White-Box Testing

Spring, 2023

Yi Xiang xiangyi@scut.edu.cn

Contents

- Control Flow Testing
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Decision Condition Coverage
  - Condition Combination Coverage

  - Path Coverage
  - Basis Path Testing

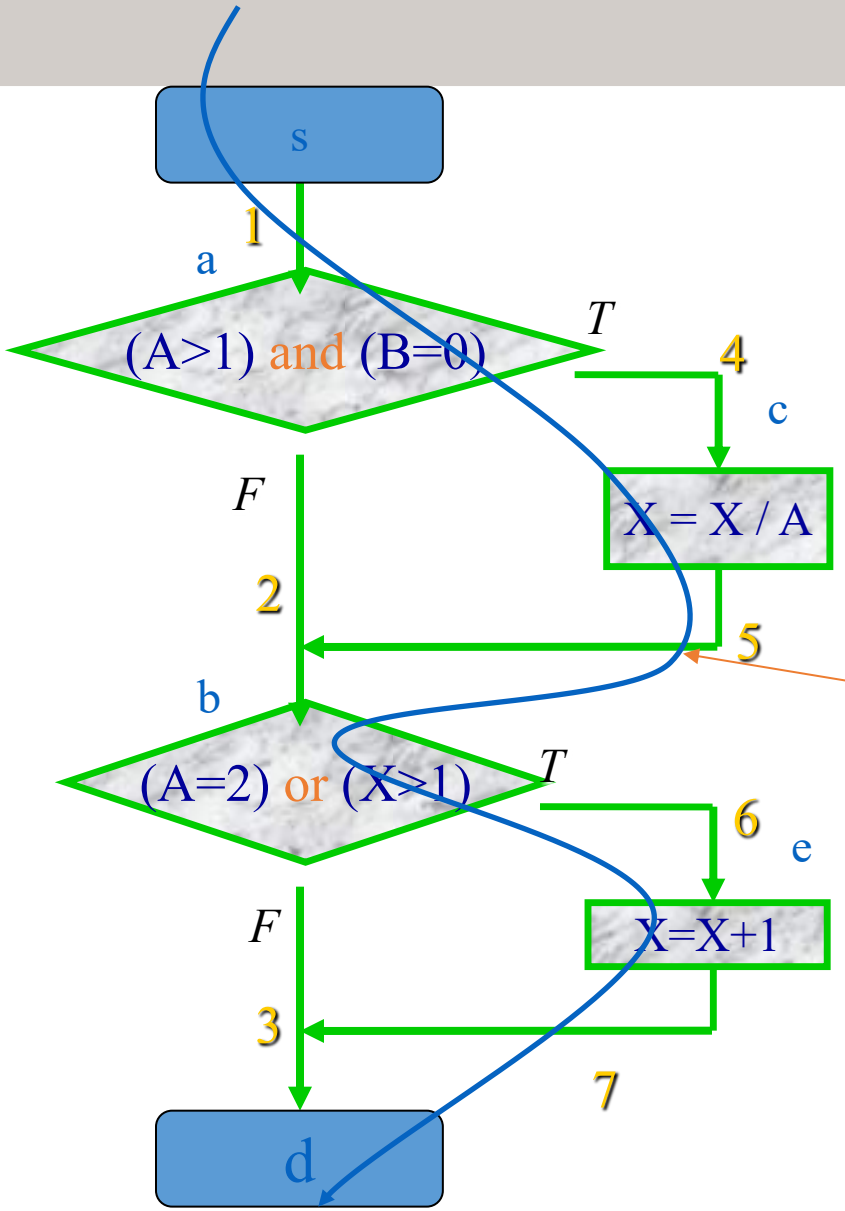
Control Flow Testing

- Control-flow testing is a structural testing strategy that uses the program's control flow as a model.
- Requires the tester to have a clear understanding of the logical structure of the program, and even to be able to master all the details of the source program.
- Most applicable to new software for unit testing.

1. Statement Coverage

- Design test cases and work out input values required to ensure that every source code statement is executed.
- Also known as point coverage
- The weakest logical coverage, be used interoperatively with other testing methods.

Program Flow Graph



Source Code

```
public static float Example(float A,B,X){

    if ( A>1 &&  B==0 )
        X= X / A;
    if  (A==2 || X>1)
        X=X+1;
    return X;

}
```

I. A=2, B= 0, X=4 ---- sacbed

TestCase	Input (A，B，X)	Path	Output (X)
1	2,0,4	**sacbed**	3

Statement Coverage

The statement coverage seem like it validates every statement comprehensively, but it's quite weak. Why?

If there is a problem with the logical operation of the two decisions, like:

- AND → OR
- OR → AND
- X>1 → X>0

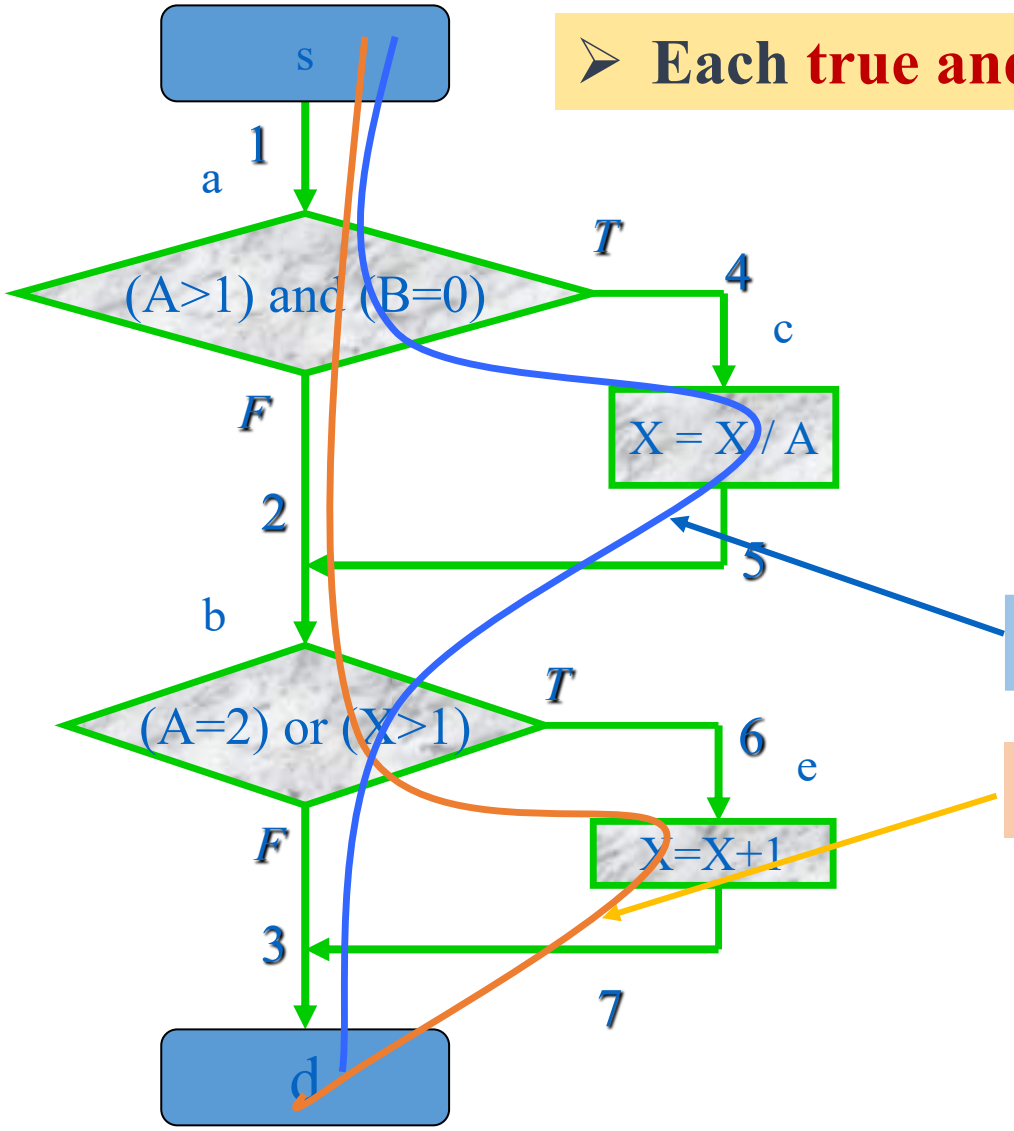
.....

the above test case cannot detect it.

