Introduction to code-based software testing

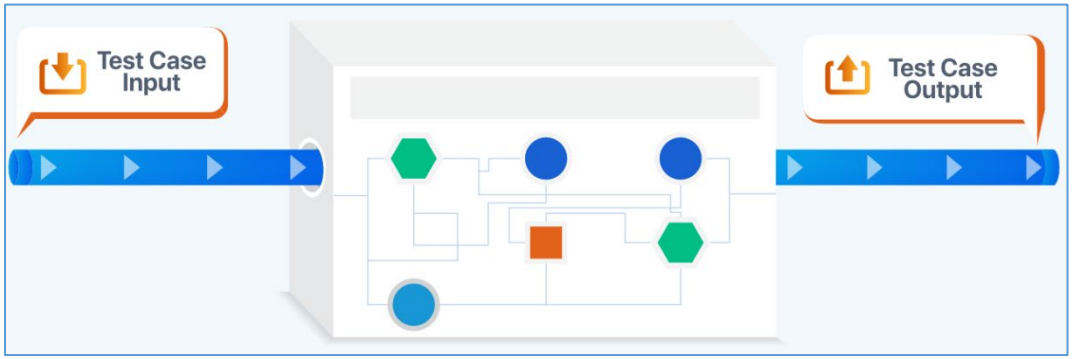
Professor Hossein Saiedian

EECS 348: Software Engineering

November 2023

KU THE UNIVERSITY OF KANSAS

Code-based (structural) testing



- Control coverage
  - Statement testing (node in a flowgraph)
  - Branch/decision testing (edge in a flowgraph)
  - Condition testing
  - Compound condition testing
  - MC/DC (Modified Condition/Decision Coverage) testing
  - Path testing
- Dataflow coverage

Review: A good test case

- A good test case
  - Test case identifier (usually a short name for test management purposes)
  - Name
  - Purpose (e.g., a business rule)
  - Pre-conditions (if any)
  - Inputs
  - Expected outputs
  - Observed (actual) outputs
- A collection of test cases
  - A test case set, a set of test cases, a test set

Test coverage terminology

- Coverage criteria
- Subsumption

Example: jellybean coverage

- **Flavors**: Lemon, Pistachio, Cantaloupe, Pear, Tangerine, Apricot
- **Colors**: Yellow (Lemon, Apricot), Green (Pistachio), Orange (Cantaloupe, Tangerine), White (Pear)
- Possible "taste" coverage criteria:
  - Taste one jellybean of each flavor
  - Taste one jellybean of each color

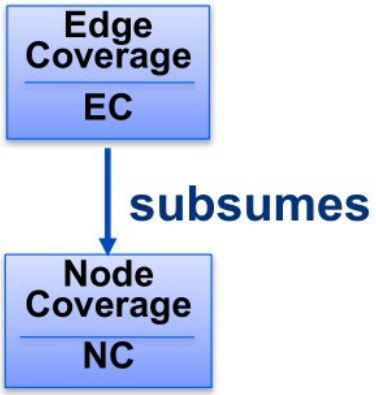
Example: jellybean coverage

- T1 = {three Lemons, one Pistachio, two Cantaloupes, one Pear, one Tangerine, four Apricots }
  - Does test set T1 satisfy the flavor criterion?
- T2 = {One Lemon, two Pistachios, one Pear, three Tangerines }
  - Does test set T2 satisfy the flavor criterion?
  - Does test set T2 satisfy the color criterion?

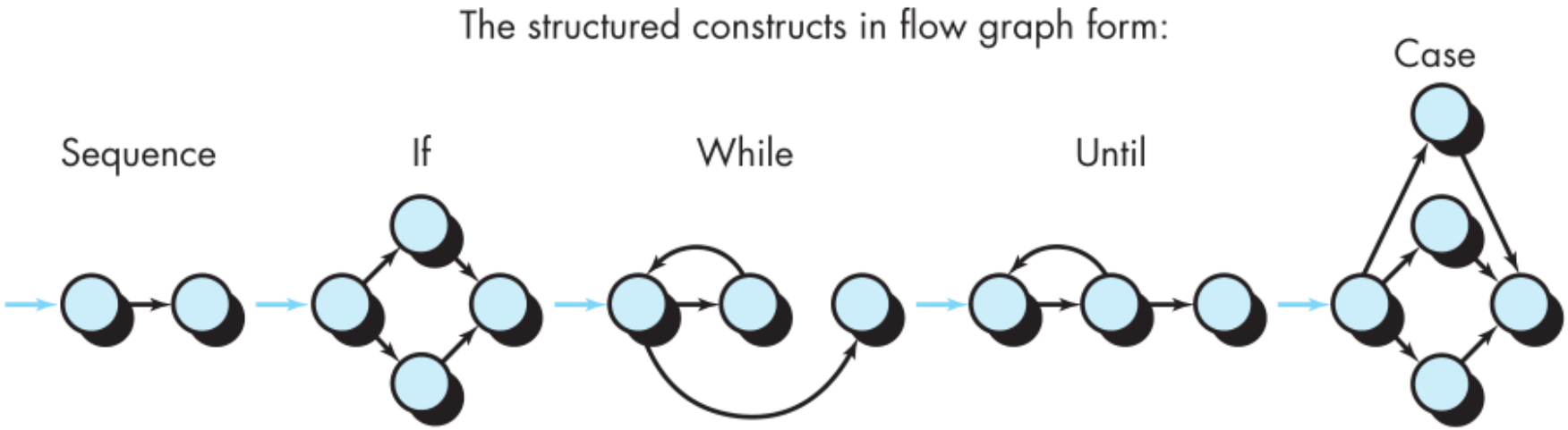
- T3 = {two of each jellybeans)
- T4 = {two Cantaloupes, three Tangerines}
- T5: A jar of jellybeans

Subsumption

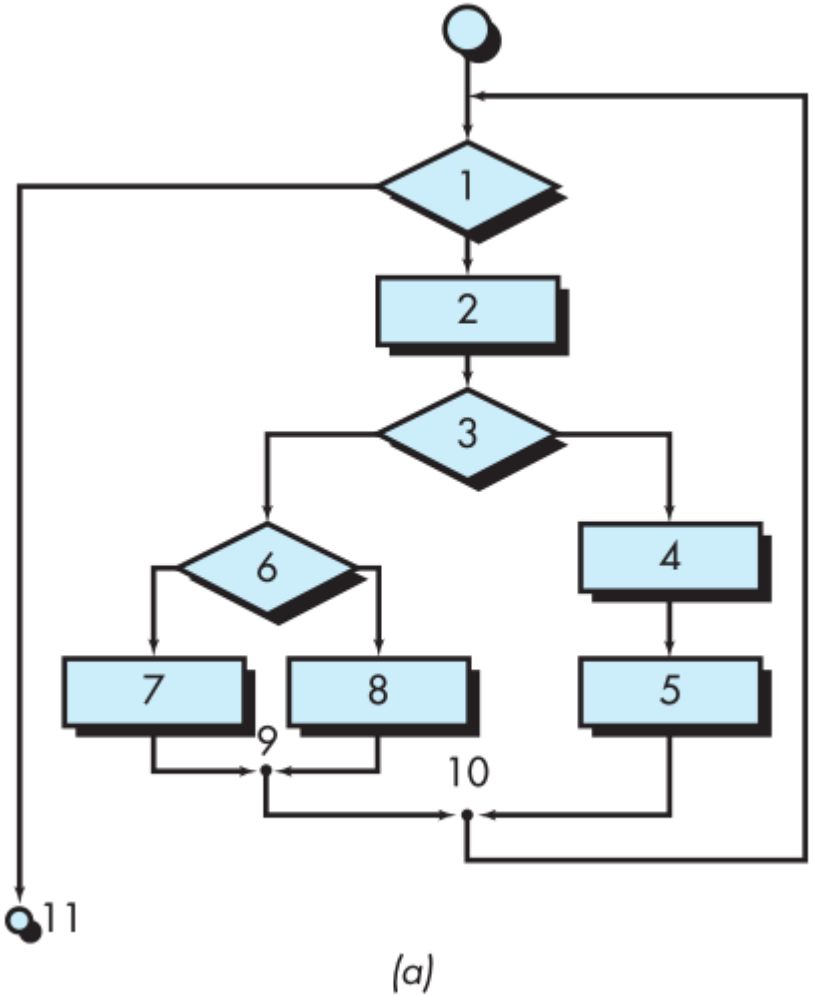
- Criteria subsumption: A test criterion C1 subsumes C2 if and only if every set of test cases that satisfies criterion C1 also satisfies C2
- Must be true for every set of test cases
- ***Example : If a test case set has covered every branch in a program (satisfied the branch criterion), then the test set is guaranteed to also have covered every statement***



