软件测试考试

前五章是重点

学习通考，环境要自己配

PPT在学习通上

- 5个选择题（10分）
- 3个编程（每个10-15分）：简单题

  JUnit2道

  Selenium只有1题（考试重点在于写程序）
- 3-4个简答（每个10-15分）

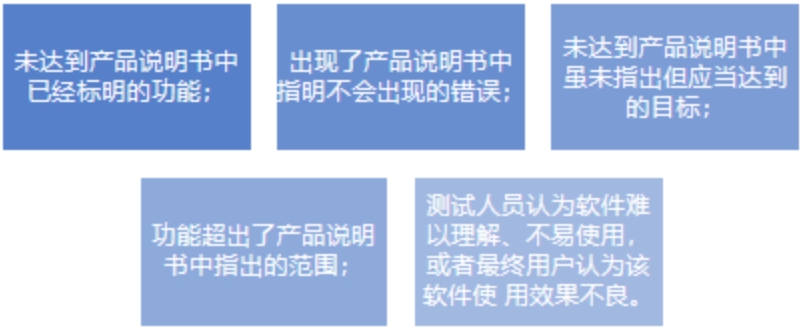
考前注意：

- eclipse的自动补全配置
- selenium的配置
- 浏览器驱动的配置
- 简答题的背诵

（一）测试基础理论知识（找课本相关位置）

1、软件缺陷

软件缺陷是软件产品中所存在的问题，最终表现为用户所需要的功能没有完全实现，没有满足用户的需求。



软件公司对缺陷严重性级别的定义一般可概括为以下几种：

致命的	严重的	一般的	微小的
• 致命的错误，造成系统或应用程序崩溃、死机、系统悬挂，或造成数据丢失、主要功能完全丧失等。	• 严重错误，指功能或特性没有实现，主要功能丧失，导致严重的问题，或致命的错误声明。	• 不太严重的错误，这样的软件缺陷虽然不影响系统的基本使用， 但没有很好地实现功能，没有达到预期效果。如次要功能丧失，提示信息 不太准确，或用户界面差，操作时间长等。	• 一些小问题，对功能几乎没有影响，产品及属性仍可使用，如有个别错误字、文字排列不整齐等。

（二）、软件缺陷的产生

软件缺陷的产生是不可避免的，那么造成软件缺陷的原因是什么呢?

1. 需求解释有错误;
2. 用户需求定义错误;
3. 需求记录错误;
4. 设计说明有误;
5. 编码说明有误;
6. 程序代码有误;
7. 其他如：数据输入有误，问题修改不正确。

2、软件测试基础理论

软件测试就是在软件投入运行前，对软件需求分析、设计规格说明和编码实现的最终审查，它是软件质量保证的关键步骤。

软件测试是为了尽快尽早地发现在软件产品中所存在的各种软件缺陷而展开的贯穿整个软件开发生命周期、对软件产品（包括阶段性产品）进行验证和确认的活动过程。

该测试过程包括6个主要活动如下所述。

（1）测试计划，确定测试基本原则、生成测试概要设计。
（2）测试需求分析。
（3）测试设计，包括测试用例设计和测试规程规格说明。
（4）测试规程实现。
（5）测试执行。
（6）总结生成报告。