2. Decision Coverage (Branch Coverage)

- Design test cases and work out input values required to ensure that every source code branch is taken.
  - Each true and false branch of the program is executed at least once

- Also known as edge coverage



➢ **Each true and false branch is executed at least once**

Is there other test cases combination satisfying branch coverage?

Does it cover all the statement?

I: A=3, B=0,X=1: sacbd

II: A=2, B=1,X=1: sabed

Decision Coverage (Branch Coverage)

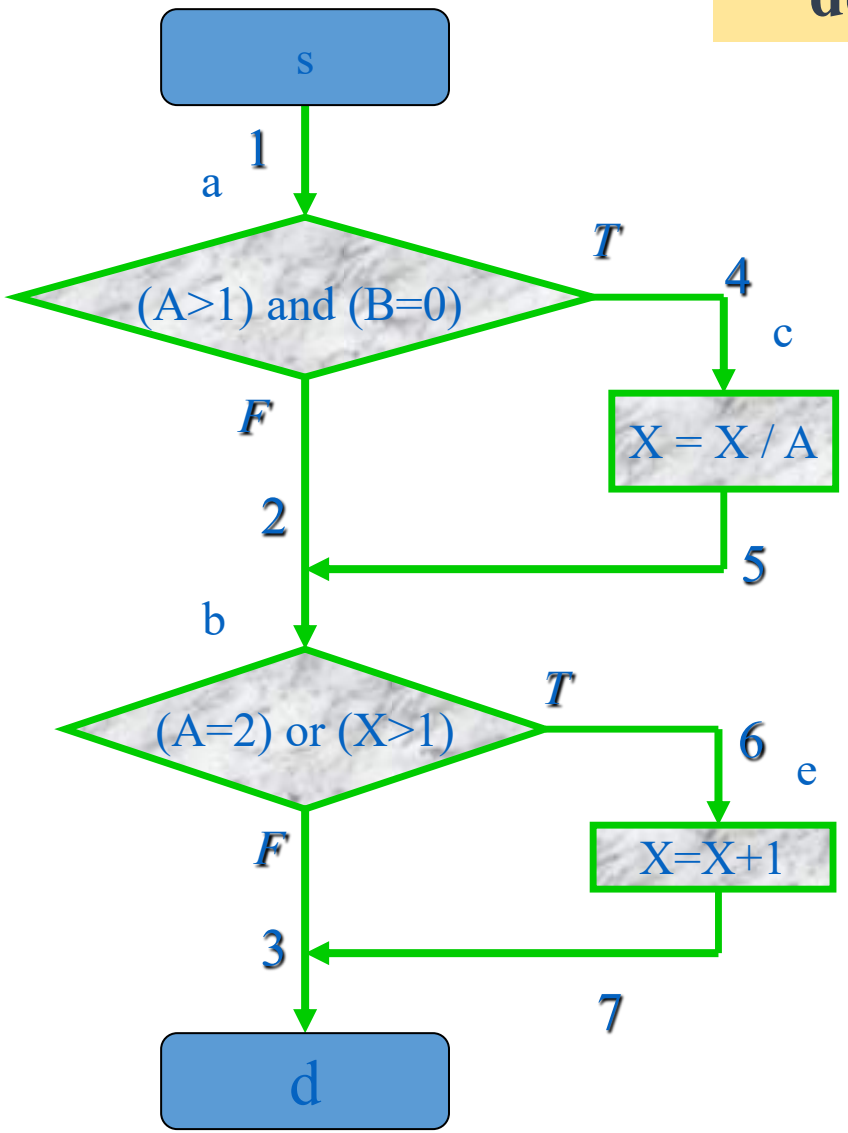
- Test cases that satisfy decision coverage definitely satisfy statement coverage.
- Decision Coverage is better than statement coverage, but it's still weak logical coverage.
  - X>1 → X<1

  the above test cases cannot detect it.
- Decision coverage does not guarantee that errors in decision conditions can be detected. Therefore, stronger logical coverage criteria are needed to test the internal conditions.

3. Condition Coverage

- A complex decision is formed from multiple (Boolean) conditions.
- Condition Coverage extends Branch Coverage by ensuring that, for complex decisions, each condition within the decision is tested for its true and false values.
- There is a caveat: it is not necessary that the decision itself take on true and false values!
- Test data is selected to ensure that every condition in every decision takes on the value true and false.

