# 软件测试 Software Testing
# 10. 逻辑覆盖

程适

cheng@snnu.edu.cn

计算机科学学院

2016 年 10 月 13 日


陕西師範大學
SHAANXI NORMAL UNIVERSITY

# Outline

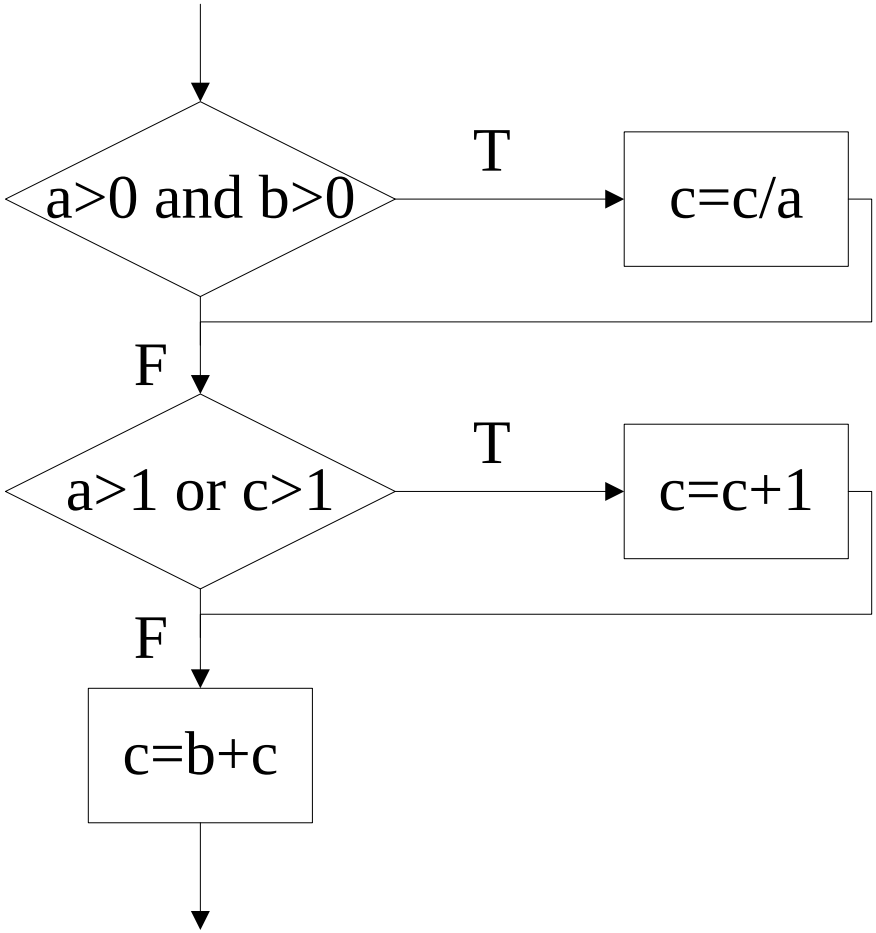
- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定-条件覆盖
- 条件组合覆盖
- 路径覆盖