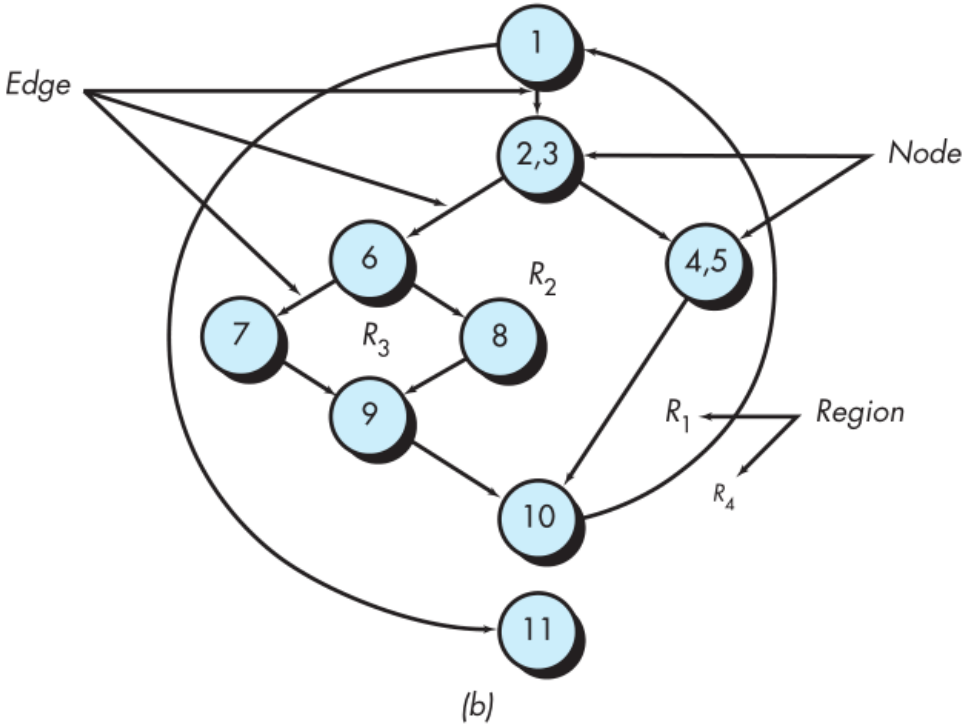
Flowgraph construction



Flowcharts and flowgraphs



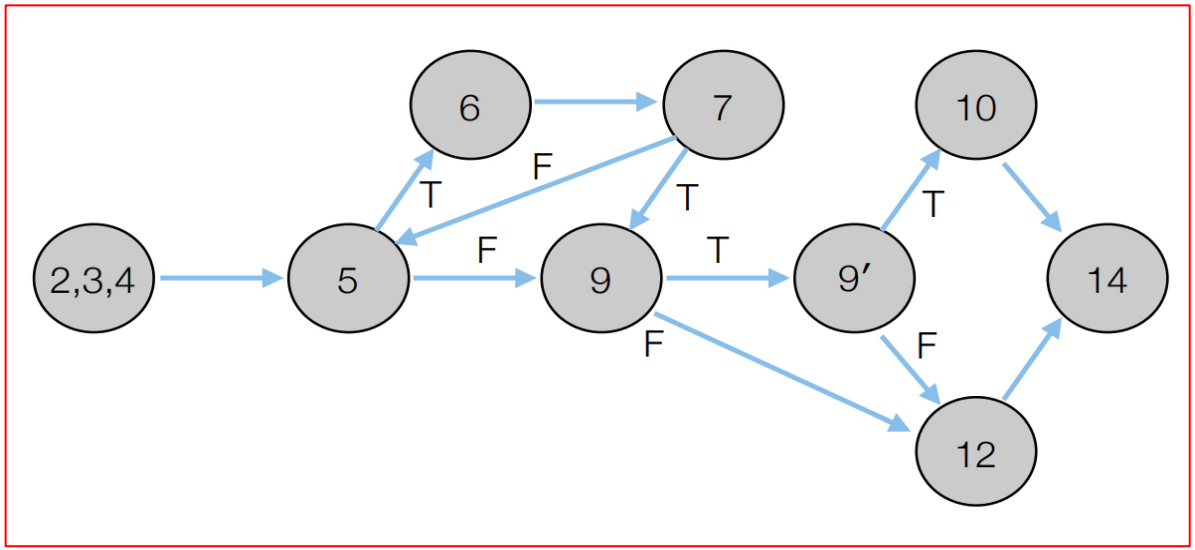
(a)



(b)

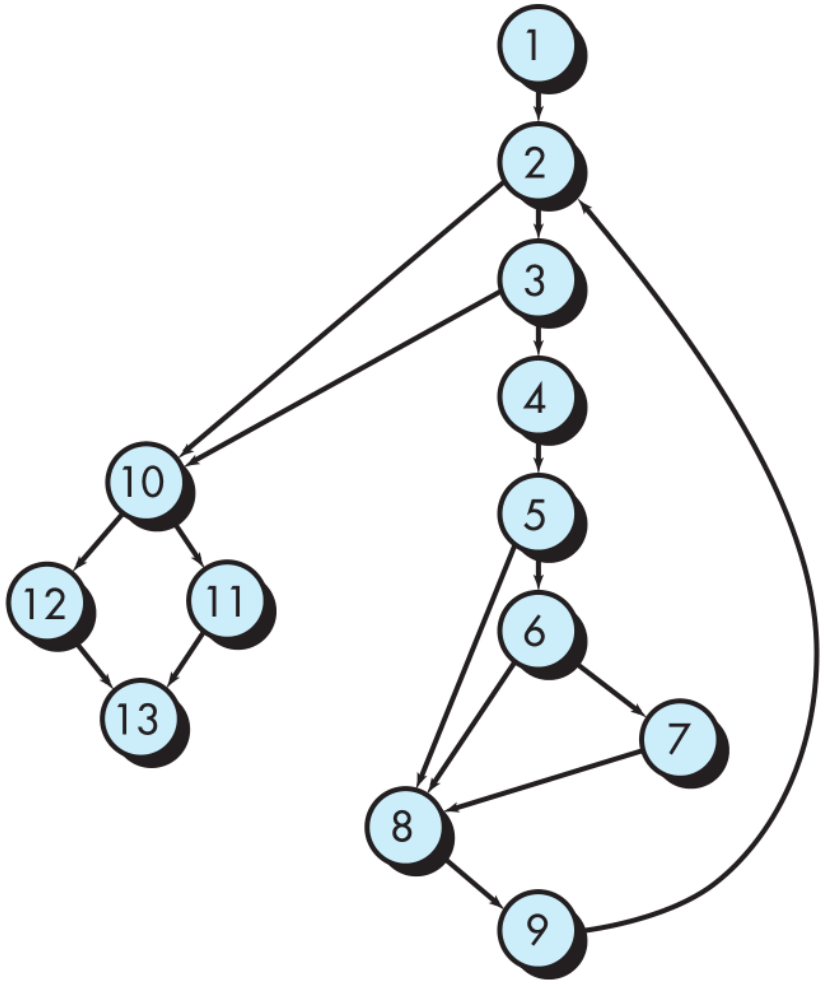
Another example

```
1       function P return INTEGER is
2       begin
3          X, Y: INTEGER;
4          READ(X); READ(Y);
5          while (X > 10) loop
6             X := X – 10;
7             exit when X = 10;
8          end loop;
9          if (Y < 20 and then X mod 2 = 0) then
10            Y := Y + 20;
11         else
12            Y := Y – 20;
13         end if;
14         return 2 * X + Y;
15      end P;
```



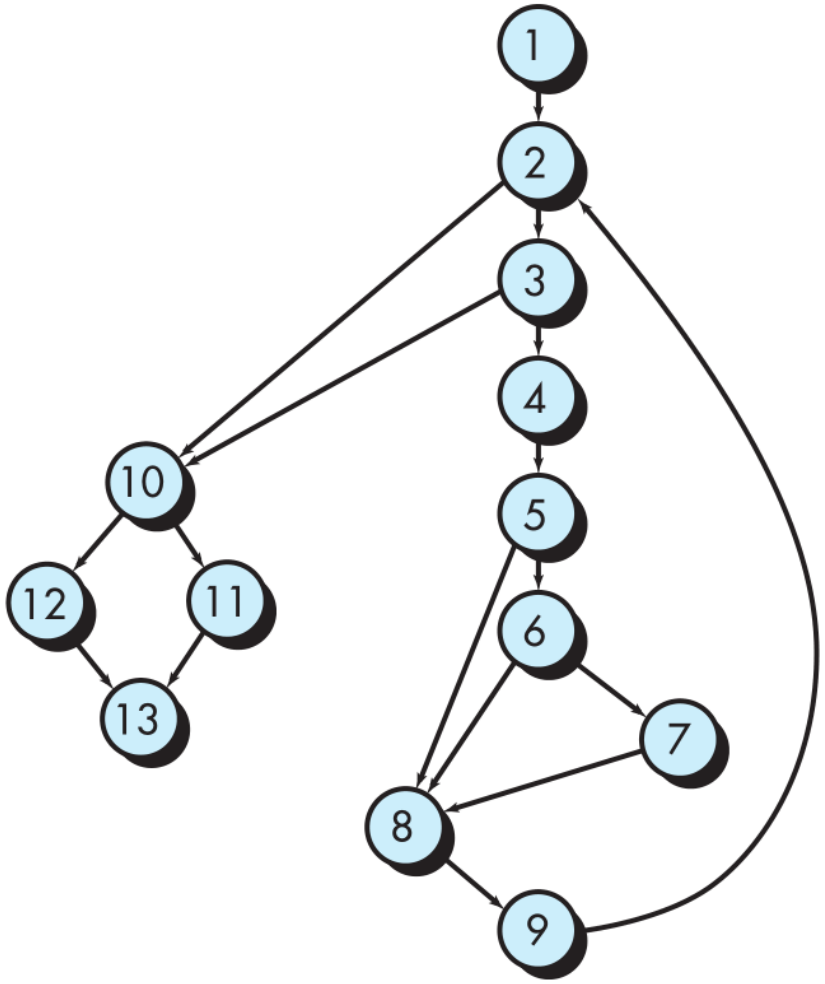
Another example

```
PROCEDURE average;

    *  This procedure computes the average of 100 or fewer
       numbers that lie between bounding values; it also computes the
       sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
         minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;
    sum = 0;
    DO WHILE value[i] <> –999 AND total.input < 100
4       increment total.input by 1;
        IF value[i] > = minimum AND value[i] < = maximum
5           THEN increment total.valid by 1;
7                sum = s sum + value[i]
            ELSE skip
8       ENDIF
        increment i by 1;
9   ENDDO
    IF total.valid > 0
11      THEN average = sum / total.valid;
12      ELSE average = –999;
13  ENDIF
END average
```



How many paths?

- How many paths?
  - V(G) = 17 edges – 13 nodes + 2 = 6
  - V(G) = 6 regions (planes)
- What are the paths?
  - Path 1: 1-2-10-11-13
  - Path 2: 1-2-10-12-13
  - Path 3: 1-2-3-10-11-13
  - Path 4: 1-2-3-4-5-8-9-2-. . .
  - Path 5: 1-2-3-4-5-6-8-9-2-. . .
  - Path 6: 1-2-3-4-5-6-7-8-9-2-. . .

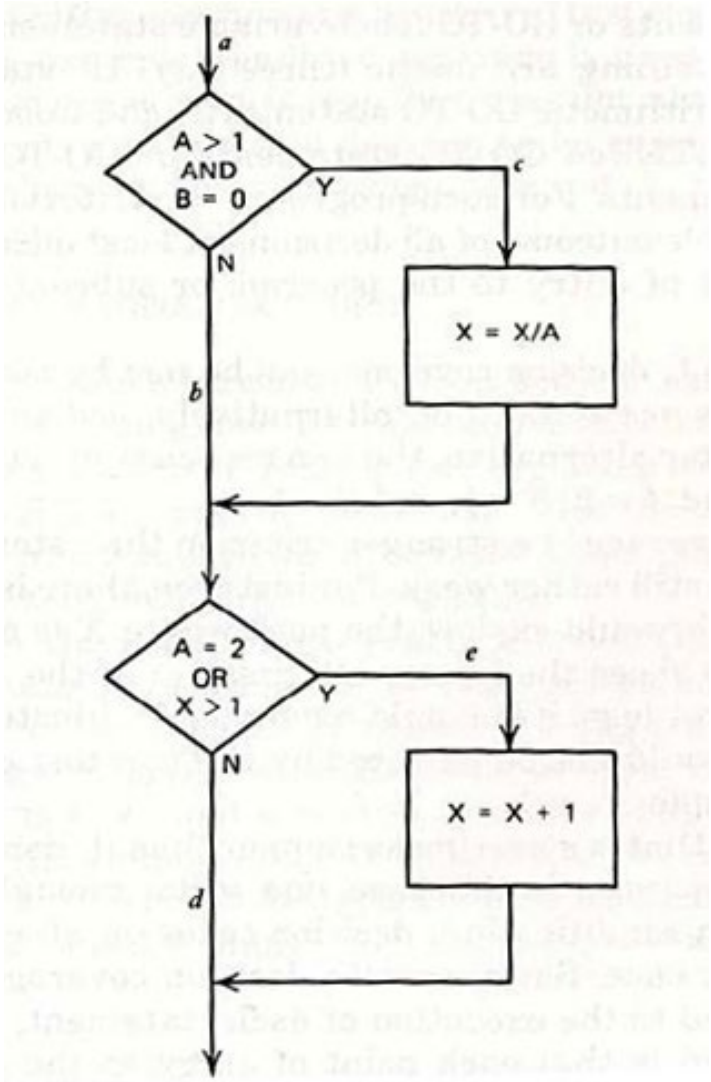


Structural testing examples

KU THE UNIVERSITY OF KANSAS

Code used for some examples

```
If ((A>1) & (B=0)) then
        X=X/A;
END;
If ((A==2) | (X>1)) then
        X=X+1;
END;
```