3、软件生命周期，质量

4、测试的关键问题四个方面

5、软件测试的目的三个

6、阶段任务（8页）

7、测试的信息流

8、测试的策略

9、测试的分类

> 从是否需要执行被测软件的角度看分为：

静态测试

动态测试

> 从是否针对系统内部结构和具体实现算法的角度看分为：

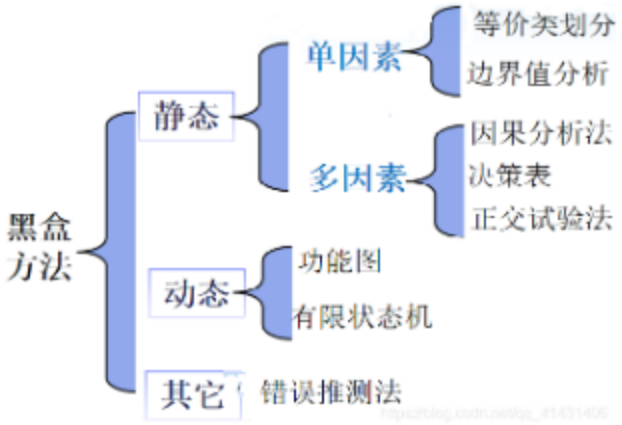
黑盒测试：不关注具体的代码逻辑，只关心系统能否对给定的输入输出期望的值

白盒测试：需要写代码，需要关注程序具体的执行流程

10、动态测试基本内容（24页）

11、测试的管理过程（88页）

（二）测试分类：黑盒测试



黑盒测试方法：

等价类划分法 边界值分析法 判定表方法 因果图法 场景法 正交试验法 功能图法 错误推测

1、等价类划分

从这个集合（等价类）中随便找一个数据就可以代表整个集合，因此在等价类中只需要取一组测试用例即可。等价类集合的划分，提供完备性、保证无冗余性。

划分完等价类，每一条数据对应一个测试用例

测试用例应该尽可能多的覆盖有效类

设计用例只覆盖一个无效类

2、确定等价类的原则

1）输入条件规定**取值范围**，则卡定义**一个有效等价类**和**两个无效等价类**。例如学生成绩范围是0~100，则一个有效类：0<=成绩<=100,两个无效类：成绩<0,成绩>100

2）如果规定了**输入数据的个数**，则可类似的划分出**一个等价类**和**两个无效等价类**。例如一个学生一个学期选修1~3门，则一个有效等价类：1 ~ 3门，两个无效等价类：不选，选修 超过3门

3）如果规定数据的**一组值**，且程序对不同的输入做不同的处理，则每个允许的输入值是一个有效等价类，所有不允许的输入值的集合是一个无效等价类。例如输入条件说明学历可为：专科、本科、硕士、博士4种，则**一个有效等价类**：专科/本科/硕士/博士,**一个无效等价类**：其他任何学历

4）如果规定了输入数据必须遵循的规则，可以确定**一个有效类和若干个无效等价类**。例如校内电话拨号为 9开头，则一个有效等价类：9+外线号码，若干个无效等价类：非9开头+外线号码，9+非外线号码，…

5）如果确知已划分的等价类中个元素在程序中的处理方式不同，则应将此等价类进一步划分成更小的等价类

1. 有效等价类：满足需求的数据
2. 无效等价类：不满足需求的数据
3. 说等价类**弱**是因为是单缺陷（**弱==单缺陷，强==多缺陷**）
4. 说等价类**健壮**是因为考虑无效值（**一般==不考虑无效值，健壮==考虑无效值**）

5. 说最坏是因为考虑多变量同时取极值（多缺陷）

例1：报表日期

设某公司要打印2001～2010年的报表，其中报表日期为6位数字组成，其中，前4位为年份，后两位为月份。

第一步：划分等价类：

输入及外部条件	有效等价类	无效等价类
报表日期的 类型及长度	6位数字字符①	有非数字字符 ④ 少于6个数字字符 ⑤ 多于6个数字字符 ⑥
年份范围	在2001～2010之间②	小于2001 ⑦ 大于2010 ⑧
月份范围	在1～12之间③	小于1 ⑨ 大于12 ⑩

第二步：为有效等价类设计测试用例

对表中编号为①②③的3个有效等价类用一个测试用例覆盖：

测试数据	期望结果	覆盖范围
200105	**输入有效**	**等价类①②③**

第三步：为每一个无效等价类至少设计一个测试用例（控制变量原则，一次性只出现一个无效等价类）

测试数据	期望结果	覆盖范围
001MAY	输入无效	等价类④
20015	输入无效	等价类⑤
2001001	输入无效	等价类⑥
200001	输入无效	等价类⑦
201201	输入无效	等价类⑧
200100	输入无效	等价类⑨
200113	输入无效	等价类⑩

3、边界值分析法

【前言】

边界值测试（4N+1）、健壮性测试（6N+1）是单缺陷，最坏情况（5N）&健壮最坏情况测试（7N）是多缺陷。

1 思想原理

关注输入空间的边界，错误更可能出现在输入变量的极值附近。在最小值、略高于最小值、正常值、略低于最大值、最大值（5个）处取输入变量的值。失效极少是由两个（或两个以上）缺陷同时发生引起的。