# 结构性测试

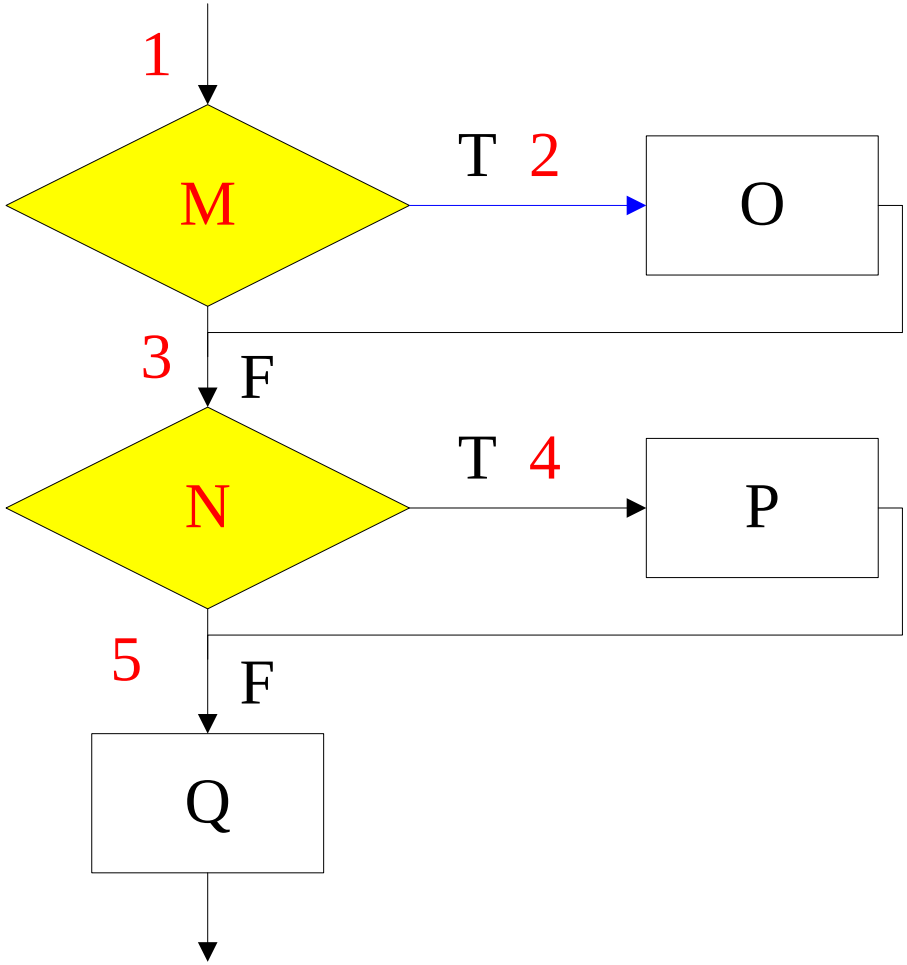
- 逻辑覆盖，是以程序内部的逻辑结构为基础的设计测试用例的技术。
  - 语句覆盖
  - 判定覆盖
  - 条件覆盖
  - 判定-条件覆盖
  - 条件组合覆盖
  - 路径覆盖
- 路径测试
- 数据流测试

# 逻辑覆盖

```
1  int a = 0;
2  int b = 0;
3  double c = 0.0;
4  if (a>0 && b>0)
5      c=c/a;
6  if (a>1 || c>1)
7      c=c+1;
8  c=b+c;
```