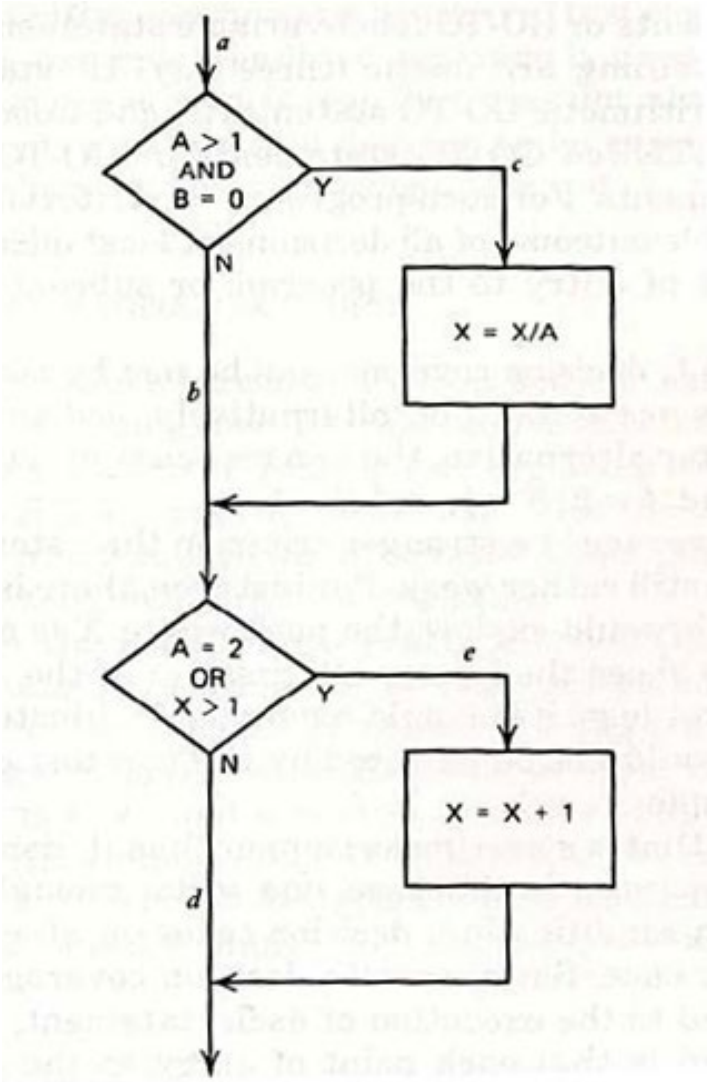


Statement coverage

- Test cases ensure that each statement in a program P is executed
- Other names: line coverage, basic block coverage
- A weak coverage
- Insensitive to some control structures

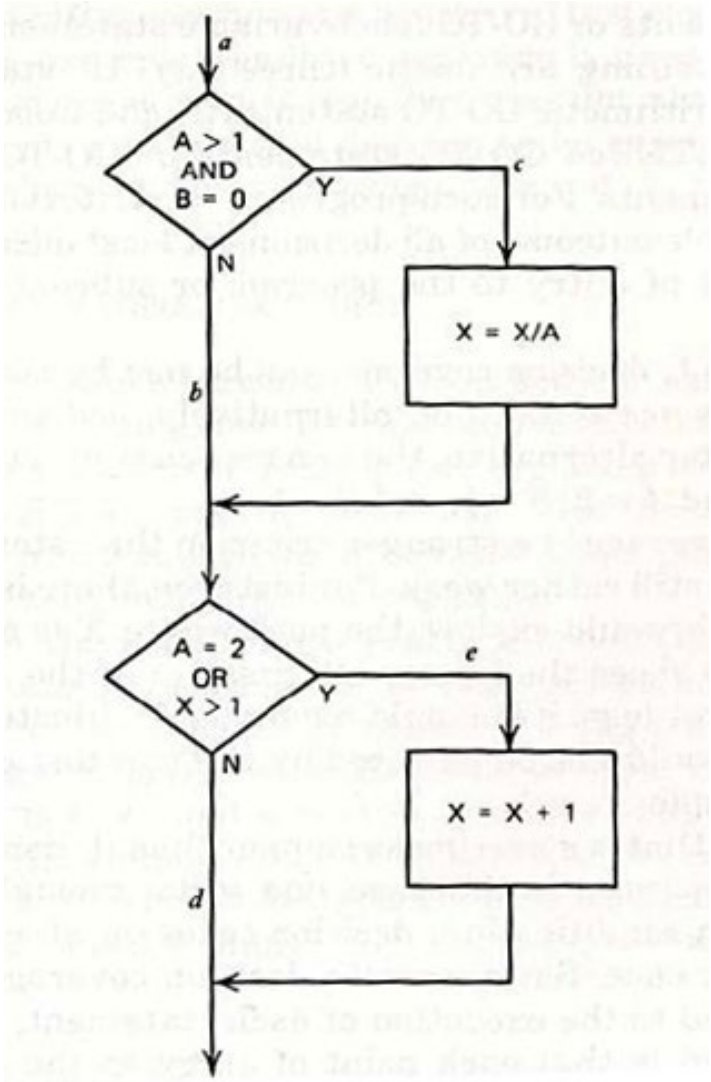
Statement coverage example

- How many test cases?



Statement coverage example

- How many test cases?
- One
  - A=2 and B=0 and X=5 (ace)

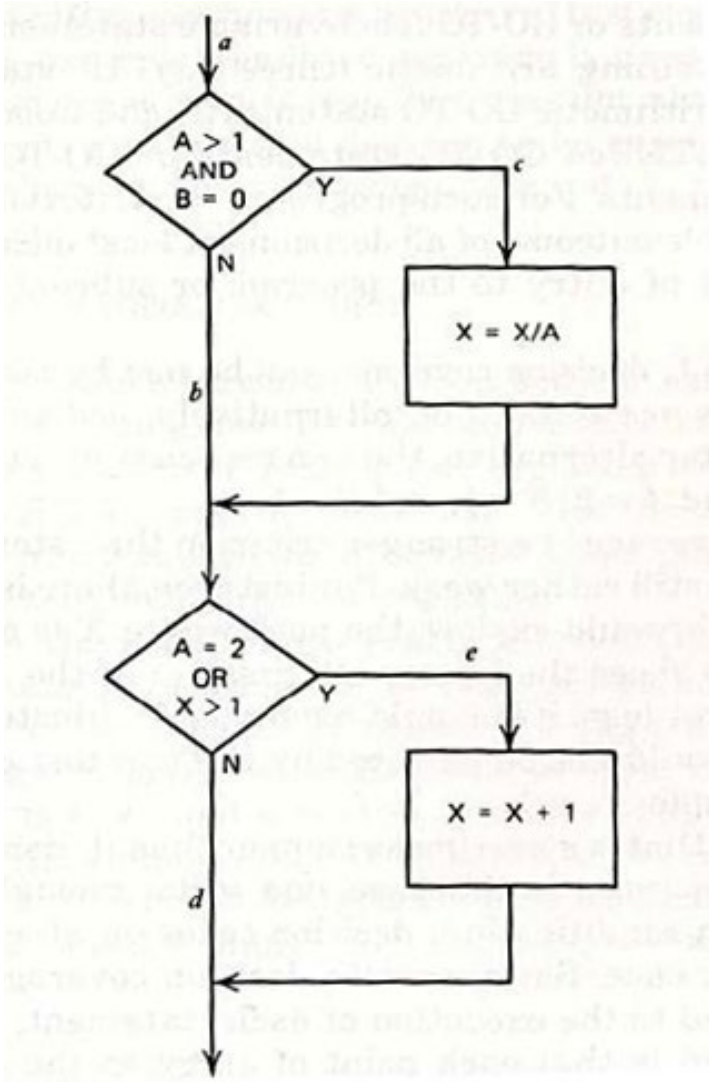


Decision coverage

- Test cases ensure Boolean expressions tested in control structures evaluated to both true and false
  - Also exercises different cases of a "switch" statement
- The entire Boolean expression is considered one true-or-false predicate regardless of whether it contains logical-and or logical-or operators
- Also known as branch or edge coverage, decision-decision-path

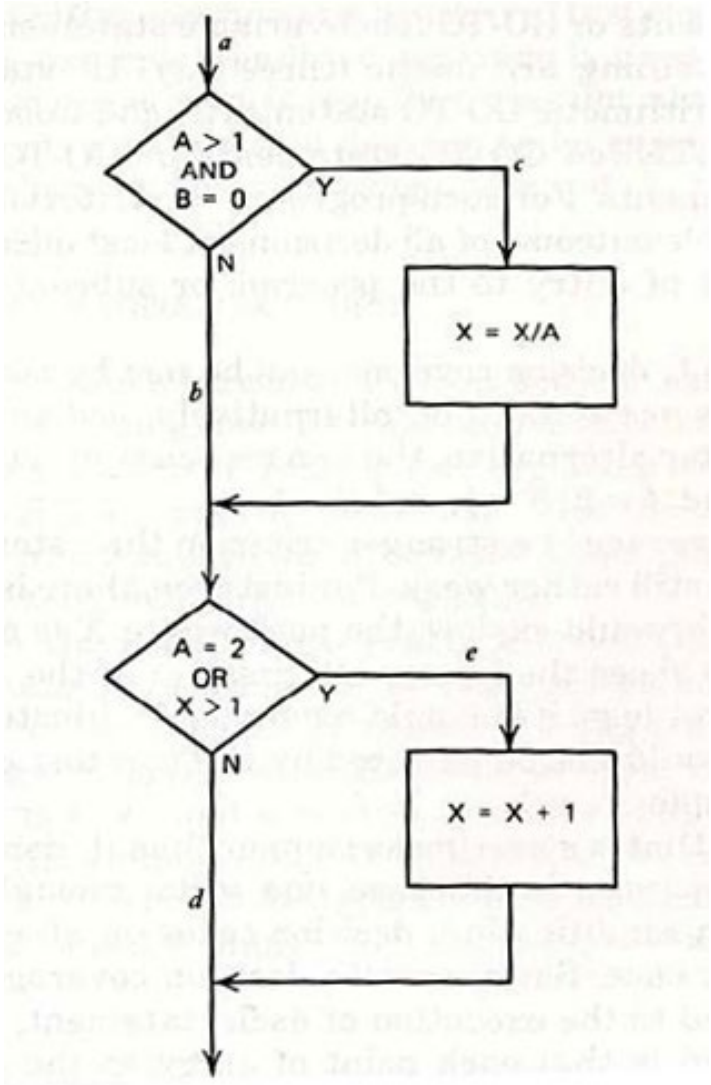
Decision coverage example

- How many test cases?