与等价划分的区别 ：

边界值分析不是从某等价类中随便挑一个作为代表，而是使这个等价类的每个边界都要作为测试条件。 边界值分析不仅考虑输入条件，还要考虑输出空间产生的测试情况。

常见的边界值：

1)对16-bit 的整数而言 32767 和 -32768 是边界

2)屏幕上光标在最左上、最右下位置

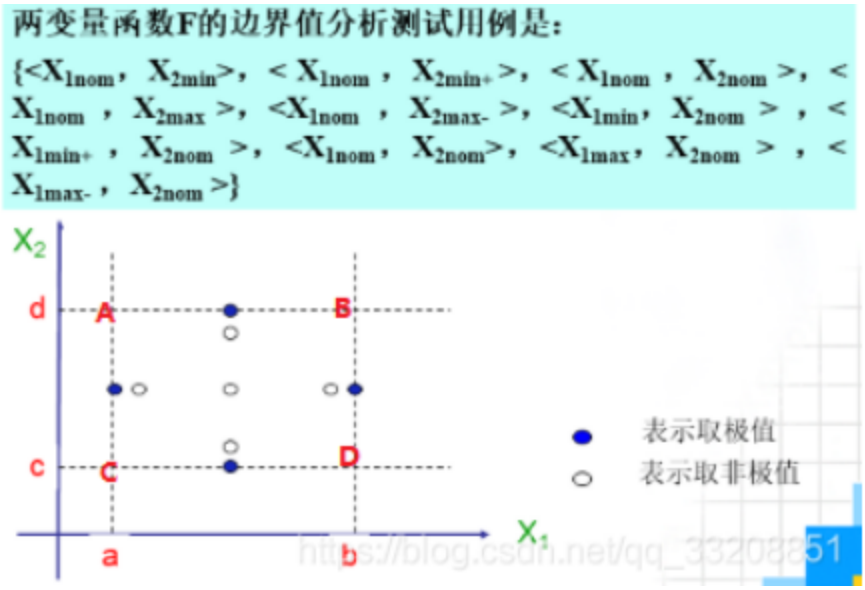
3)报表的第一行和最后一行

4)数组元素的第一个和最后一个

5)循环的第 0 次、第 1 次和倒数第 2 次、最后一次

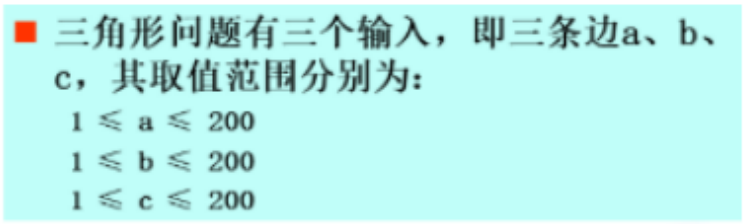
2 应用实例

两变量函数



单缺陷，不考虑A,B,C,D四个点

三角形边界



用例	a	b	c	预期输出
1	100	100	1	等腰三角形
2	100	100	2	等腰三角形
3	100	100	100	等边三角形
4	100	100	199	等腰三角形
5	100	100	200	非三角形
6	100	1	100	等腰三角形
7	100	2	100	等腰三角形
8	100	100	100	等边三角形
9	100	199	100	等腰三角形
10	100	200	100	非三角形
11	1	100	100	等腰三角形
12	2	100	100	等腰三角形
13	100	100	100	等边三角形
14	199	100	100	等腰三角形
15	200	100	100	非三角形