- **Each true and false condition in each decision is executed at least once**

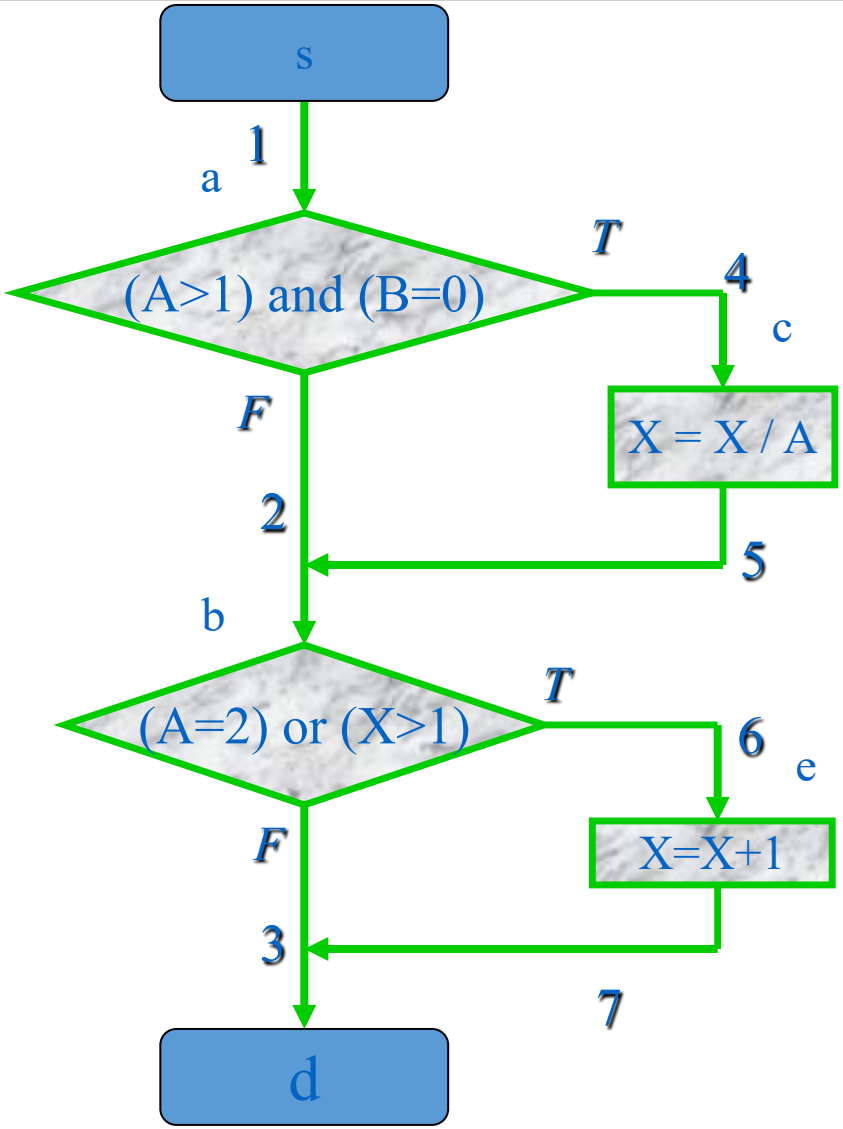


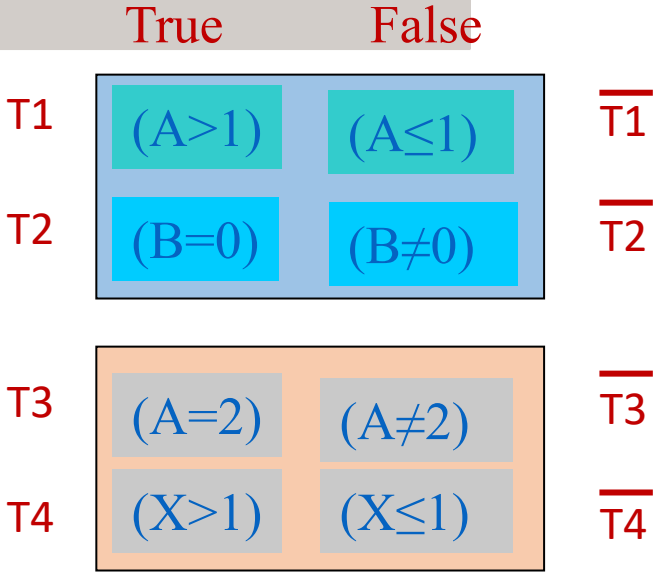
Decision Expression #1

If A>1	True	T1
	False	$\overline{T1}$
If B==0	True	T2
	False	$\overline{T2}$

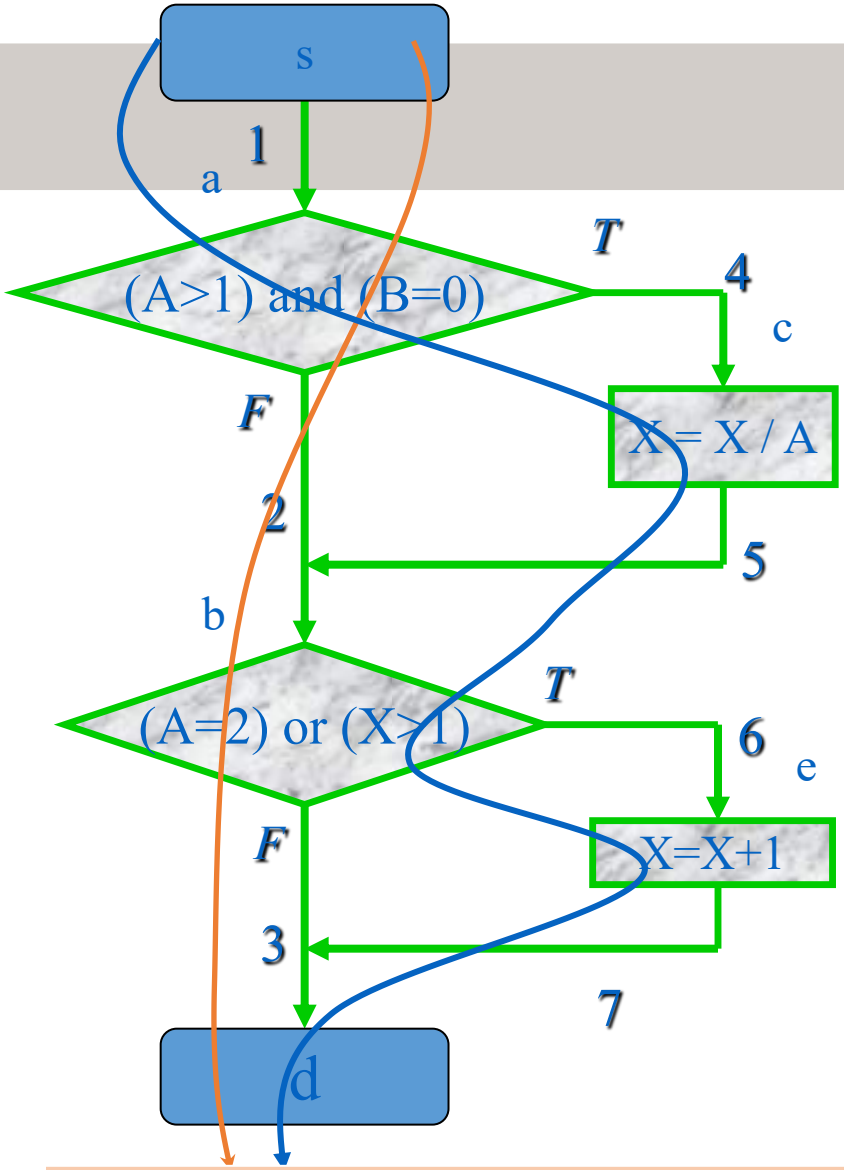
Decision Expression #2

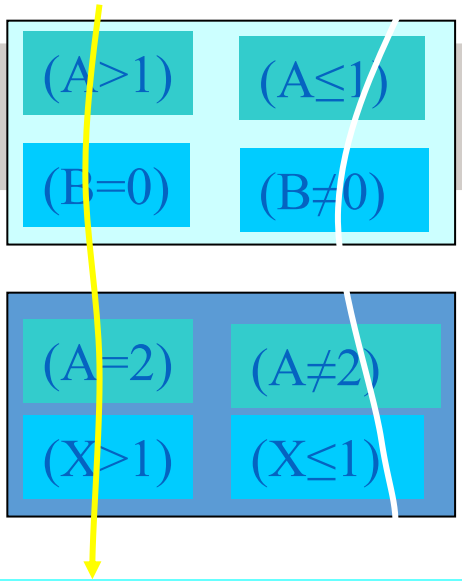
If A==2	True	T3
	False	$\overline{T3}$
If X>1	True	T4
	False	$\overline{T4}$