Decision coverage example

- How many test cases?
- Two
  - A=2 and B=0 x=0 (ace)
  - A=1 and B=1 x=0 (abd)

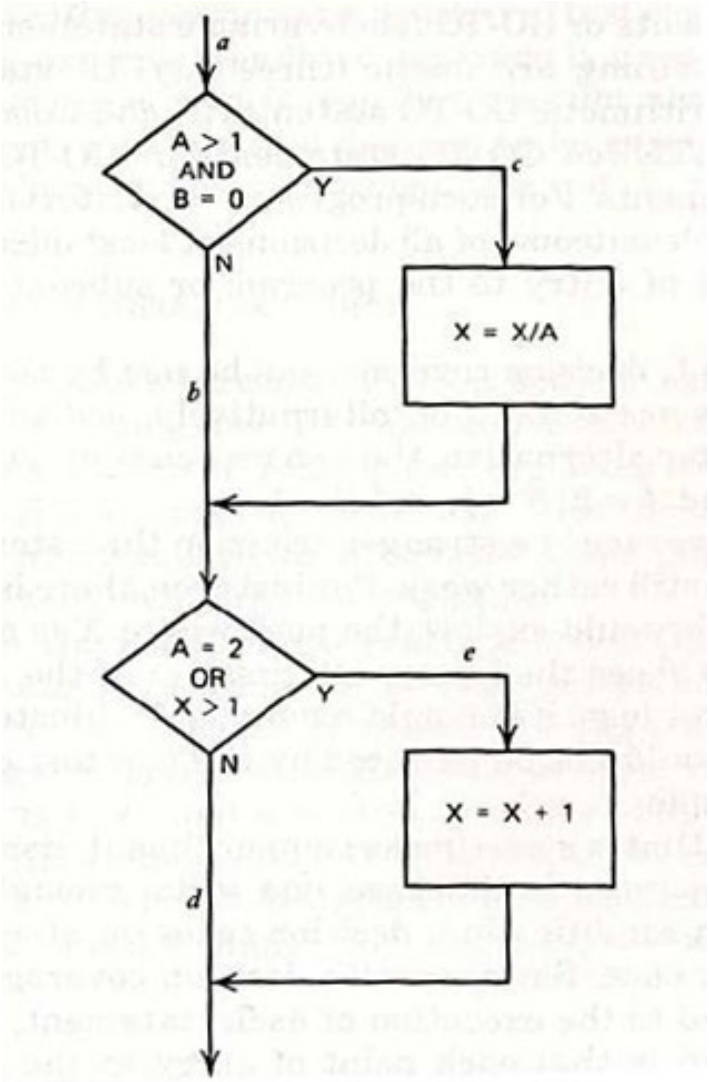


Condition coverage

- Condition coverage reports the true or false outcome of each Boolean sub-expression, separated by logical-and (&&) and logical-or (||) if they occur
  - Condition coverage measures the sub-expressions independently of each other.
- Similar to decision coverage but has better sensitivity to the control flow
- However, full condition coverage does not guarantee full decision coverage

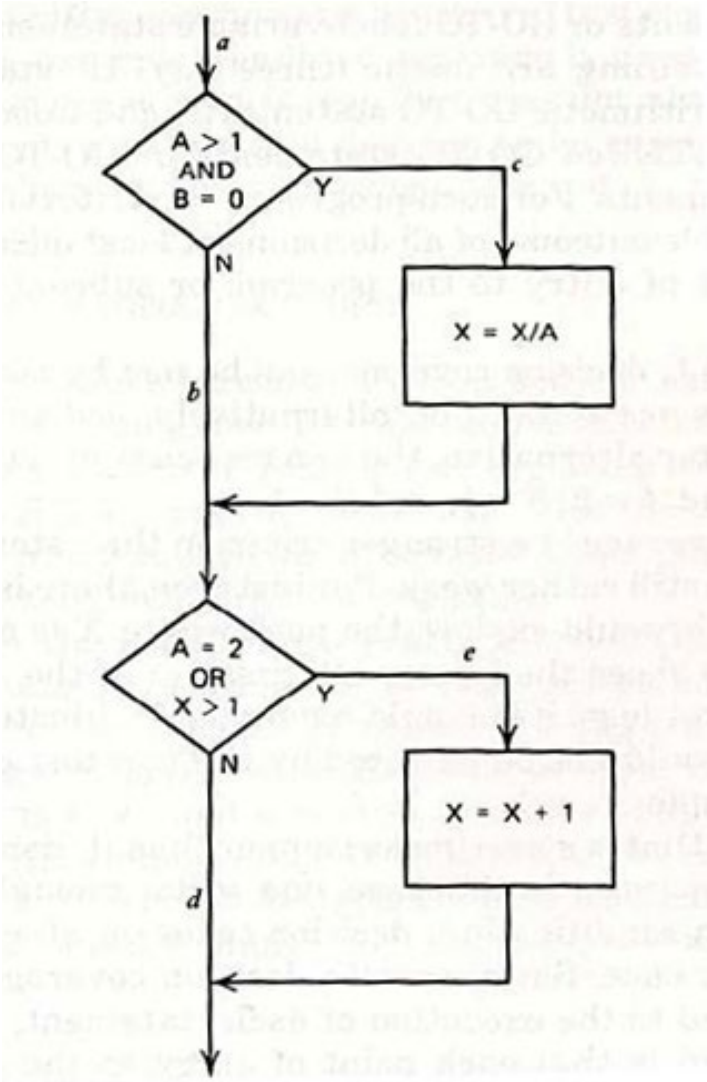
Condition coverage example

- How many test cases?
  - Conditions: A>1, B=0, A=2, X>1



Condition coverage example

- How many test cases?
  - Conditions: A>1, B=0, A=2, X>1
  - How many test cases?
  - A=2, B=0, and X=4 (ace)
  - A=1, B=1, and X=1 (abd)



Condition coverage concern

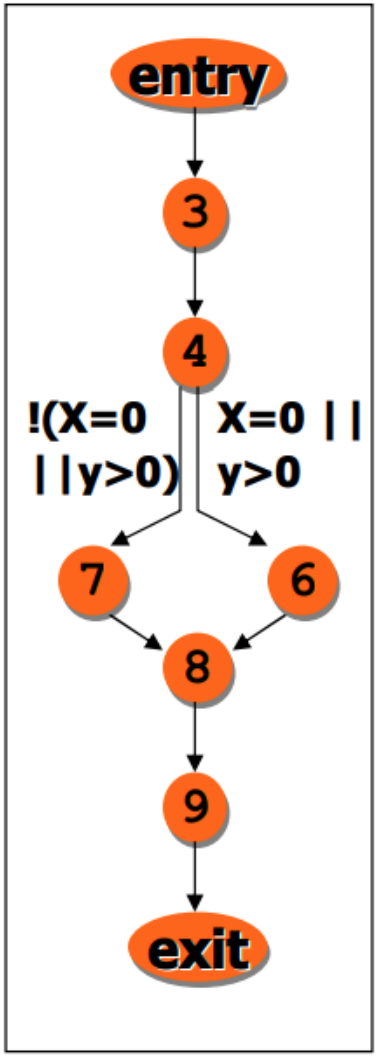
- Consider the following (fails branch coverage)

```
if (a && b) S1 else S2;
```

  - `a==true` and `b==false`
  - `a==false` and `b==true`
  - Both result in exercising S2 (and not S1)
  - However, if one exercises this code with `a` and `b` having *all possible combinations of values*, (known as compound or multi-condition coverage), condition coverage reports full coverage

Condition coverage concern

```
1. void main() {
2.   float x, y;
3.   read(x);
4.   read(y);
5.   if(x==0)||(y>0)
6.     y = y/x;
7.   else x = y+2;
8.   write(x);
9.   write(y);
10.}
```



- Consider test cases {(x=0,y=-5), (x=5, y=5)}
- The test suite is adequate for basic condition coverage, but it does not reveal the fault at statement 6
- The test suite is not adequate for branch coverage.

⇒*Branch and condition coverage*

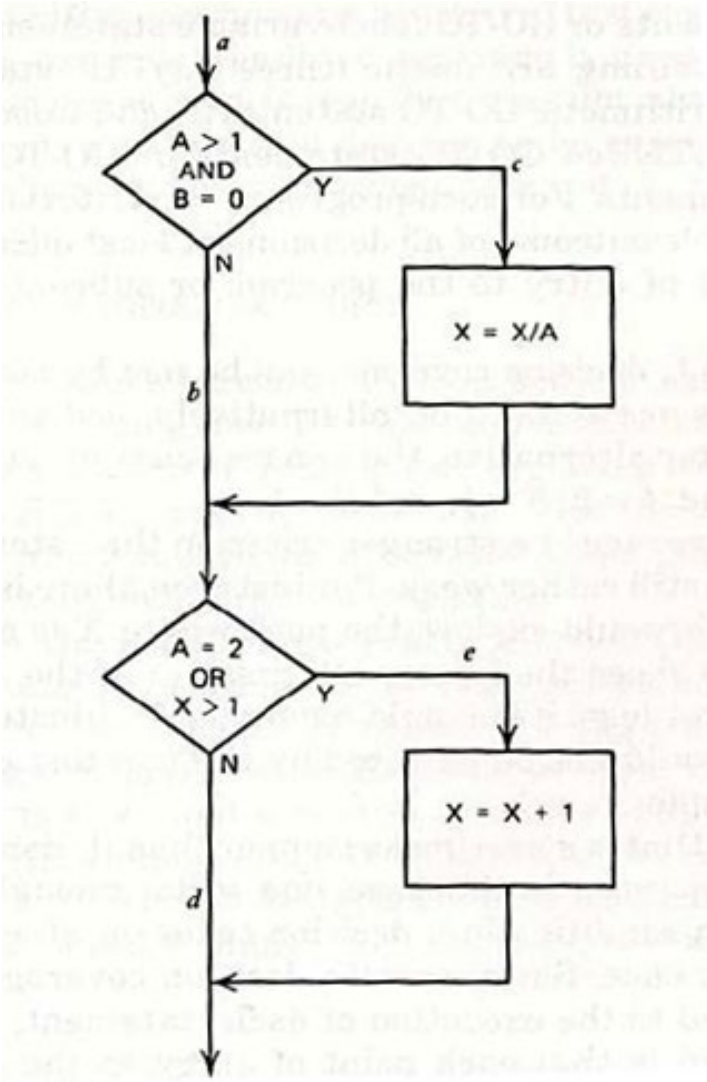
Compound or multiple condition coverage

- The test cases for all the possible combinations of outcomes of conditions in a decision (therefore the complete decision table) to be tested (exercised) at least once
  - As with condition coverage, the sub-expressions are separated by logical-and and logical-or, when present
  - The test cases required for full multiple condition coverage of a condition are given by the logical operator truth table for the condition
- Some test cases may be unnecessary
- A disadvantage: tedious to determine the minimum set of test cases required, especially for very complex Boolean expressions