Triangle类：

```
package com.example.test.prjtriangle;

public class Triangle {
  protected long lborderA = 0;
  protected long lborderB = 0;
  protected long lborderC = 0;

  // Constructor
  public Triangle(long lborderA, long lborderB, long lborderC) {
```

```
  this.lborderA = lborderA;
  this.lborderB = lborderB;
  this.lborderC = lborderC;

}

/**
 * check if it is a triangle
 *
 * @return true for triangle and false not
 */
public boolean isTriangle(Triangle triangle) {
  boolean isTriangle = false;

  // check boundary
  if ((triangle.lborderA > 0 && triangle.lborderA <= Long.MAX_VALUE)
      && (triangle.lborderB > 0 && triangle.lborderB <= Long.MAX_VALUE)
      && (triangle.lborderC > 0 && triangle.lborderC <= Long.MAX_VALUE)) {

    // check if subtraction of two border larger than the third
    if (diffOfBorders(triangle.lborderA, triangle.lborderB) < triangle.lborderC
        && diffOfBorders(triangle.lborderB, triangle.lborderC) < triangle.lborderA
        && diffOfBorders(triangle.lborderC, triangle.lborderA) < triangle.lborderB) {
      isTriangle = true;
    }

  }
  return isTriangle;
}

/**
 * Check the type of triangle
 *
 * Consists of "Illegal", "Regular", "Scalene", "Isosceles"
 */
public String getType(Triangle triangle) {
  String strType = "Illegal";

  if (isTriangle(triangle)) {
    // Is Regular
    if (triangle.lborderA == triangle.lborderB
        && triangle.lborderB == triangle.lborderC) {
      strType = "Regular";
    }
    // If scalene
    else if ((triangle.lborderA != triangle.lborderB)
        && (triangle.lborderB != triangle.lborderC)
        && (triangle.lborderA != triangle.lborderC)) {
      strType = "Scalene";//Regular
    }
    // if isosceles
    else {
      strType = "Isosceles";//等腰
```

```
      }
    }

    return strType;
  }

  /**
   * calculate the diff between borders
   *
   * */
  public long diffOfBorders(long a, long b) {
    return (a > b) ? (a - b) : (b - a);
  }

  /**
   * get length of borders
   */
  public long[] getBorders() {
    long[] borders = new long[3];
    borders[0] = this.lborderA;
    borders[1] = this.lborderB;
    borders[2] = this.lborderC;
    return borders;
  }
}
```

TriangleTest类：(只测试了“判断三角形类型的功能”，测试用例见上表)

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class TriangleTest {

  Triangle T1 = new Triangle(100,100,1);
  Triangle T2 = new Triangle(100,100,2);
  Triangle T3 = new Triangle(100,100,100);
  Triangle T4 = new Triangle(100,100,199);
  Triangle T5 = new Triangle(100,100,200);

  Triangle T6 = new Triangle(100,1,100);
  Triangle T7 = new Triangle(100,2,100);
  Triangle T8 = new Triangle(100,100,100);
  Triangle T9 = new Triangle(100,199,100);
  Triangle T10 = new Triangle(100,200,100);

  Triangle T11 = new Triangle(1,100,100);
  Triangle T12 = new Triangle(2,100,100);
  Triangle T13 = new Triangle(100,100,100);
```

```
  Triangle T14 = new Triangle(199,100,100);
  Triangle T15 = new Triangle(200,100,100);


  @Test
  void testGetType() {
    assertEquals("Isosceles", T1.getType(T1));
    assertEquals("Isosceles", T2.getType(T2));
    assertEquals("Regular", T3.getType(T3));
    assertEquals("Isosceles", T4.getType(T4));
    assertEquals("Illegal", T5.getType(T5));

    assertEquals("Isosceles", T6.getType(T6));
    assertEquals("Isosceles", T7.getType(T7));
    assertEquals("Regular", T8.getType(T8));
    assertEquals("Isosceles", T9.getType(T9));
    assertEquals("Illegal", T10.getType(T10));

    assertEquals("Isosceles", T11.getType(T11));
    assertEquals("Isosceles", T12.getType(T12));
    assertEquals("Regular", T13.getType(T13));
    assertEquals("Isosceles", T14.getType(T14));
    assertEquals("Illegal", T15.getType(T15));
  }