TestCase			Path	Condition	Output
A	B	X			X





Situation #1

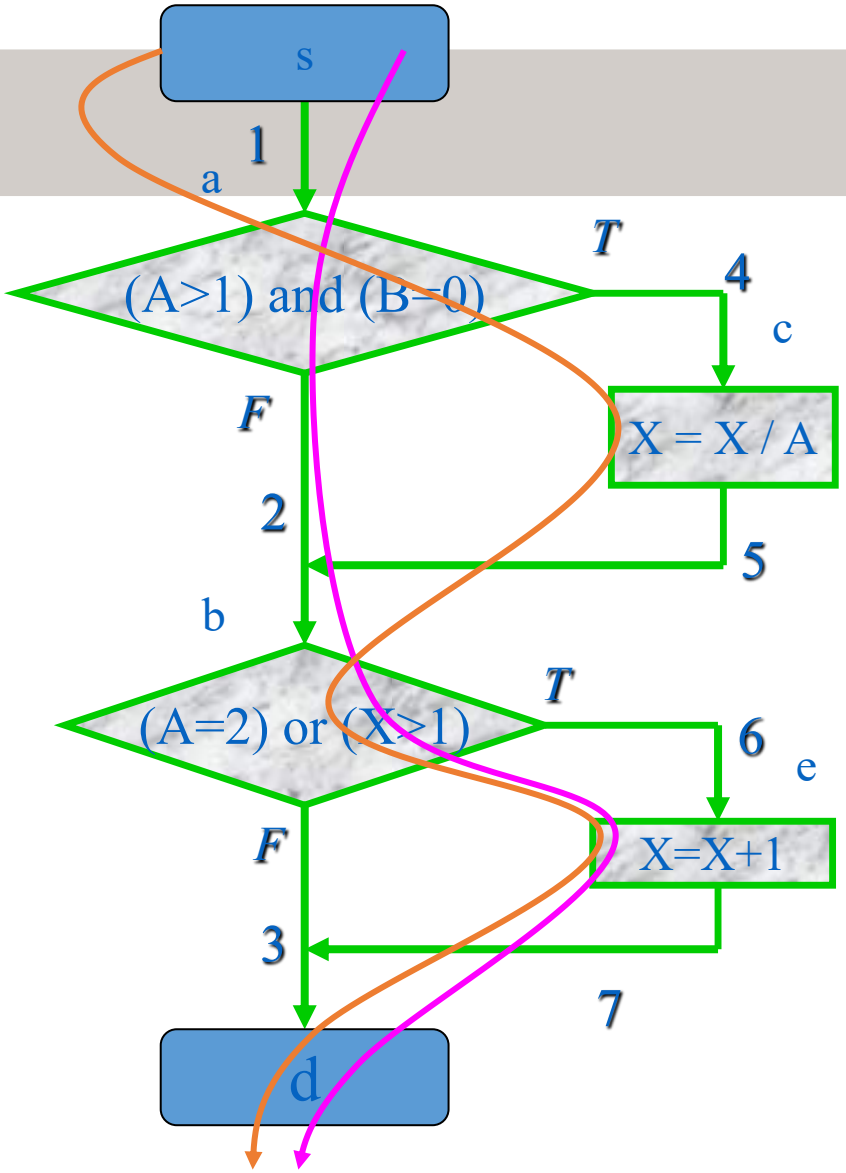
Ⅰ: A=2, B=0,X=4: sacbed

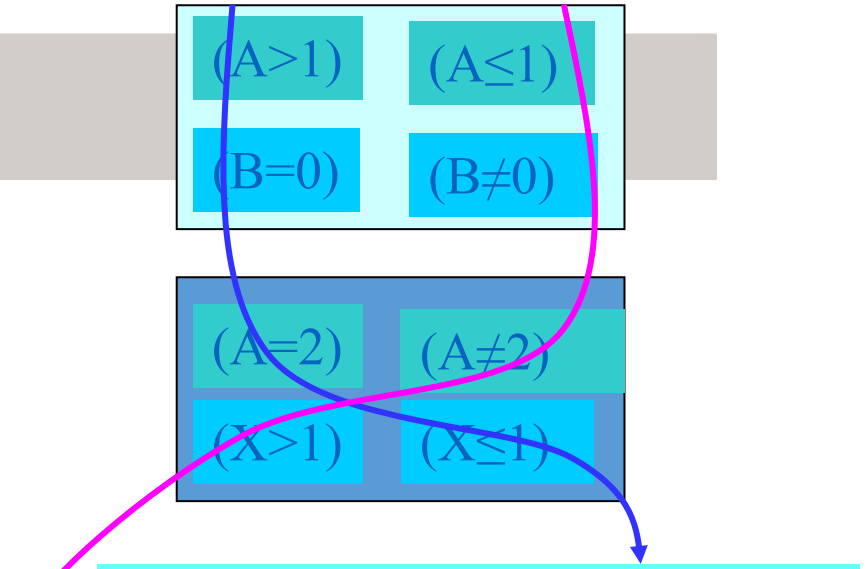
Ⅱ: A=1, B=1,X=1: sabd

TestCase			Path	Condition	X
A	B	X			
2	0	4	sacbed	T1,T2,T3,T4	3
1	1	1	sabd	$\overline{T1},\overline{T2},\overline{T3},\overline{T4}$	1