Multiple condition example

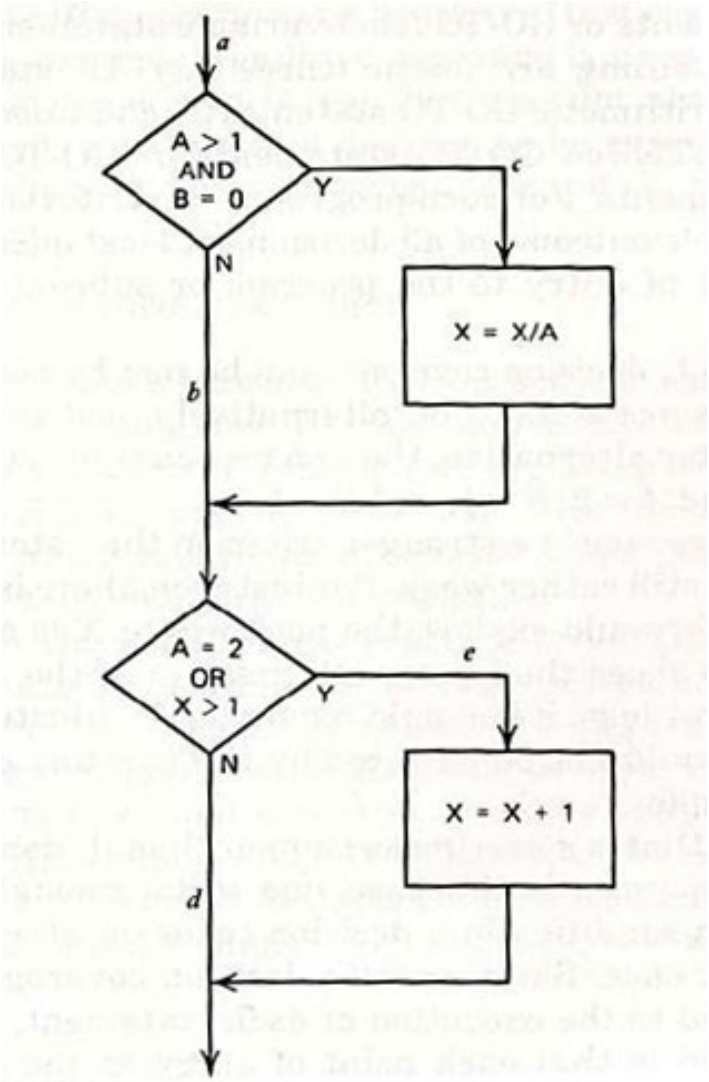
- Conditions
  - A>1, B=0
  - A>1, B<>0
  - A<=1, B=0
  - A<=1, B<>0

  - A=2, X>1
  - A=2, X<=1
  - A<>2, X>1
  - A<>2, X<=1



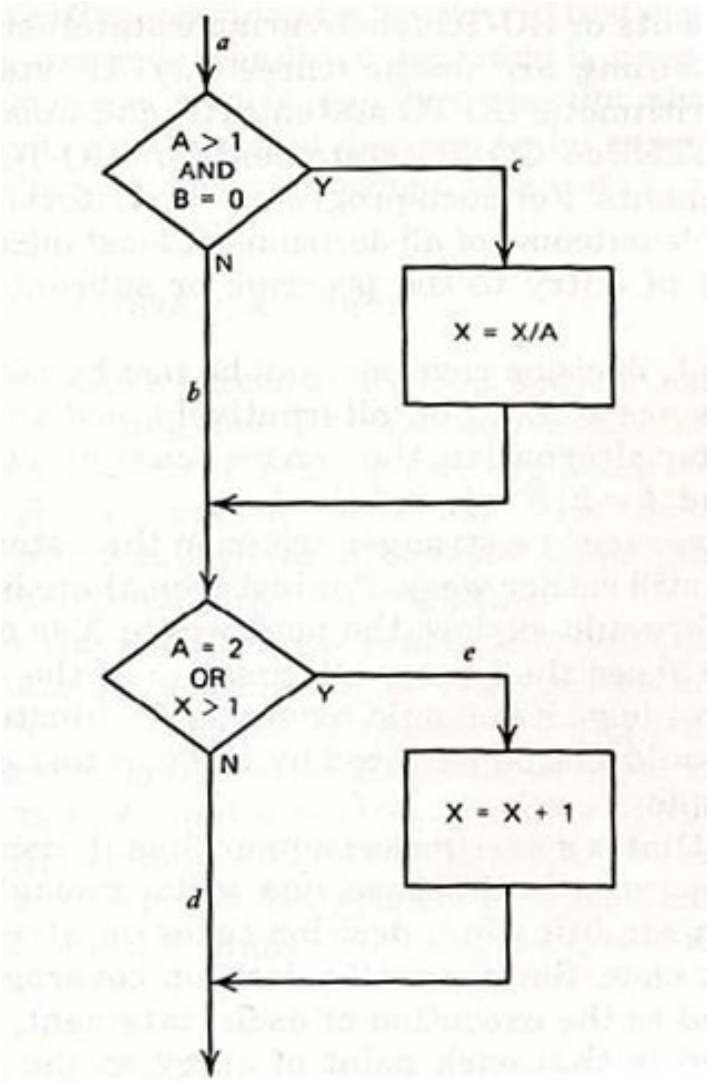
Multiple condition example

- How many test cases?



Multiple condition example

- How many test cases?
  - A=2, B=0, X=4
  - A=2, B=1, X=1
  - A=1, B=0, X=2
  - A=1, B=1, X=1



Modified condition/decision coverage

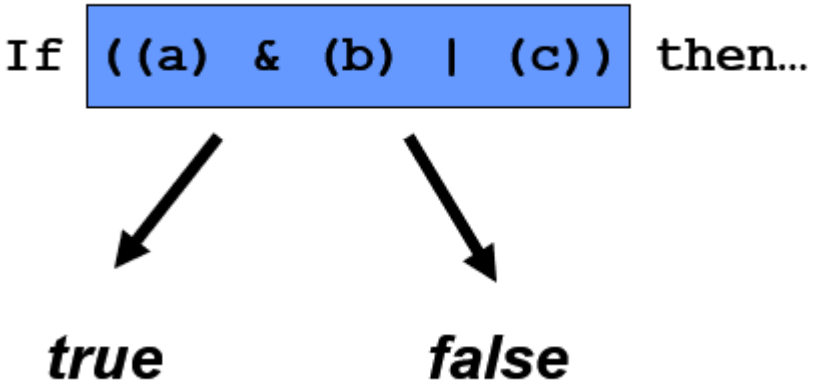
- MCDC or MC/DC
- Stands for **M**odified **C**ondition/**D**ecision **C**overage
- Created at Boeing and is required for aviation software by RCTA/DO-178C
  - DO-178C: A guideline dealing with the safety of safety-critical software used in certain airborne systems
- *A de facto standard for developing avionics software systems*
  - FAA, European Aviation, and Transport Canada

Why MC/DC Coverage

- Compound (multiple, combinatoric) condition testing generates many test cases (2^n)
  - Huge number of test cases
- Main idea of MC/DC: only consider conditions that independently impact of the outcome of a decision
  - Remember, the objective is covering the outcome

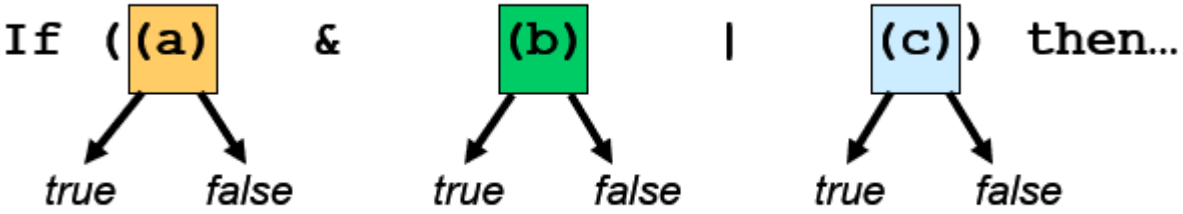
Branch coverage

- Cover both the true and false possibilities



Condition coverage

- For every condition, we consider test cases that consider the true and false value independently for each condition



- May not necessarily achieve branch coverage

Multiple condition coverage

- Compound or multiple or combinatoric condition coverage
- We consider all possible true and false combinations of all conditions
- 2^*n* test cases
- OK when *n* is 3, 4, or maybe 5
- Impractical when $n = 30$, 2^30 = 1,073,741,824 test cases
  - When $n = 50$, 2^50 = 1,125,899,900,000,000 test cases
- But does $n \geq 50$ happen in the real world?

Does n ≥ 50 happen in real world?

- "In avionics systems, complex Boolean expressions are common. Table 2 shows the number of Boolean expressions with n conditions for all of the logic expressions taken from the airborne software of five different Line Replaceable Units (LRUs) from level A systems. The five LRUs came from five different airborne systems from two different airplane models in 1995."

NASA/TM-2001-210876

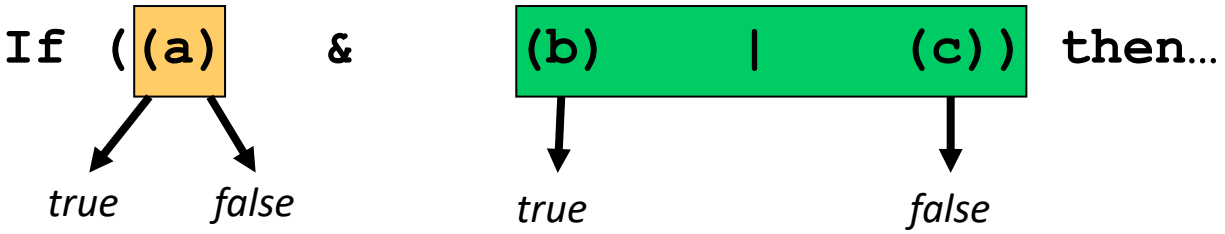
A Practical Tutorial on Modified Condition/ Decision Coverage

Table 2. Boolean Expression Profile for 5 Line Replaceable Units

	Number of Conditions, *n*									
	1	**2**	**3**	**4**	**5**	**6-10**	**11-15**	**16-20**	**21-35**	**36-76**
Number of Boolean expressions with *n* conditions	16491	2262	685	391	131	219	35	36	4	2

Requirement

- Every condition in the decision independently affects the decision's outcome.
- Not like anything we have talked about, so let's elaborate…



Change the value of each condition individually while keeping all other conditions constant.