}
```

NextDate函数

- NextDate是一个有三个变量（月份、日和年）的函数，函数返回输入日期后面的那个日期。变量月份、日和年都具有整数值，且满足以下条件：
  - 1 ≤ 月份 ≤ 12
  - 1 ≤ 日 ≤ 31
  - 1812 ≤ 年 ≤ 2012

月份 = {1，2，6，11，12}
日 = {1，2，15，30，31}
年 = {1812，1813，1912，2011，2012}

用例	月份	日期	年	预期输出
1	6	15	1812	6/16/1812
2	6	15	1813	6/16/1813
3	6	15	1912	6/16/1912
4	6	15	2011	6/16/2011
5	6	15	2012	6/16/2012
6	6	1	1912	6/2/1912
7	6	2	1912	6/3/1912
8	6	15	1912	6/16/1912
9	6	30	1912	7/1/1912
10	6	31	1912	非法输入
11	1	15	1912	1/16/1912
12	2	15	1912	2/16/1912
13	6	15	1912	6/16/1912
14	11	15	1912	11/16/1912
15	12	15	1912	12/16/1912

3 总结

对于N变量函数，边界值分析会产生4N+1个测试用例

适用于多独立变量，简单机械

边界值测试用例设计SOP：

1、**划分边界：**

1. 最小值
2. 略高于最小值
3. 正常值（通常是区间的中值）
4. 略低于最大值
5. 最大值

2、**控制变量，一次性只测试一个变量的所有取值情况；其余变量固定为正常值（中值）**

3、**确定预期输出**

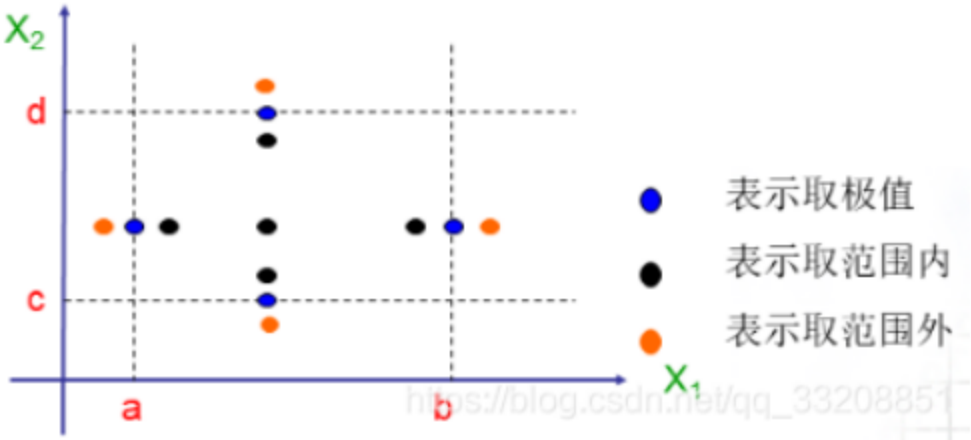
4、**以上测试用例去掉重复的，得出结论**

4、健壮性测试

健壮性测试是在边界值测试的基础上做的扩充

在边界值测试的基础上做的扩充：关注输入空间的边界，错误更可能出现在输入变量的极值附近。在`略小于最小值、最小值、略高于最小值、正常值、略低于最大值、最大值、略大于最大值`（7个）处取输入变量的值。失效极少是由两个（或两个以上）缺陷同时发生引起的。

两变量函数



三角形边界（仍然是控制变量思想）

- 三角形问题有三个输入，即三条边a、b、c，其取值范围为：
  - $1 \leqslant a \leqslant 200$
  - $1 \leqslant b \leqslant 200$
  - $1 \leqslant c \leqslant 200$

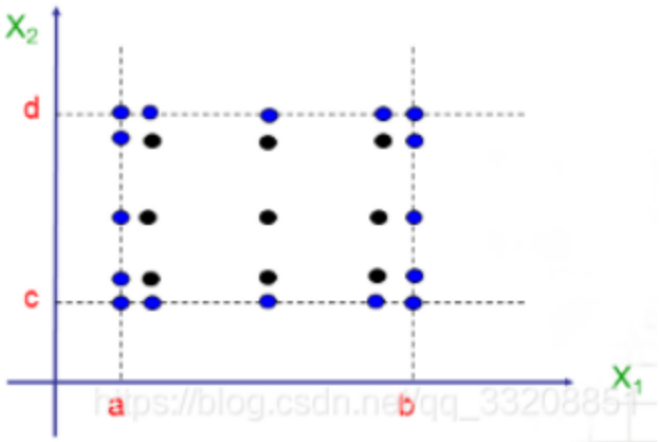
a = {0，1，2，100，199，200，201}
b = {0，1，2，100，199，200，201}
c = {0，1，2，100，199，200，201}

用例	a	b	c	预期输出
1	100	100	0	非法输入