# 逻辑覆盖



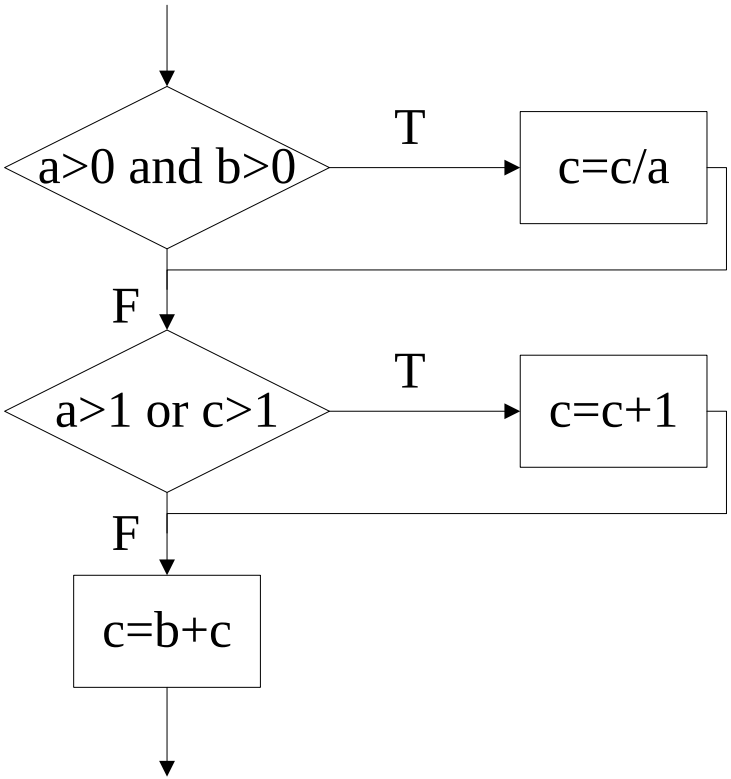
# 逻辑覆盖

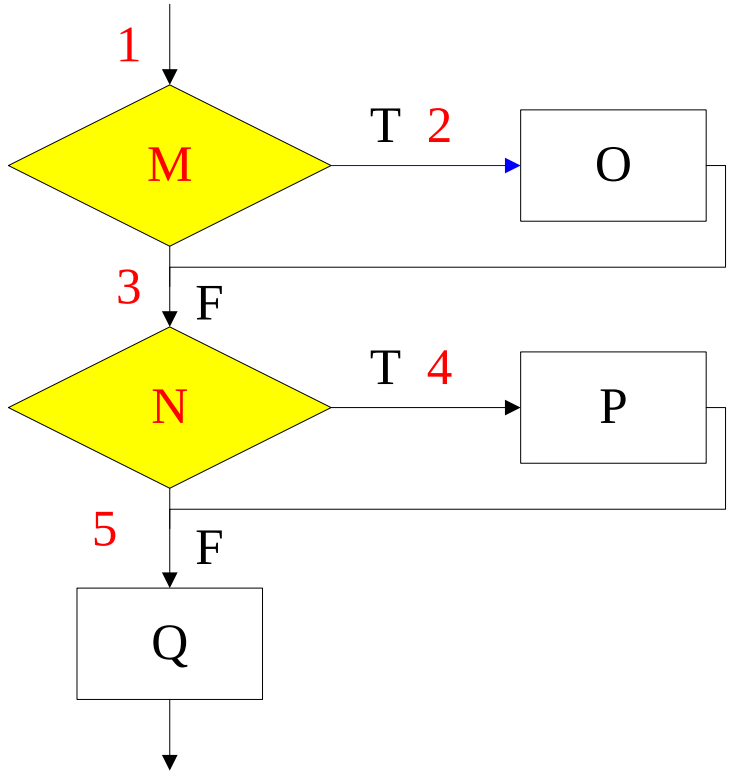
- P1:(1-2-4) P2:(1-2-5) P3:(1-3-4) P4:(1-3-5)
- 将里面的判定条件和过程记录如下：
  - 条件M{a > 0 and b > 0}
  - 条件N = {a > 1 or c > 1}
  - 这样，程序的4条不同路径可以表示为：
  - P1:(1-2-4)=M and N P2:(1-2-5)=M and ∼N
  - P3:(1-3-4)=∼M and N P4:(1-3-5)=∼M and ∼N

# 语句覆盖

- 基本思想是：设计若干测试用例，运行被测程序，使程序中每个可执行语句至少执行一次。
- P1:(1-2-4)=M and N　令$a = 2$, $b = 1$, $c = 6$, 此时满足条件M{a > 0 and b > 0} 和条件N = {a > 1 or c > 1}, （注：此时$c = c/a = 3$），
- 这样，测试用例的输入$\{a = 2, b = 1, c = 6\}$ 和对应的输出$\{a = 2, b = 1, c = 5\}$覆盖路径P1。