Objective of the MC/DC Approach

- Every decision in the program has taken all possible outcomes at least once
- Every condition in a decision in the program has taken all possible *outcomes* at least once
  - Every condition in a decision has been shown to independently affect that decision's outcome
- Only a subset of compound conditions are considered
  - Not 2^n but rather n+1 test cases will be needed
  - If n=4, compound condition coverage will require 16 test cases whereas MC/DC will require 5 test cases

MC/DC

- A happy medium between decision (branch) coverage and compound (multiple) condition coverage
  - Decision coverage plus the criteria that every condition in a decision is shown to independently impact the decision outcome
  - That is the reason for "modified condition"
  - For each condition, find two rows in the decision table where the condition changes and outcome changes while other conditions remain the same

Example 1: A ∧ B ∨ C

- It is : (A && B) || C

Row	C(A)	C(B)	C(C)	Result	P(A)	P(B)	P(C)
1	T	T	T	T			
2	T	T	F	T	6	4	
3	T	F	T	T			4
4	T	F	F	F		2	3
5	F	T	T	T			
6	F	T	F	F	2		
7	F	F	T	T			
8	F	F	F	F			

Conclusions

- MC/DC subsumes branch and condition coverages
- MC/DC is weaker than multiple condition coverage
  - Compound condition coverage subsumes MC/DC
  - MC/DC subsumes condition, decision, and statement (branch) coverages
- For an expression with *n* conditions, the MC/DC criterion can be met with a minimum of *n* + 1 test cases

Summary

- Testing is one element of software quality assurance
- Verification and validation can occur in any phase
- Code-based testing: generate test cases based on inspecting the code
  - Various code coverages: statement, branch, condition, MCDC, …, path coverages

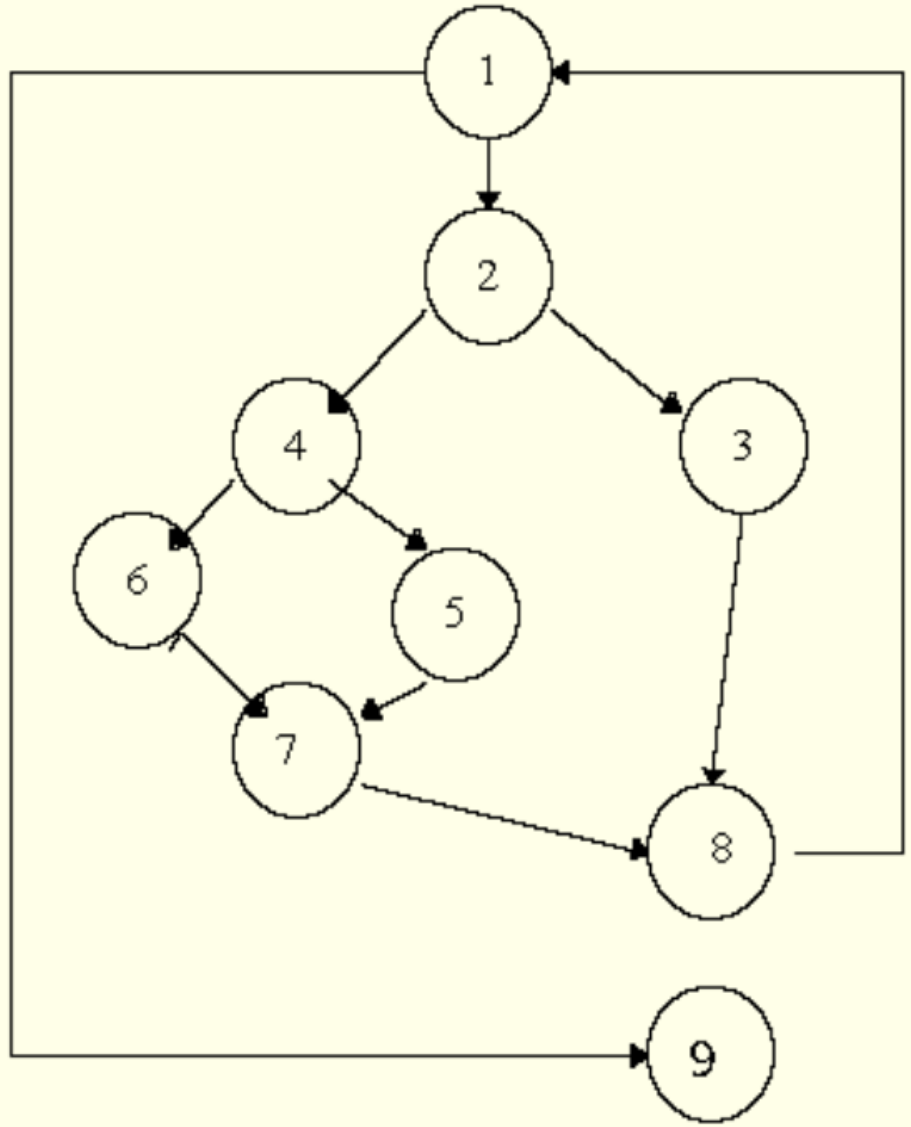
Examples

KU THE UNIVERSITY OF KANSAS

A sample pseudocode

```
1:      WHILE NOT EOF LOOP
2:         Read Record;
2:         IF field1 equals 0 THEN
3:            Add field1 to Total
3:            Increment Counter
4:         ELSE
4:             IF field2 equals 0 THEN
5:                 Print Total, Counter
5:                 Reset Counter
6:             ELSE
6:                 Subtract field2 from Total
7:             END IF
8:         END IF
8:         Print "End Record"
9:      END LOOP
9:      Print Counter
```

A sample pseudocode



Paths:

1-9
1-2-3-8-1-…
1-2-4-5-8-1-…
1-2-4-6-8-1-…

Another pseudocode

```
Procedure Validate_Pin (Valid_Pin, Return_Code)
Valid_Pin = FALSE
Return_Code = GOOD
Pin_Count = 0
do until Valid_Pin = TRUE or Pin_Count > 2 or
                        Return_Code = CANCEL
begin
      get Pin_Number (Pin_Number, Return_Code)
      if (Return_Code ≠ CANCEL)
          begin
          call Validate Pin_Number (Pin_Number, Valid_Pin)
          if (Valid_Pin = FALSE) then
                begin
                output "Invalid PIN, please re-enter PIN"
                Pin_Count = Pin_Count + 1
                end
          end
end
return (Valid_Pin, Return_Code)
```

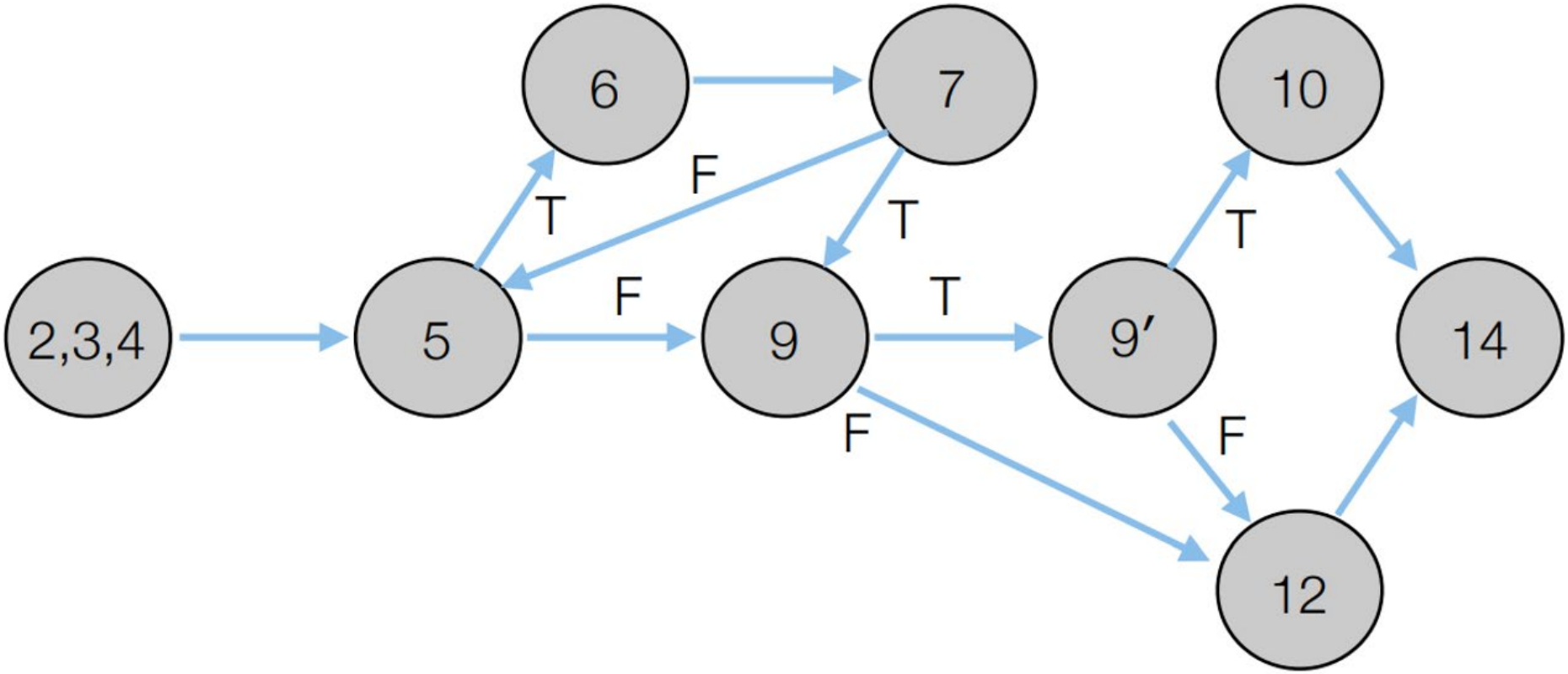
A sample Ada program

```
 1        function P return INTEGER is
 2        begin
 3            X, Y: INTEGER;
 4            READ(X); READ(Y);
 5            while (X > 10) loop
 6                X := X – 10;
 7                exit when X = 10;
 8            end loop;
 9            if (Y < 20 and then X mod 2 = 0) then
10                Y := Y + 20;
11            else
12                Y := Y – 20;
13            end if;
14            return 2 * X + Y;
15        end P;
```

A sample Ada program

```
1       function P return INTEGER is
2       begin
3          X, Y: INTEGER;
4          READ(X); READ(Y);
5          while (X > 10) loop
6             X := X – 10;
7             exit when X = 10;
8          end loop;
9          if (Y < 20 and then X mod 2 = 0) then
10            Y := Y + 20;
11         else
12            Y := Y – 20;
13         end if;
14         return 2 * X + Y;
15      end P;
```