Whether decision coverage is satisfied

Decision coverage is satisfied



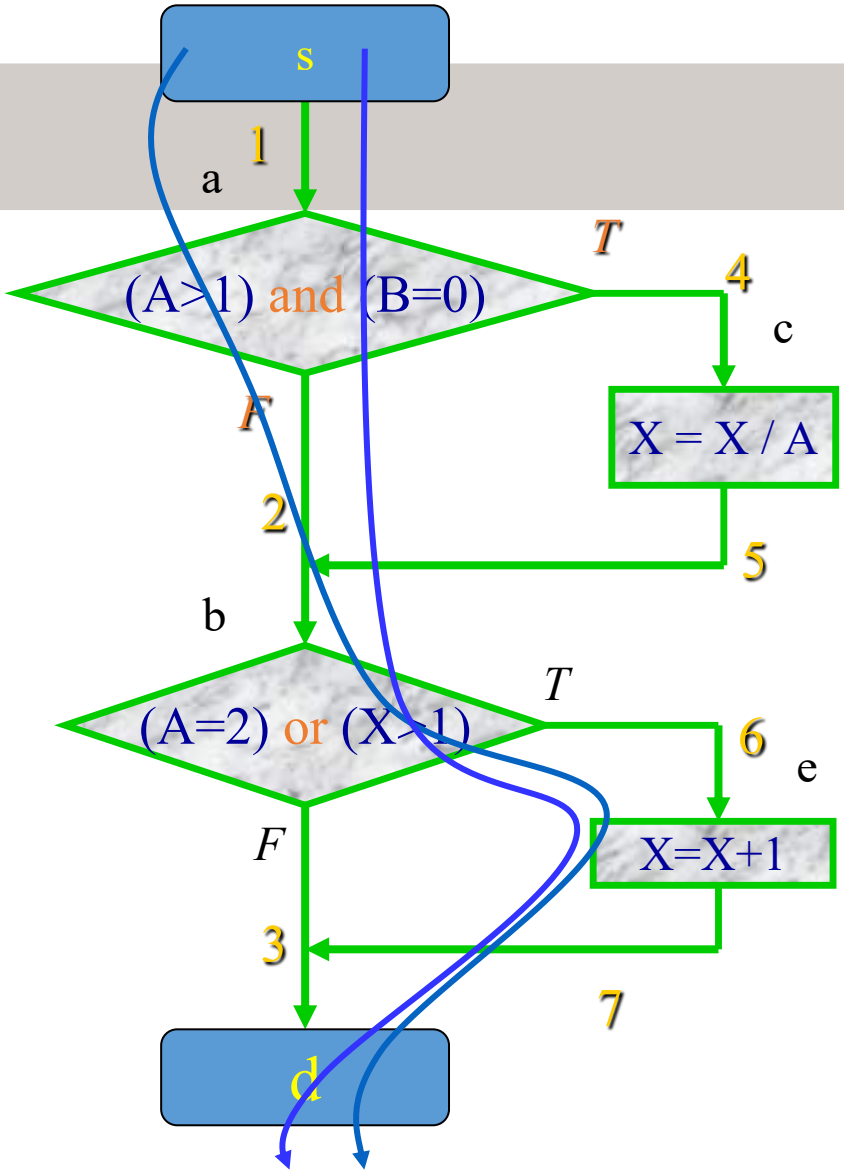


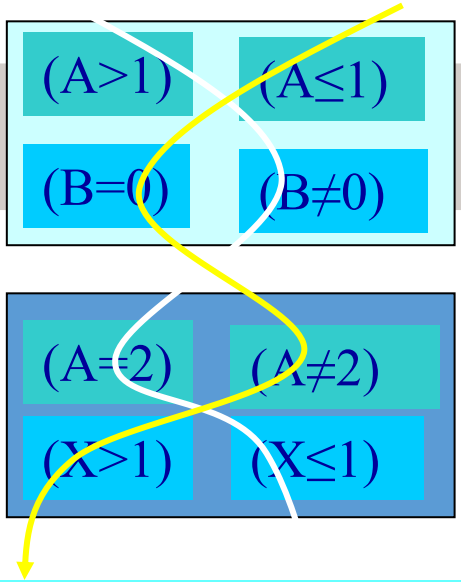
III: A=2, B=0,X=1: sacbed

IV: A=1, B=1,X=2: sabed

Decision coverage isn't satisfied

TestCase			Path	Condition	X
A	B	X			
2	0	1	sacbed	$T1,T2,T3,\overline{T4}$	1.5
1	1	2	sabed	$\overline{T1},\overline{T2},\overline{T3},T4$	3





Situation #3

V : A=1, B=0,X=3: sabed

VI: A=2, B=1,X=1: sabed

TestCase			Path	Conditions	X
A	B	X			
1	0	3	sabed	$\overline{T1}$,T2,$\overline{T3}$,T4	4
2	1	1	sabed	T1,$\overline{T2}$,T3,$\overline{T4}$	2

Neither decision coverage
nor statement coverage is satisfied

Condition Coverage

- Strength: focuses on condition outcomes and thus extends Branch coverage

- Weakness: may fail to achieve branch coverage as it is not necessary for the decision itself to take on true and false outcomes.

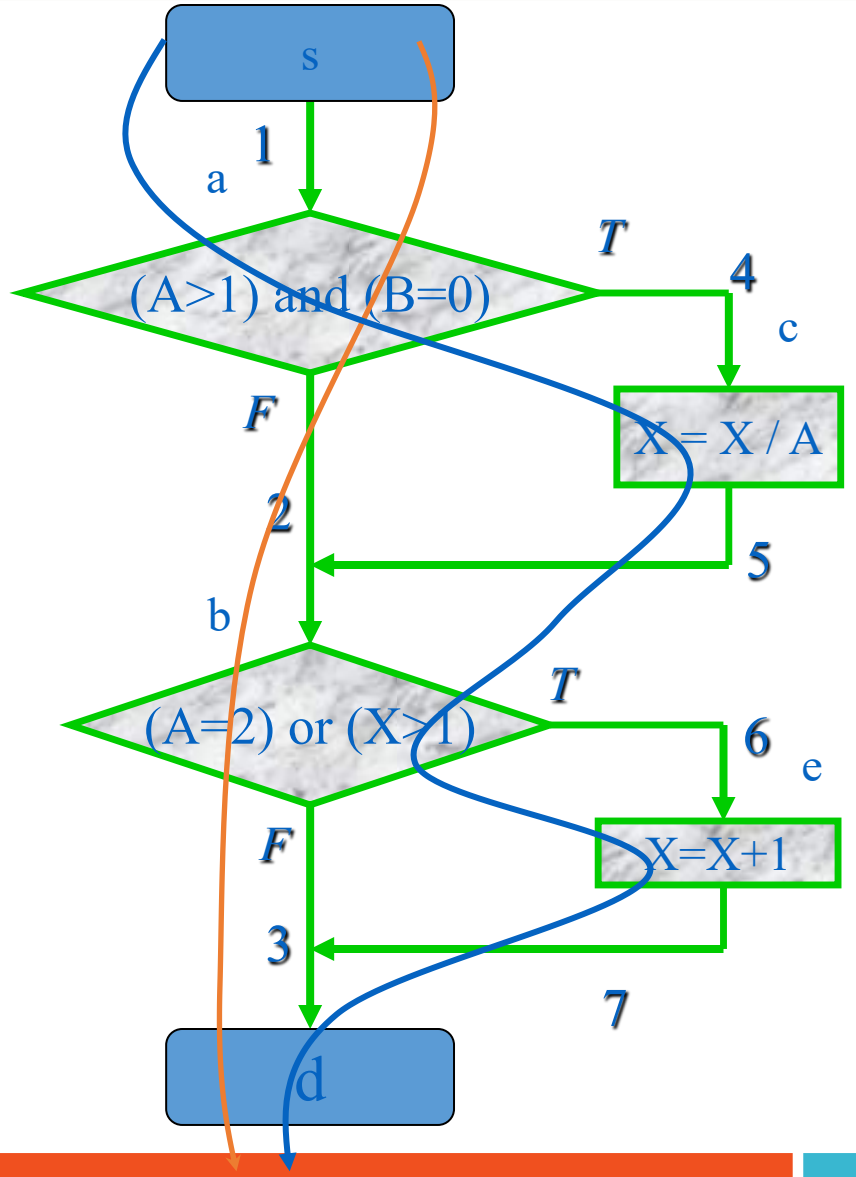
consider the test cases
(1) a=true and b= false
(2) a=false and b=true

Each condition (a and b) have taken on the values of true and false but the decision itself always evaluates to false.

Thus, Branch Coverage has not been achieved

4. Decision/Condition Coverage

- Generate test data such that all conditions in a decision take on both outcomes (if possible) at least once and exercise the true and false outcomes of every decision.
  - Each decision has True and False test cases
  - In addition, each condition in a decision has True and False test cases (if possible)
- It is a combination of Condition Coverage and Branch Testing. It uses the same test data as for Condition Coverage but must additionally ensure that each branch or decision takes a true or false outcome.
  - Single condition decision: 2 test cases
  - 2-condition decisions: 2+ test cases



TestCase			Path	Condition	Decision #1,#2	X
A	B	X				
2	0	4	sacbed	T1,T2,T3,T4	T, T	3
1	1	1	sabd	$\overline{T1}$,$\overline{T2}$, $\overline{T3}$, $\overline{T4}$	F,F	1

Airline Seat reservation Example

Program Code:

```
(1) public static Boolean seatsAvailable(
       int freeSeats, int seatsRequired)
(2) {
(3)    boolean rv=false;
(4)    if ( (freeSets>=0) && (seatsRequired>=1) &&
            (seatsRequired<=freeSeats) )
(5)       rv=true
(6)    return rv;
(7) }
```

In the program there is one decision with three conditions on line 4.

```
if ( (freeSeats>=0) && (seatsRequired>=1)
      && (seatsRequired<=freeSeats) )
```

Airline Seat reservation Example

Each Boolean value for each decision

```
1)((freeSeats>=0) && (seatsRequired>=1)
  && (seatsRequired<=freeSeats) )
2)!((freeSeats>=0) && (seatsRequired>=1)
  && (seatsRequired<=freeSeats) )
```

In addition, each Boolean value for each condition

```
3)(freeSeats>=0)
4)!(freeSeats>=0)
5)(seatsRequired>=1)
6)!(seatsRequired>=1)
7)(seatsRequired<=freeSeats)
8)!(seatsRequired<=freeSeats)
```

Airline Seat reservation ----Test Cases

Test No.	Test Cases/Decisions & Conditions Covered	Inputs		Expected Outputs
		freeSeats	*seatsRequired*	*Return Value*
1	1, 3, 5, 7	50	25	True
2	2, 4, 6, 8	-50	-25	False

Decision/Condition Coverage Comments

- Steps
  - Identify all the decisions in the program
  - List all the conditions
  - Generate the test data to cover the above decisions and conditions

- Addresses one of deficiencies of condition coverage by forcing each branch to be exercised .
  - Conditions can be masked due to the potential lazy evaluation of compound conditions, e.g.

```
while(!(found) && (i<=x))
```

  - For the decision to be true, both conditions must evaluate to true.
  - For the decision to be false, only the first condition need to evaluate to false.