# 语句覆盖



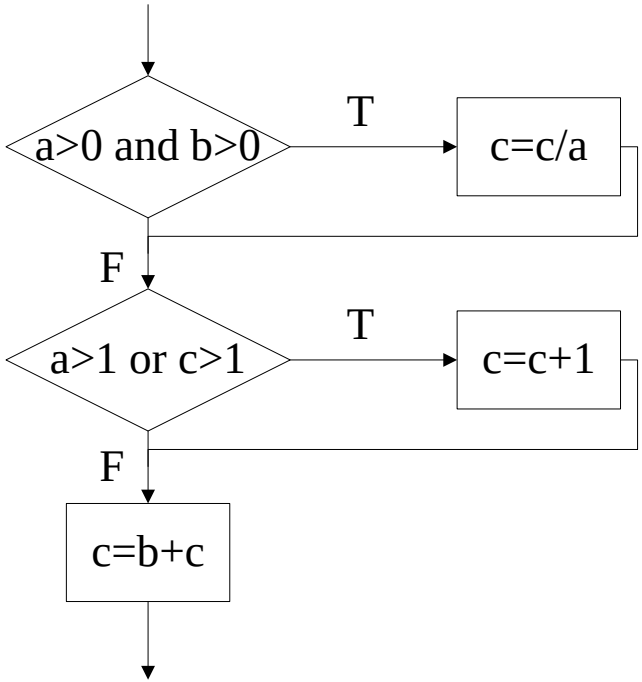



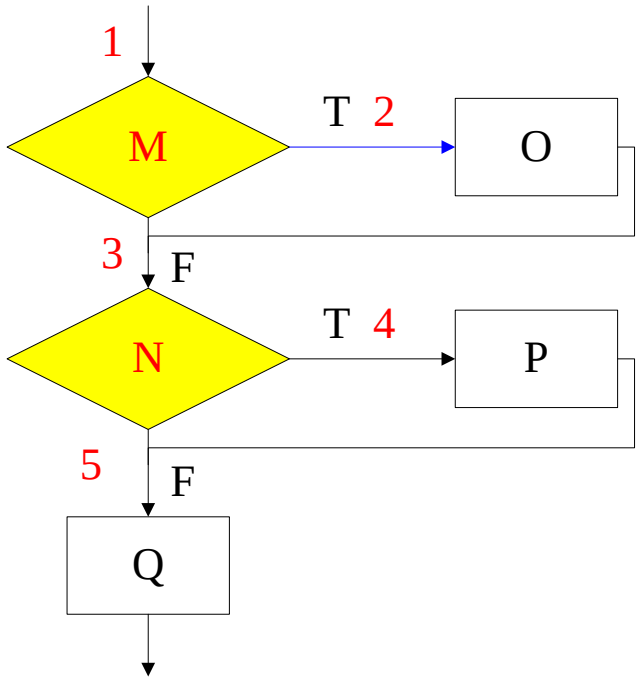
# 语句覆盖

- 优点：可以很直观地从源代码得到测试用例，无须细分每条判定表达式。
- 缺点：由于这种测试方法仅仅针对程序逻辑中显式存在的语句，但对于隐藏的条件和可能到达的隐式逻辑分支，是无法测试的。语句覆盖对于多分支的逻辑运算也是无法全面反映的。语句覆盖可能发现不了判断中逻辑运算中出现的错误；

# 判定覆盖

- 基本思想是：设计若干测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断真假值均曾被满足。



# 判定覆盖测试用例设计

| 测试用例 | 判定M的取值 | 判定N的取值 | 覆盖路径 |
|---|---|---|---|
| 输入：a=2,b=1,c=6<br>输出：a=2,b=1,c=5 | T | T | P1(1-2-4) |
| 输入：a=-1,b=-2,c=-3<br>输出：a=-1,b=-2,c=-5 | F | F | P4(1-3-5) |
| 输入：a=1,b=1,c=-3<br>输出：a=1,b=1,c=-2 | T | F | P2(1-2-5) |
| 输入：a=-1,b=2,c=3<br>输出：a=-1,b=2,c=6 | F | T | P3(1-3-4) |