A sample Ada program



An example: compute the average

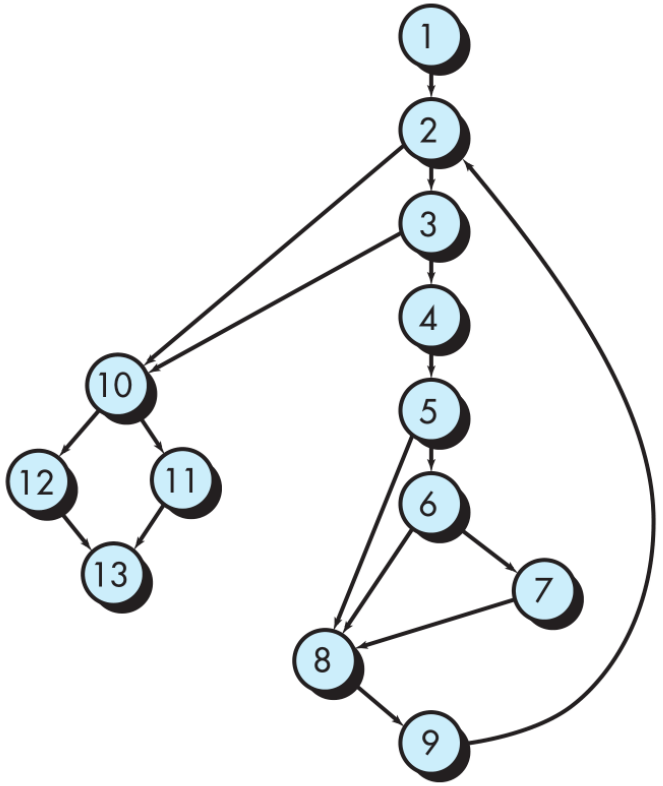
- Input
  - An array of up to 100 integers; it is called **value []**
  - An integer called **minimum**
  - An integer called **maximum**
  - **value []** has a sentinel -999 that shows the last number in the list
- Objective: Compute the average of the numbers in **value [ ]**
  - Only consider **value [i]** that are between **minimum** and **maximum**

An example

```
Procedure average;
  value : array [1..100] of integers;
  average, input, valid : integer;
  minimum, maximum, sum, i : integer;
  i = 1;
  input = valid = 0;
  sum = 0;
  while value[i] <> -999 and input < 100
    input = input + 1;
    if value[i] >= minimum and value[i] <= maximum
      then valid = valid + 1;
           sum = sum + value[i];
      else skip
    endif
    i = i + 1;
  endwhile
  if valid > 0
    then average = sum / valid;
    else average = -999;
  endif
```

An example

```
PROCEDURE average;

    *  This procedure computes the average of 100 or fewer
       numbers that lie between bounding values; it also computes the
       sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
         minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;                          2
    sum = 0;
    DO WHILE value[i] <> –999 AND total.input < 100         3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum      6
5       THEN increment total.valid by 1;
7                 sum = s sum + value[i]
        ELSE skip
      ENDIF
8     increment i by 1;
9   ENDDO
    IF total.valid > 0                                      10
11    THEN average = sum / total.valid;
12  ELSE average = –999;
13  ENDIF
END average
```



Path 1: 1-2-10-11-13
Path 2: 1-2-10-12-13
Path 3: 1-2-3-10-11-13
Path 4: 1-2-3-4-5-8-9-2-. . .
Path 5: 1-2-3-4-5-6-8-9-2-. . .
Path 6: 1-2-3-4-5-6-7-8-9-2-. . .

Test cases

Path 1 test case

attempt to process 101 or more values

first 100 values should be valid

expected result: correct average based on 100 values and proper totals

Path 2 test case

value (i) = valid input where i < 100

value (k) < minimum where k < i

expected results: correct average based on k values and proper totals

Path 3 test case

value (i) = valid input where i < 100

value (k) > maximum where k <= i

expected results: correct average based on k values and proper totals

Test cases

Path 4 test case:

value(i) = valid input where i < 100

expected result: correct average based on 100 values and proper totals

Path 5 test case:

value(k) = valid input where k < i

value(i) = -999 where 2 <= i<= 100

expected results: correct average based on k values and proper totals

Note: cannot be tested alone; must be with path 2, 3, 4 tests

Path 6 test case:

value(1) = -999

expected results: average = -999; other totals at initial values

Example 2: A && B && C

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

- Start with **A**: Find two rows where the value of **A** and outcome change but **B** and **C** stay the same
- Rows 1 and 5
- Anymore?

Example 2: A && B && C

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

- Start with **B**: Find two rows where the value of **B** and outcome change but **A** and **C** stay the same
- Rows 1 and 3
- Anymore?

Example 2: A && B && C

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

- Start with **C**: Find two rows where the value of **C** and outcome change but **A** and **B** stay the same
- Rows 1 and 2
- Anymore?

Example 2: A && B && C

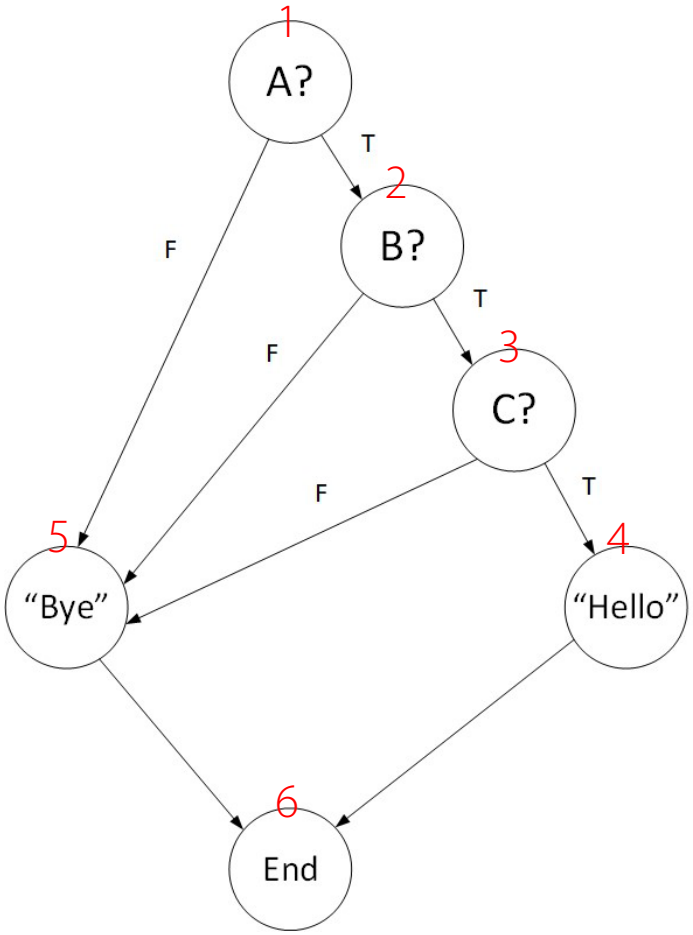
Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

- We must consider developing test cases for the following rows: {1,5} (because of **A**), {1,3} (b/c of **B**), {1,2} (b/c of **C**)
- We need test cases for rows: 1,2,3 and 5
- We went from 2^3 (8 test cases) to 3+1 (4 test cases)

Example 2: an observation

if (A && B) && C print "Hello" else print "Bye";

- Convert to flowgraph
- How many paths?
- No loops; paths are for edge coverage
- What are the paths?
  - 1-2-3-4-6 (corresponds to row 1)
  - 1-2-3-5-6 (corresponds to row 2)
  - 1-2-5-6 (corresponds to row 3)
  - 1-5-6 (corresponds to row 5)



Example 3: $A \wedge (B \vee C)$

- It is : $A \;\&\&\; (B \;||\; C)$

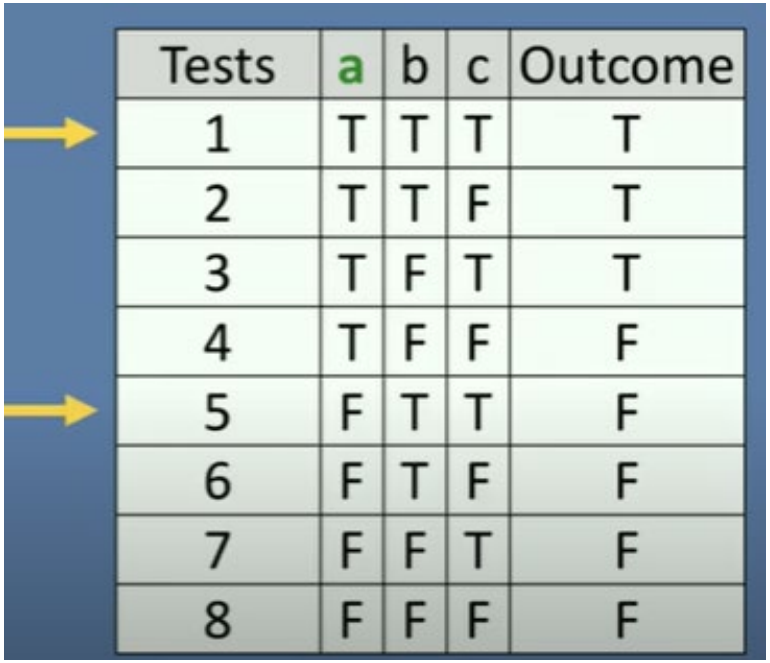
Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Consider A: find two rows where value of A changes, B and C stay the same, but the outcome changes



Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

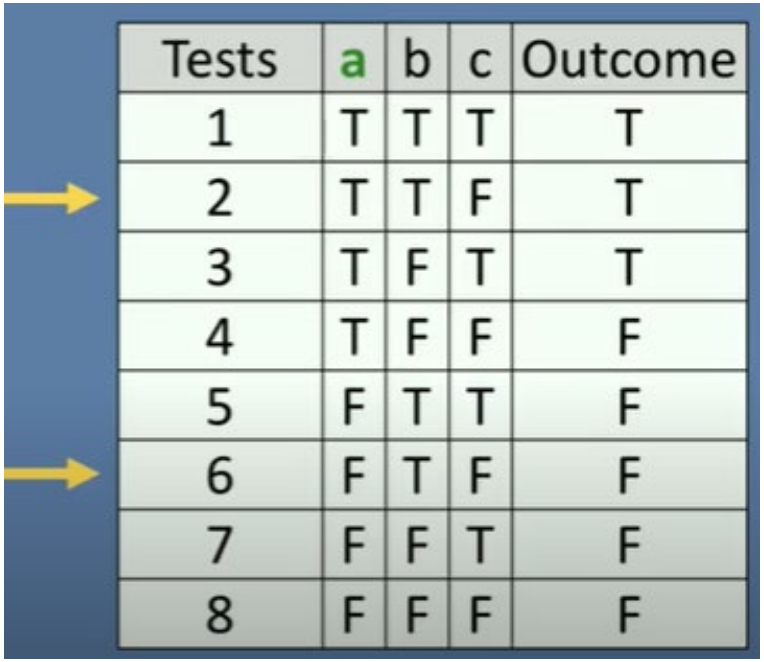
- Rows 1 and 5: keep for testing

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Consider A: find two rows where value of A changes, B and C stay the same, but the outcome changes



Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

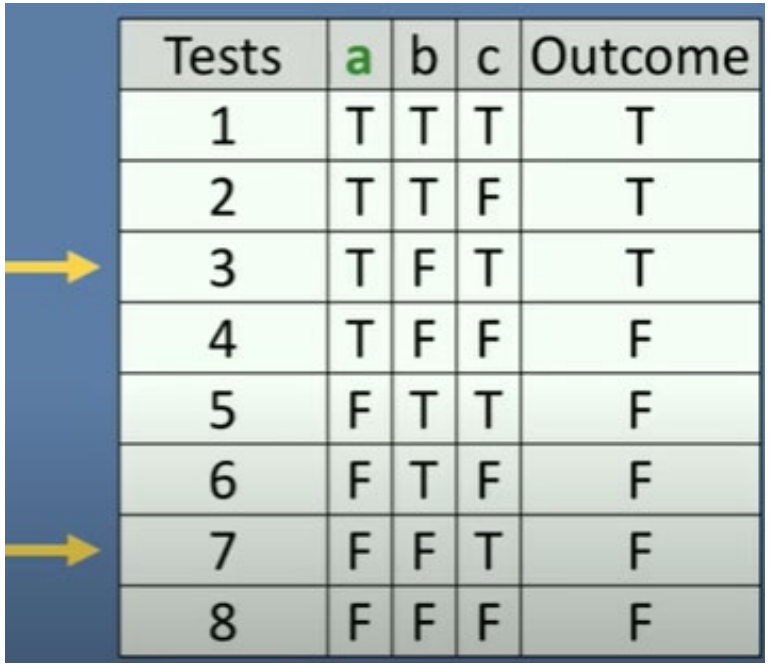
- Rows 2 and 6: keep for testing

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Consider A: find two rows where value of A changes, B and C stay the same, but the outcome changes



Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Rows 3 and 7: keep for testing

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

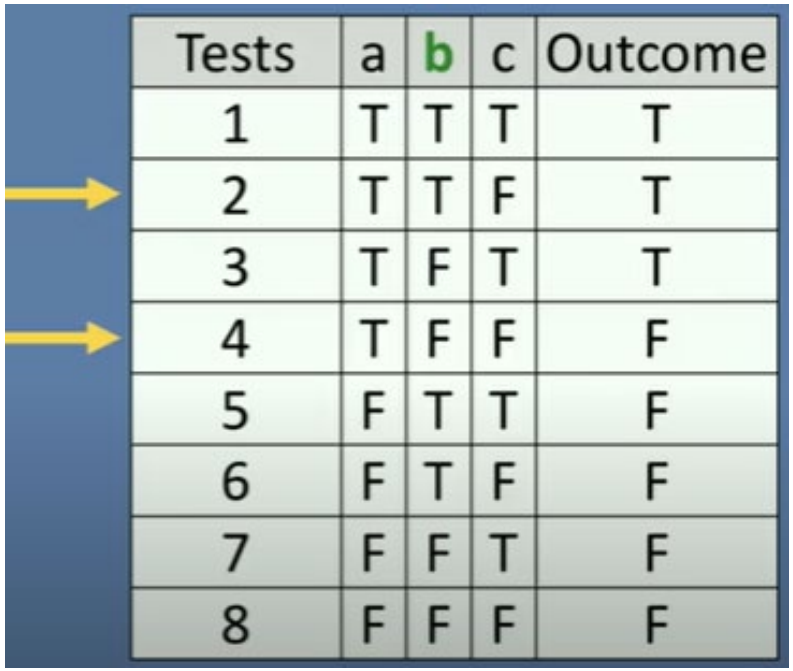
- Consider A: find two rows where value of A changes, B and C stay the same, but the outcome changes
- Continue with the above process
- Anymore? 4 and 8?
- For A we must develop test cases for: {1,5}, {2,6}, {3,7}

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Consider B: find two rows where value of B changes, A and C stay the same, but the outcome changes



Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Rows 2 and 4: keep for testing

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

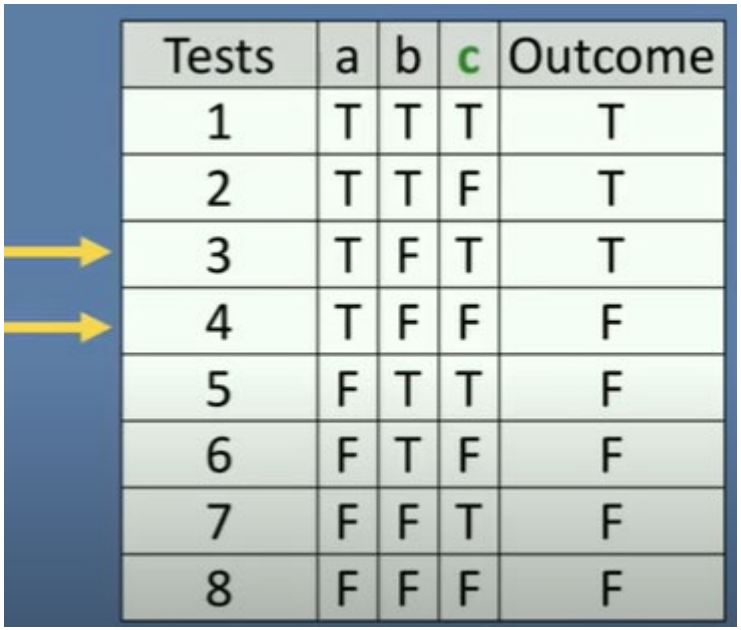
- Consider B: find two rows where value of B changes, A and C stay the same, but the outcome changes
- Continue with the above process
- Anymore?
- For B we must develop test cases for: {2, 4}

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Consider C: find two rows where value of B changes, A and B stay the same, but the outcome



Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- For C we must develop test cases for: {3, 4}

Example 3: $A \wedge (B \vee C)$

- A && (B || C)

Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- Consider C: find two rows where value of B changes, A and B stay the same, but the outcome Continue with the above process
- Continue with the above process
- Anymore?
- For C we must develop test cases for: {3, 4}

Example 3: A ∧ (B ∨ C)

- A && (B || C)

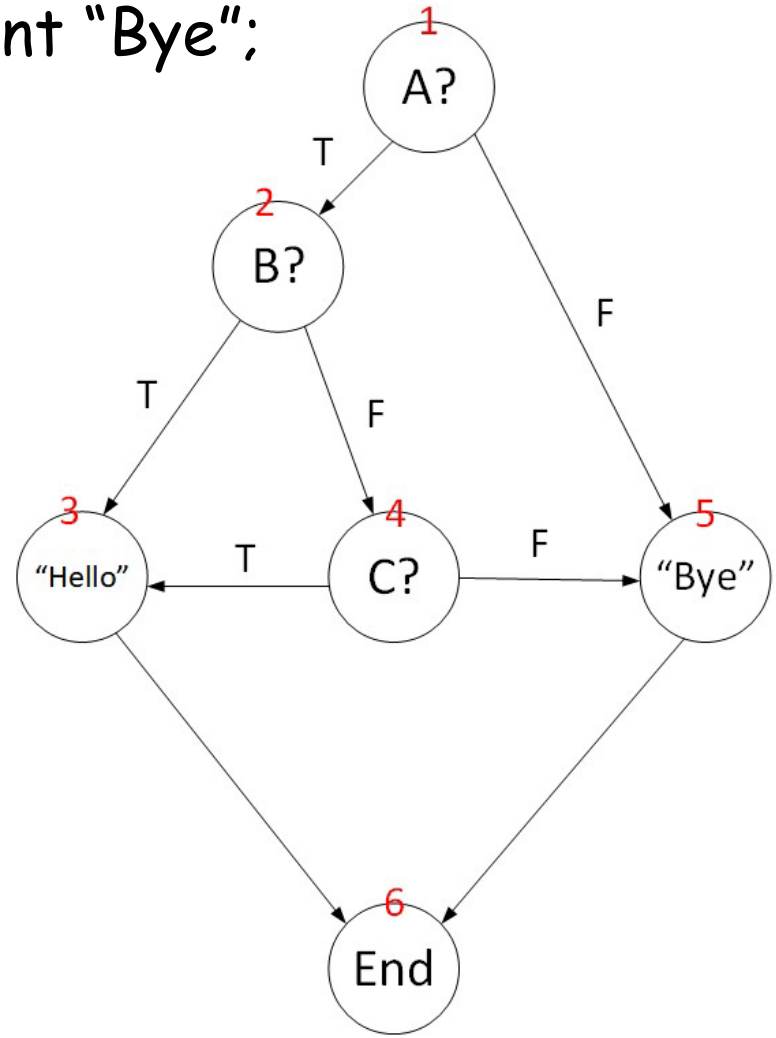
Tests	a	b	c	Outcome
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

- For A we have three options: {1,5}, {2,6}, {3,7}
- For B we have one option: {2,4}
- For C we have one option: {3,4}
- If we only consider {2,3,4,6} we will have a pair for A, a pair for B, and pair for C
- We can consider test cases for {1,2,3,5,6,7}, but no need to
- We have n+1 test cases

Example 3: an observation

if A && (B || C) print "Hello" else print "Bye";

- Convert to flowgraph
- How many paths?
- No loops; paths are for edge coverage
- What are the paths?
  - 1-2-3-6 (corresponds to row 2)
  - 1-2-4-3-6 (corresponds to row 3)
  - 1-2-4-5-6 (corresponds to row 4)
  - 1-4-5-6 (corresponds to row 6)



Example 4

- **(((a || b) && c) || d) && e**
- Five logical clauses
- Combinatoric coverage: 2^5 = 32 test cases
- Short circuiting or "I don't care" entries reduce the test cases to 13 test cases (many compilers already do it)
  - Still very good
- MC/DC reduce it to n+1 or 6 test cases

Example 4: short-circuited

```
(((a || b) && c) || d) && e
```

Test Case	a	b	c	d	e
(1)	T	—	T	—	T
(2)	F	T	T	—	T
(3)	T	—	F	T	T
(4)	F	T	F	T	T
(5)	F	F	—	T	T
(6)	T	—	T	—	F
(7)	F	T	T	—	F
(8)	T	—	F	T	F
(9)	F	T	F	T	F
(10)	F	F	—	T	F
(11)	T	—	F	F	—
(12)	F	T	F	F	—
(13)	F	F	—	F	—

-- Short-circuit evaluation often reduces this to a more manageable number

-- The above shows test cases for the compound condition coverage

Example 4: test cases

```
(((a || b) && c) || d) && e
```

Test case	a	b	c	d	e	outcome
(1)	true	--	true	--	true	true
(2)	false	true	true	--	true	true
(3)	true	--	false	true	true	true
(6)	true	--	true	--	false	false
(11)	true	--	false	false	--	false
(13)	false	false	--	false	--	false

- Underlined values independently affect the output of the decision
- Required by the RTCA/DO-178B standard