- Thus, the consequence of the second condition evaluating to false might be insufficiently considered.

Decision/Condition Coverage Strengths & Weaknesses

- The true and false outcomes of every decision and every condition are covered
- This gives stronger coverage than just Condition Coverage or Decision Coverage

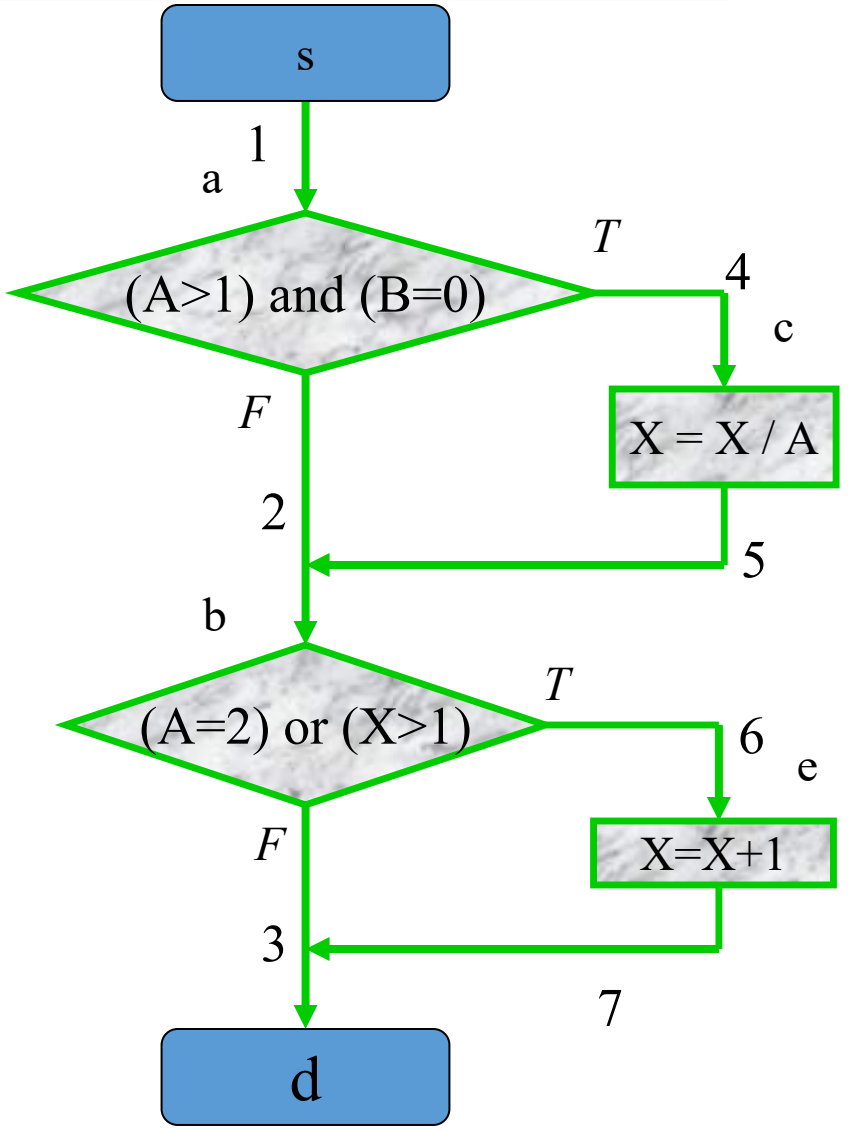
- Even though every decision is tested, and every condition is tested, not every possible combination of conditions is tested.

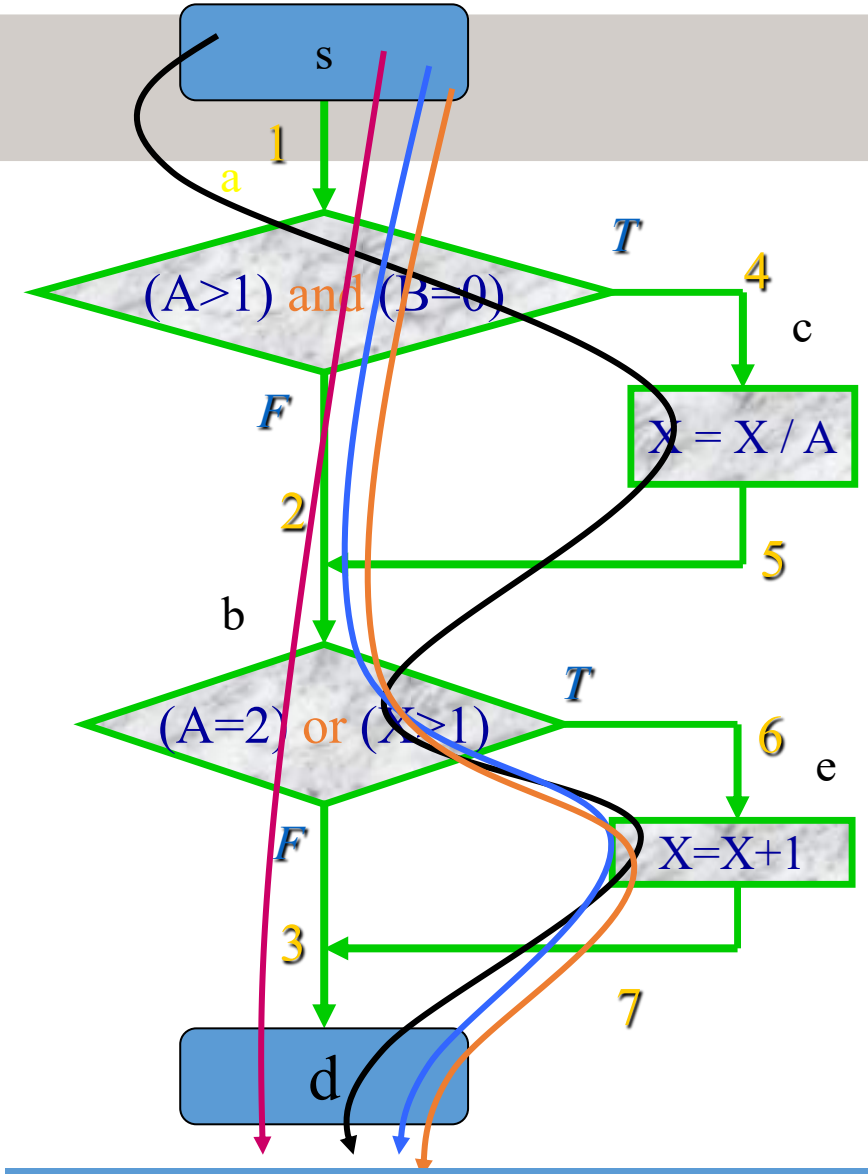
5. Condition Combination Coverage

- Tests are generated to cause every possible combination of conditions for every decision to be tested.
- The goal is to achieve 100% coverage of every decision and 100% coverage of every condition.
- A Truth-Table is the best way to identify all the possible combinations of values.