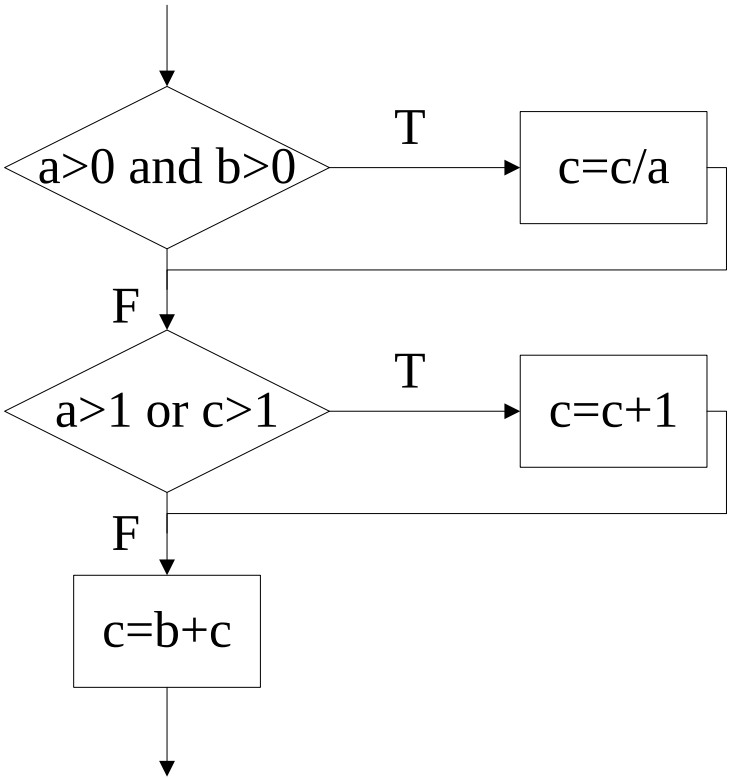
# 判定覆盖

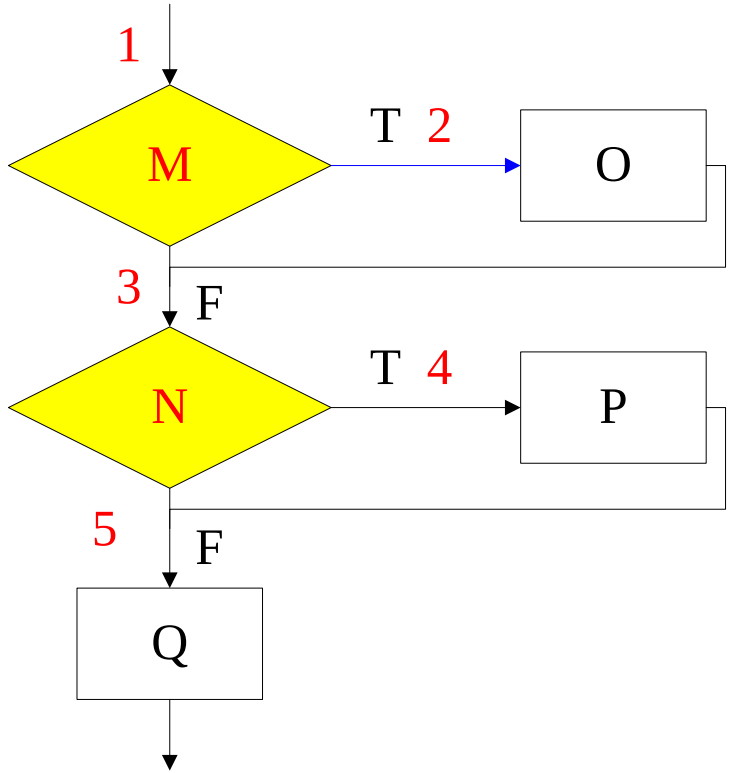
- 优点：判定覆盖比语句覆盖要多几乎一倍的测试路径，当然也就具有比语句覆盖更强的测试能力。同样判定覆盖也具有和语句覆盖一样的简单性，无须细分每个判定就可以得到测试用例。
- 缺点：往往大部分的判定语句是由多个逻辑条件组合而成（如，判定语句中包含AND、OR、CASE），若仅仅判断其整个最终结果，而忽略每个条件的取值情况，必然会遗漏部分测试路径。判定覆盖不能保证一定能查出在判断条件中存在的错误。

# 条件覆盖

- 基本思想是：设计若干测试用例，执行被测程序以后要使每个判断中每个条件的可能取值至少满足一次。
- 对于M：$a > 0$ 取真时T1，取假时F1; $b > 0$取真时T2，取假时F2；
- 对于N：$a > 1$取真时T3，取假时F3; $c > 1$取真时T4，取假时F4。

# 条件覆盖



# 条件覆盖测试用例设计

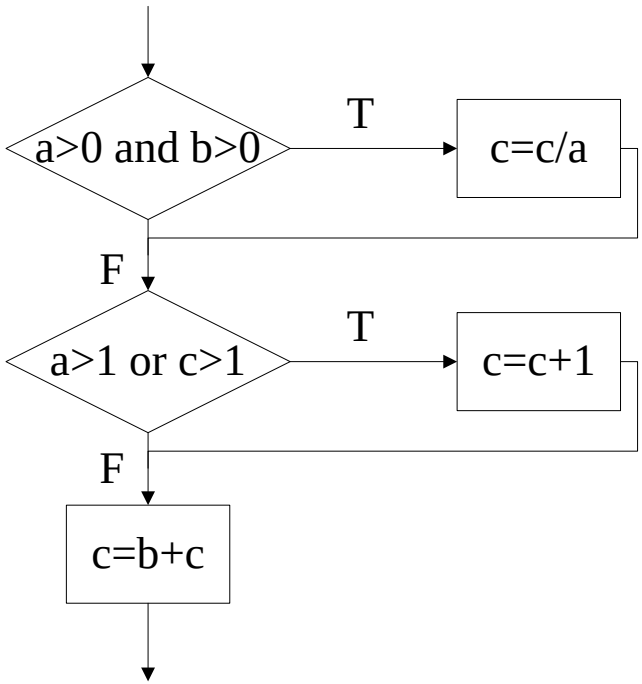
| 测试用例 | 取值条件 | 具体取值条件 | 覆盖路径 |
| --- | --- | --- | --- |
| 输入：a=2,b=-1,c=-2<br>输出：a=2,b=-1,c=-2 | T1, F2, T3, F4 | $a > 0$, $b \leq 0$, $a > 1$, $c \leq 1$ | P3(1-3-4) |
| 输入：a=-1,b=2,c=3<br>输出：a=-1,b=2,c=6 | F1, T2, F3, T4 | $a \leq 0$, $b > 0$ ,$a \leq 1$, $c > 1$ | P3(1-3-4) |

