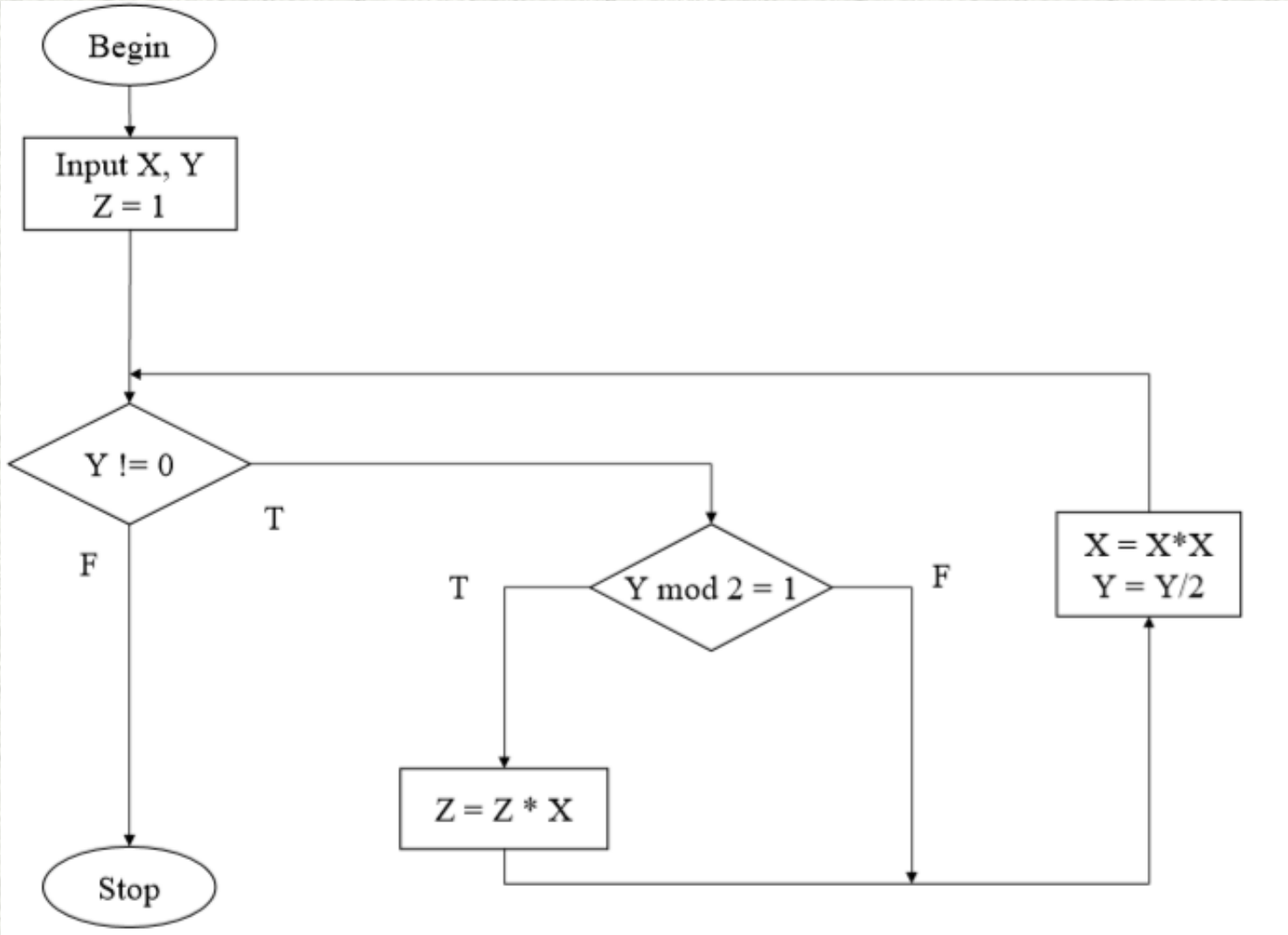
- Every statement be executed at least once.
- Every branch be executed at least once.
- …..
- ….
- Every path be executed at least once.