Every combination of conditions for every decision be taken at least once

①	A＞1,	B＝0,	TT
②	A＞1,	B≠0,	TF
③	A≯1,	B＝0,	FT
④	A≯1,	B≠0,	FF
⑤	A＝2,	X＞1,	TT
⑥	A＝2,	X≯1,	TF
⑦	A≠2,	X＞1,	FT
⑧	A≠2,	X≯1,	FF





1. A > 1, B = 0
2. A > 1, B≠0
3. A≯1, B = 0
4. A≯1, B≠0
5. A = 2, X > 1
6. A = 2, X≯1
7. A≠2, X > 1
8. A≠2, X≯1

I. A=2,B=0,X=4

II. A=2,B=1,X=1

III. A=1,B=0,X=2

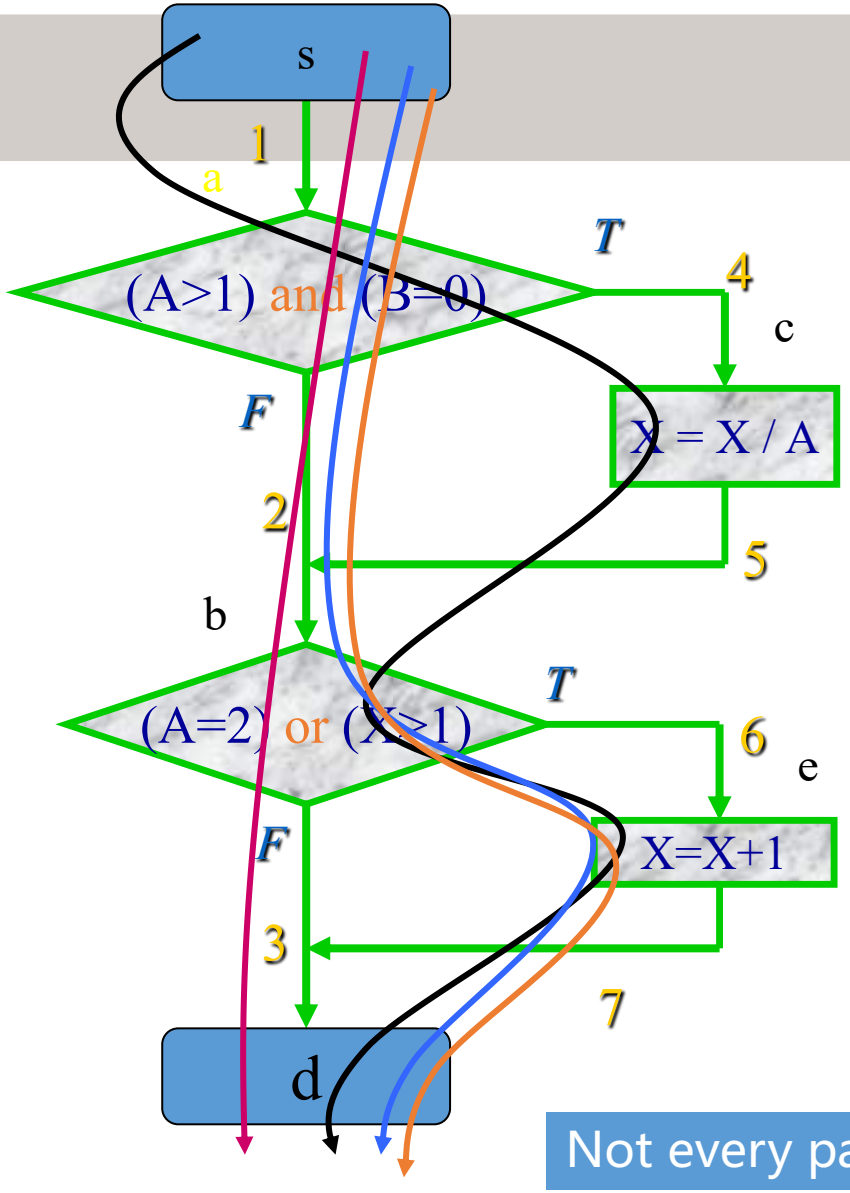
IV. A=1,B=1,X=1

Path
I: sacbed
II: sabed
III: sabed
IV: sabd

Meeting the criteria of conditional combination coverage means meeting the criteria of decision coverage, conditional coverage and decision/conditional coverage.

Four testcases allow each of the eight condition combinations to occur at least once:

TestCase			Path	Conditions	Condition Combination	Decisions	Expected Output
A	B	X					
2	0	4	sacbed	T1,T2,T3,T4	1, 5	TT	3
2	1	1	sabed	$T1,\overline{T2},T3,\overline{T4}$	2, 6	FT	2
1	0	2	sabed	$\overline{T1},T2,\overline{T3},T4$	3, 7	FT	3
1	1	1	sabd	$\overline{T1},\overline{T2},\overline{T3},\overline{T4}$	4, 8	FF	1



1. A > 1, B = 0
2. A > 1, B≠0
3. A≯1, B = 0
4. A≯1, B≠0

I. A=2,B=0,X=4

II. A=2,B=1,X=1

5. A = 2, X > 1
6. A = 2, X≯1
7. A≠2, X > 1
8. A≠2, X≯1

III. A=1,B=0,X=2

IV. A=1,B=1,X=1

Path
I: sacbed
II: sabed
III: sabed
IV: sabd

Not every path in the program can be executed , for example, sacbd.

Airline Seat reservation Example

Program Code:

```
(1) public static Boolean seatsAvailable(
       int freeSeats, int seatsRequired)
(2) {
(3)    boolean rv=false;
(4)    if ( (freeSets>=0) && (seatsRequired>=1) &&
            (seatsRequired<=freeSeats) )
(5)       rv=true
(6)    return rv;
(7) }
```

In the program there is one decision with three conditions on line 4.

```
if ( (freeSeats>=0) && (seatsRequired>=1)
      && (seatsRequired<=freeSeats) )
```

Airline Seat reservation Example

Case	freeSeats>=0	seatsRequired>=1	seatsRequired<=freeSeats
1	T	T	T
2	T	T	F
3	T	F	T
4	T	F	F
5	F	T	T
6	F	T	F
7	F	F	T
8	F	F	F

The shaded rows highlight impossible Test Cases that we cannot test

Airline Seat reservation ----Test Cases

Test No.	Test Cases/Combinations Covered	Inputs		Expected Outputs
		freeSeats	*seatsRequired*	*Return Value*
1	1	50	25	True
2	2	50	75	False
3	3	50	-25	False
4	6	-50	25	False
5	7	-50	-75	False
6	8	-50	-25	False

Condition Combination Coverage Strengths & Weaknesses

- Tests all possible combinations of conditions in every decision

- Can be expensive: n conditions in a decision give 2^n test cases
- Can be difficult to determine the required input parameter values

- Even though multiple condition testing covers every possible combination of conditions in a decision, it does not cause every possible execution path to be taken

EX. Grade Specification

The program Grade combines an exam and coursework mark into a single grade. The values for exam and coursework are integers.

If the exam or coursework mark is less than 50% then the grade returned is a ‘Fail’. To pass the course with a ‘Pass, C’, the student must score between 50% and 60% in the exam, and at least 50% in the coursework.