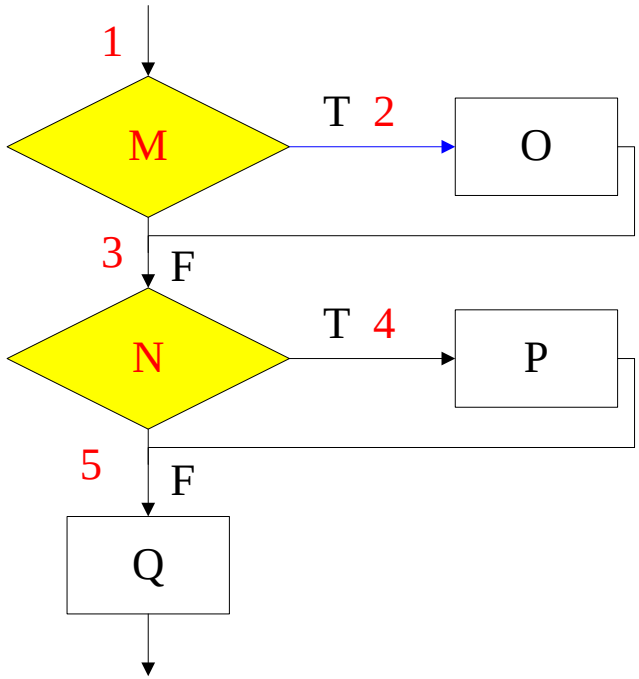
# 条件覆盖

- 优点：显然条件覆盖比判定覆盖，增加了对符合判定情况的测试，增加了测试路径。
- 缺点：要达到条件覆盖，需要足够多的测试用例，但条件覆盖并不能保证判定覆盖。条件覆盖只能保证每个条件至少有一次为真，而不考虑所有的判定结果。

# 判定-条件覆盖

- 基本思想是：设计足够的测试用例，使得判断条件中的所有条件可能至少执行一次取值，同时，所有判断的可能结果至少执行一次。



# 判定-条件覆盖测试用例设计

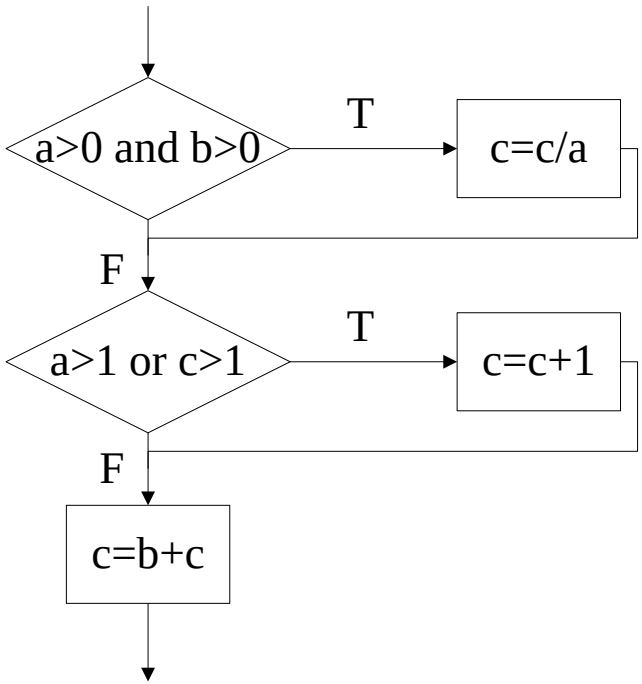
| 测试用例 | 取值条件 | 具体取值条件 | 覆盖路径 |
|---|---|---|---|
| 输入：a=2,b=1,c=6<br>输出：a=2,b=1,c=5 | T1, T2, T3, T4 | $a > 0, b > 0, a > 1, c > 1$ | P1(1-2-4) |
| 输入：a=-1,b=-2,c=-3<br>输出：a=-1,b=-2,c=-5 | F1, F2, F3, F4 | $a \leq 0, b \leq 0, a \leq 1, c \leq 1$ | P4(1-3-5) |