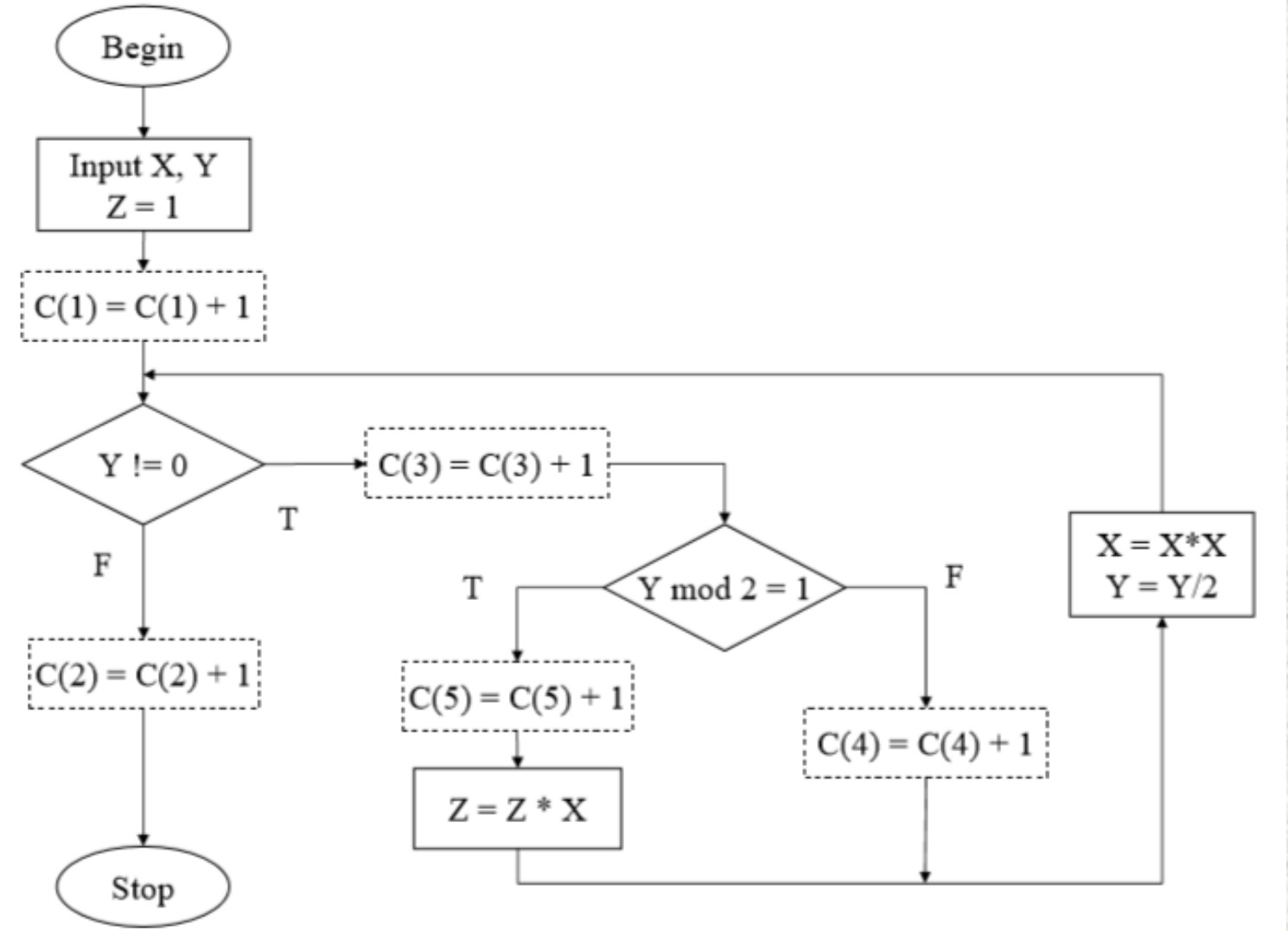
Example



Example (continued)




Begin
Input X, Y
Z = 1
Y != 0
T
F
Y mod 2 = 1
T
F
Z = Z * X
X = X*X
Y = Y/2
Stop


Begin
Input X, Y
Z = 1
C(1) = C(1) + 1
Y != 0
T
F
C(3) = C(3) + 1
Y mod 2 = 1
T
F
C(5) = C(5) + 1
Z = Z * X
C(4) = C(4) + 1
X = X*X
Y = Y/2
C(2) = C(2) + 1
Stop

TEST CASES						
X	Y	C[1]	C[2]	C[3]	C[4]	C[5]
5	0	1	1	0	0	0
10	1	1	1	1	0	1
15	2	1	1	2	1	1
20	3	1	1	3	1	2
25	4	1	1	3	2	1

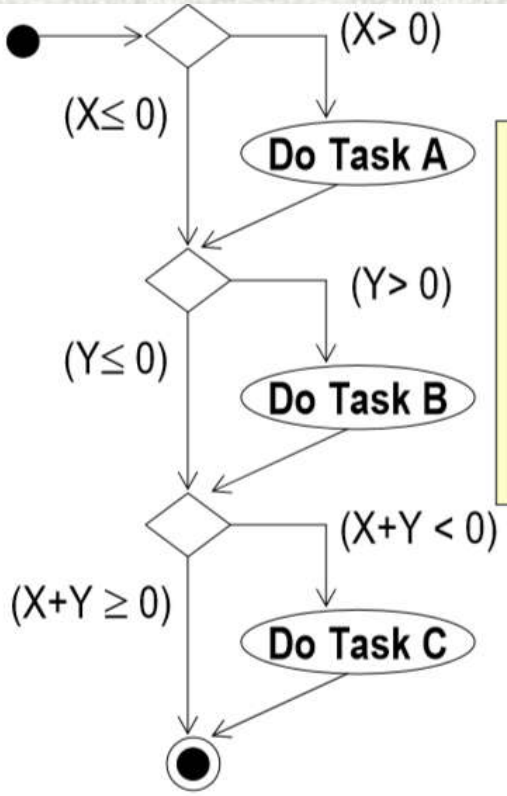
How about data flow analysis?

Two special examples for loop and if structure

How to generate a test case that executes an item?

- It is a search problem
  - -- Normally, there are more than one paths that will reach the item
- The task is to solve ONE of the path conditions related to the item
  - -- Random approach
  - -- Search algorithms, such as the genetic algorithm
  - -- Constraint solver

Or, just solve the path conditions manually



Path 1 (X> 0) (Y>0)(X+Y < 0)	**Infeasible**
Path 2(X> 0) (Y>0)(X+Y ≥ 0)	Feasible
Path 3(X> 0) (Y≤0)(X+Y < 0)	Feasible
Path 4(X> 0) (Y≤0)(X+Y ≥ 0)	Feasible
Path 5(X≤ 0) (Y>0)(X+Y < 0)	Feasible
Path 6(X≤ 0) (Y>0)(X+Y ≥ 0)	Feasible
Path 7(X≤ 0) (Y≤0)(X+Y < 0)	Feasible
Path 8(X≤ 0) (Y≤0)(X+Y ≥ 0)	Feasible