They will pass the course with ‘Pass, B’, if they score over 60% in the exam and 50% in the coursework.

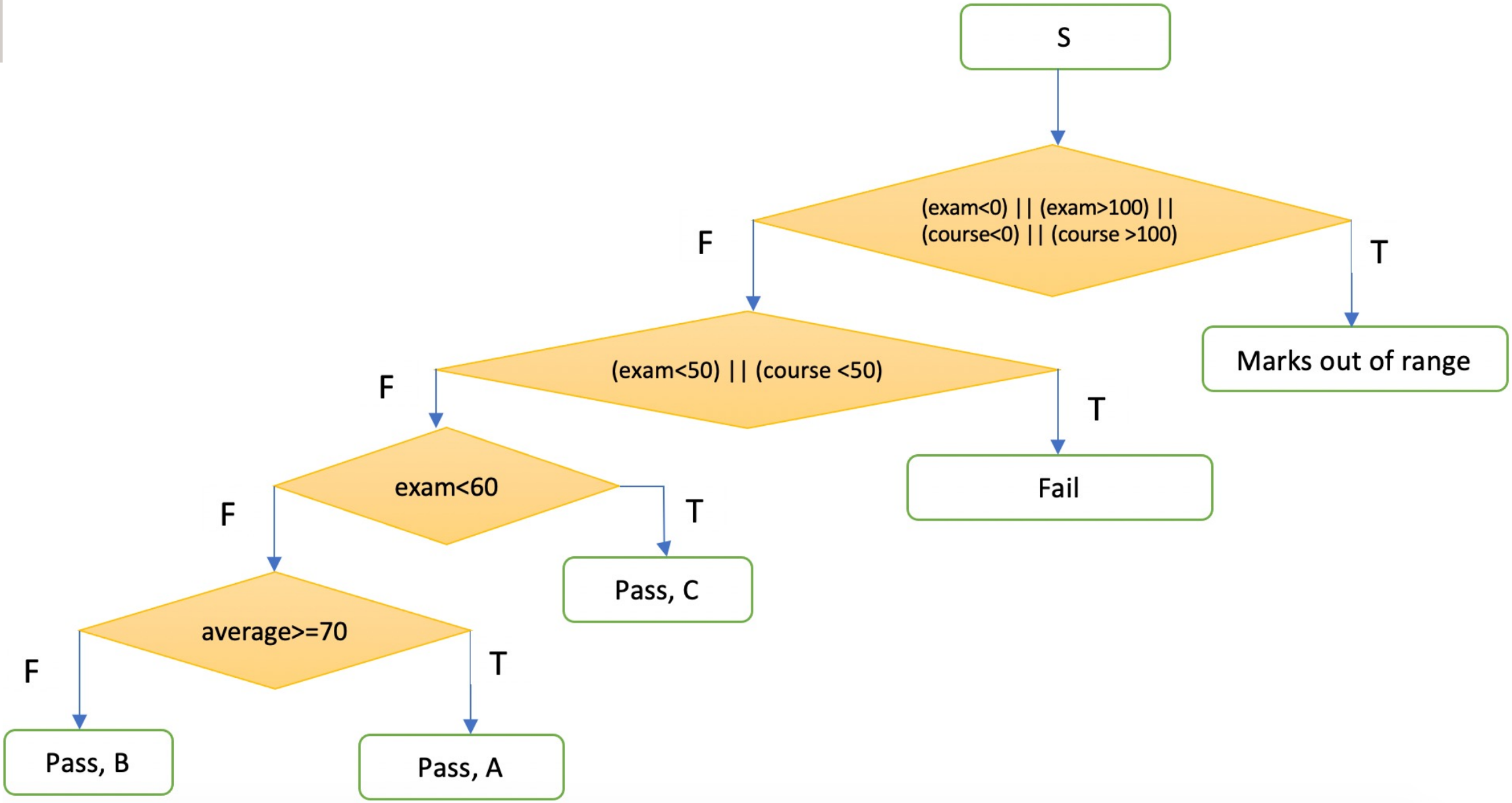
In addition to this, if the average of the exam and the coursework is at least 70%, then they are awarded a ‘Pass, A’. Input values that are less than 0 or greater than 100 for either the exam or coursework are invalid and the program will return a message to say ‘Marks out of range’.

```
1  public static String Grade (int exam, int course) {
2  String result="null";
3  long average;
4  average = Math.round((exam+course)/2);
5  if ( (exam<0) || (exam>100) || (course<0) || (course>100) )
6  result="Marks out of range";
7  else {
8  if ( (exam<50) || (course<50)) {
9  result="Fail";
10 }
11 else if (exam < 60) {
12 result="Pass,C";
13 }
14 else if ( average >= 70) {
15 result="Pass,A";
16 }
17 else {
18 result="Pass,B";
19 }
20 }
21 return result;
22 }
```

1) Please draw the program flow chart of the above code

2) Please list all the decisions and their conditions of the above program.

3) Please use the Condition Combination coverage testing method to design the testcases for the above code


S
(exam<0) || (exam>100) ||
(course<0) || (course >100)
F
T
Marks out of range
(exam<50) || (course <50)
F
T
Fail
exam<60
F
T
Pass, C
average>=70
F
T
Pass, B
Pass, A

- **Decision and Conditions**
- On line 5/node 1 there is one decision with four conditions

```
if ((exam<0) || (exam>100) ||
  (course<0) || (course>100))
```

- On line 8/node 3 there is one decision with two conditions

```
if ((exam<50) || (course<50))
```

- On line 11/node 5 there is one decision with one condition

```
else if (exam < 60)
```

- On line 14/node 7 there is one decision with one condition

```
else if (average >= 70)
```

Case	exam<0	exam>100	course<0	Course>100
1	T	T	T	T
2	T	T	T	F
3	T	T	F	T
4	T	T	F	F
5	T	F	T	T
6	T	F	T	F
7	T	F	F	T
8	T	F	F	F
9	F	T	T	T
10	F	T	T	F
11	F	T	F	T
12	F	T	F	F
13	F	F	T	T
14	F	F	T	F
15	F	F	F	T
16	F	F	F	F

if ((exam<0) || (exam>100) || (course<0) || (course>100))

Test Cases and Test Data

Case	exam<50	course<50
17	T	T
18	T	F
19	F	T
20	F	F

if ((exam<50) || (course<50))

Case	exam<60
21	T
22	F

else if (exam<60)

Case	average>=70
23	T
24	F

else if (average>=70)

Test Cases and Test Data

Test No.	Test Cases/Multiple Conditions Covered	Inputs		Expected Outputs
		exam	*course*	*Result*
1	6	-1	-1	Marks out of Range
2	7	-1	101	Marks out of Range
3	8	-1	40	Marks out of Range
4	10	101	-1	Marks out of Range

Test Cases and Test Data

Test No.	Test Cases/Multiple Conditions Covered	Inputs		Expected Outputs
		exam	*course*	*Result*
5	11	101	101	Marks out of Range
6	12	101	40	Marks out of Range
7	14	40	-1	Marks out of Range
8	15	40	101	Marks out of Range

Test Cases and Test Data

Test No.	Test Cases/Multiple Conditions Covered	Inputs		Expected Outputs
		exam	*course*	*Result*
9	16, 17	49	49	Fail
10	16,18	49	51	Fail
11	16,19	51	49	Fail
12	16, 20, 21	59	100	Pass, C
13	16, 20, 22, 23	75	75	Pass, A
14	16, 20, 22, 24	61	61	Pass, B