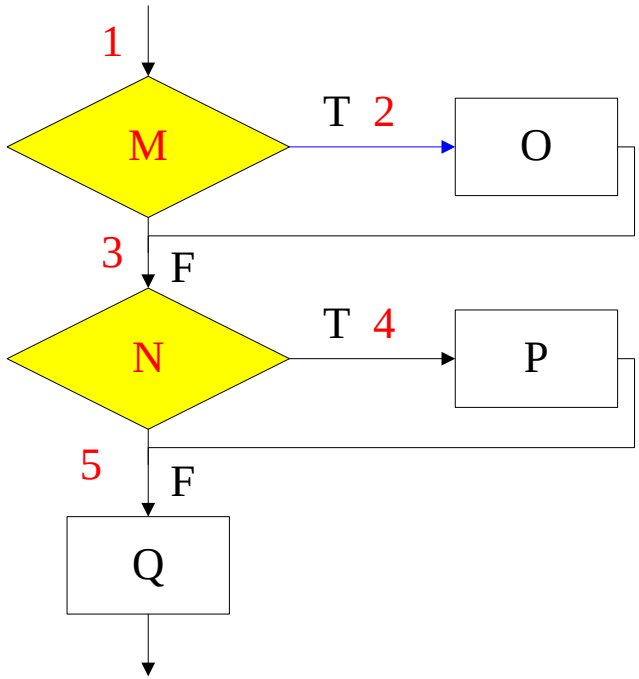
# 判定-条件覆盖

- 优点：判定/条件覆盖满足判定覆盖准则和条件覆盖准则，弥补了二者的不足。
- 缺点：判定/条件覆盖准则的缺点是未考虑条件的组合情况。

# 条件组合覆盖

- 基本思想是：设计足够的测试用例，使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次。



# 判定覆盖测试用例设计

| 编号 | 覆盖条件取值 | 判定条件取值 | 判定-条件组合 |
|---|---|---|---|
| 1 | T1, T2 | M | $a > 0,\ b > 0$ |
| 2 | T1, F2 | $\sim$M | $a > 0,\ b \leq 0$ |
| 3 | F1, T2 | $\sim$M | $a \leq 0,\ b > 0$ |
| 4 | F1, F2 | $\sim$M | $a \leq 0,\ b \leq 0$ |
| 5 | T3, T4 | N | $a > 1,\ b > 1$ |
| 6 | T3, F4 | N | $a > 1,\ b \leq 1$ |
| 7 | F3, T4 | N | $a \leq 1,\ b > 1$ |
| 8 | F3, F4 | $\sim$N | $a \leq 1,\ b \leq 1$ |

# 条件组合覆盖测试用例设计

- 针对以上8种条件组合，来设计所有能覆盖这些组合的测试用例：

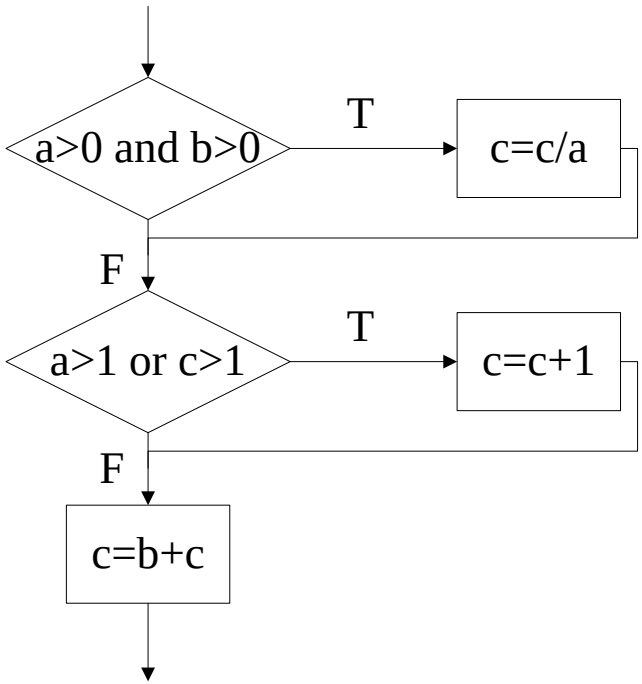
| 测试用例 | 覆盖条件 | 覆盖路径 | 覆盖组合 |
|---|---|---|---|
| 输入：a=2,b=1,c=6<br>输出：a=2,b=1,c=5 | T1, T2, T3, T4 | P1(1-2-4) | 1，5 |
| 输入：a=2,b=-1,c=-2<br>输出：a=2,b=-1,c=-3 | T1, F2, T3, F4 | P3(1-3-4) | 2，6 |
| 输入：a=-1,b=2,c=3<br>输出：a=-1,b=2,c=6 | F1, T2, F3, T4 | P3(1-3-4) | 3，7 |
| 输入：a=-1,b=-2,c=-3<br>输出：a=-1,b=-2,c=-5 | F1, F2, F3, F4 | P4(1-3-5) | 4，8 |

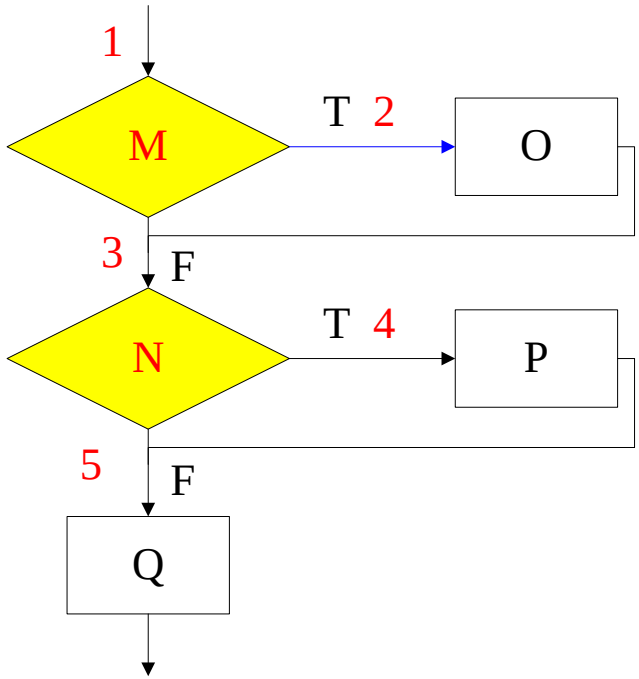
# 条件组合覆盖

- 优点：多重条件覆盖准则满足判定覆盖、条件覆盖和判定/条件覆盖准则。更改的判定/条件覆盖要求设计足够多的测试用例，使得判定中每个条件的所有可能结果至少出现一次，每个判定本身的所有可能结果也至少出现一次。并且每个条件都显示能单独影响判定结果。
- 缺点：线性地增加了测试用例的数量。

# 路径覆盖

- 基本思想是：设计所有的测试用例，来覆盖程序中的所有可能的执行路径



# 判定覆盖测试用例设计

- 路径覆盖法没有涵盖所有的条件覆盖组合，如组合2、3、6。

| 测试用例 | 覆盖条件 | 覆盖路径 | 覆盖组合 |
|---|---|---|---|
| 输入：a=2,b=1,c=6<br>输出：a=2,b=1,c=5 | T1, T2, T3, T4 | P1(1-2-4) | 1，5 |
| 输入：a=1,b=1,c=-3<br>输出：a=1,b=-1,c=-2 | T1, T2, F3, F4 | P2(1-2-5) | 1，8 |
| 输入：a=-1,b=2,c=3<br>输出：a=-1,b=2,c=6 | F1, F2, F3, T4 | P3(1-3-4) | 4，7 |
| 输入：a=-1,b=-2,c=-3<br>输出：a=-1,b=-2,c=-5 | F1, F2, F3, F4 | P4(1-3-5) | 4，8 |

# 路径覆盖

- 优点：这种测试方法可以对程序进行彻底的测试，比前面五种的覆盖面都广。
- 缺点：由于路径覆盖需要对所有可能的路径进行测试（包括循环、条件组合、分支选择等），那么需要设计大量、复杂的测试用例，使得工作量呈指数级增长。

# 路径覆盖

- 在有些情况下，一些执行路径是不可能被执行的，如：

  if (A) B++;

  if (!A) D−−;

- 这两个语句实际只包括了2条执行路径，即A为真或假时候对B 和D的处理，真或假不可能都存在，而路径覆盖测试则认为是包含了真与假的4条执行路径。这样不仅降低了测试效率，而且大量的测试结果的累积，也为排错带来麻烦。

# 逻辑覆盖

- 从前面的例子我们可以看到，采用任何一种覆盖方法都不能满足我们的要求，所以，在实际的测试用例设计过程中，可以根据需要和不同的测试用例设计特征，将不同的覆盖方法组合起来使用，以实现最佳的测试用例设计。

# 条件组合和路径覆盖结合设计测试用例

| 测试用例 | 覆盖条件 | 覆盖路径 | 覆盖组合 |
|---|---|---|---|
| 输入：a=2,b=1,c=6<br>输出：a=2,b=1,c=5 | T1, T2, T3, T4 | P1(1-2-4) | 1，5 |
| 输入：a=1,b=1,c=-3<br>输出：a=1,b=-1,c=-2 | T1, T2, F3, F4 | P2(1-2-5) | 1，8 |
| 输入：a=-1,b=2,c=3<br>输出：a=-1,b=2,c=6 | F1, F2, F3, T4 | P3(1-3-4) | 4，7 |
| 输入：a=-1,b=-2,c=-3<br>输出：a=-1,b=-2,c=-5 | F1, F2, F3, F4 | P4(1-3-5) | 4，8 |
| 输入：a=2,b=-1,c=-2<br>输出：a=2,b=-1,c=-2 | T1, F2, T3, F4 | P3(1-3-4) | 2，6 |
| 输入：a=-1,b=2,c=3<br>输出：a=-1,b=2,c=6 | F1, T2, F3, T4 | P3(1-3-4) | 3，7 |

# 逻辑覆盖

- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定-条件覆盖
- 条件组合覆盖
- 路径覆盖

# 练习

1. 动态白盒测试与调试有何区别？
2. 如何辨别发现的软件缺陷是软件Bug还是特定的配置或系统问题？
3. 使用不同的逻辑覆盖测试方法，写出三角形问题的测试用例。

```
1    if (a+b<c || a+c<b || b+c<a)
2        System.out.println(``can not constitute a triangle'');
3    if (a==b || a==c || b==c) {
4        if (a==b && b==c)
5        System.out.println(``equilateral triangle'');
6        else System.out.println (``isosceles triangle'');
7    }
8    else System.out.println(``scalene triangle'');
```

# 致谢

谢谢，欢迎提问!