2	100	100	1	等腰三角形
3	100	100	2	等腰三角形
4	100	100	100	等边三角形
5	100	100	199	等腰三角形
6	100	100	200	非三角形
7	100	100	201	非法输入
8	100	0	100	非法输入
9	100	1	100	等腰三角形
10	100	2	100	等腰三角形
11	100	100	100	等边三角形
12	100	199	100	等腰三角形
13	100	200	100	非三角形
14	100	201	100	非法输入
15	0	100	100	非法输入
16	1	100	100	等腰三角形
17	2	100	100	等腰三角形
18	100	100	100	等边三角形
19	199	100	100	等腰三角形
20	200	100	100	非三角形
21	201	100	100	

5、最坏情况测试

最坏情况测试关心的是`多变量同时取极值`（多缺陷）的情况，首先得到每个变量的`min,min+,nom,max-,max`（5个），进行**笛卡尔乘积**

两变量函数（笛卡尔积）



■ 两变量函数F的最坏情况测试用例是：

$X_1 = \{X_{1min}, X_{1min+}, X_{1nom}, X_{1max-}, X_{1max}\}$

$X_2 = \{X_{2min}, X_{2min+}, X_{2nom}, X_{2max-}, X_{2max}\}$

测试用例集合 $= X_1 \times X_2 = \{X_{1min}, X_{1min+}, X_{1nom}, X_{1max-}, X_{1max}\} \times \{X_{2min}, X_{2min+}, X_{2nom}, X_{2max-}, X_{2max}\}$

三角形边界（笛卡尔积）

- 三角形问题有三个输入，即三条边a、b、c，其取值范围为：

  $1 \leqslant a \leqslant 200$

  $1 \leqslant b \leqslant 200$

  $1 \leqslant c \leqslant 200$

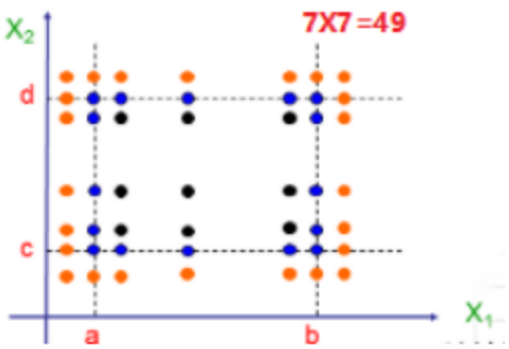
a = {1，2，100，199，200}
b = {1，2，100，199，200}
c = {1，2，100，199，200}

最坏情况测试用例集合 = a × b × c = {1，2，100，199，200} × {1，2，100，199，200} × {1，2，100，199，200}

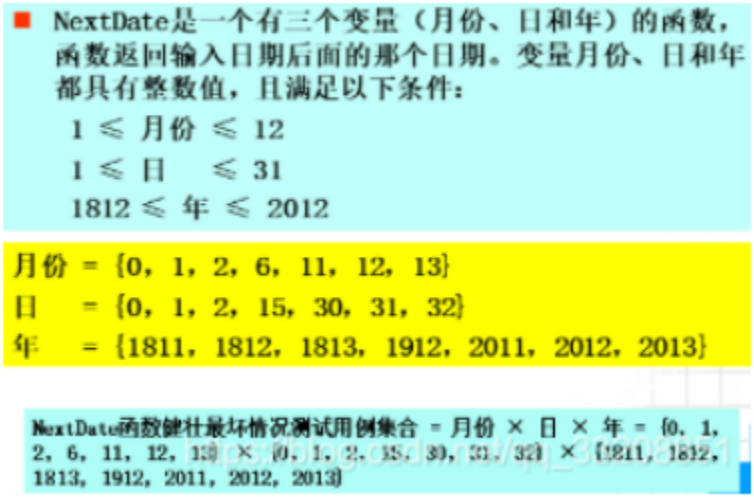
6、健壮最坏情况测试

健壮最坏情况测试关心的是`多变量同时取极值`（多缺陷）的情况，首先得到每个变量的`min-,min,min+,nom,max-,max,max+`（7个），进行**笛卡尔乘积**

两变量函数

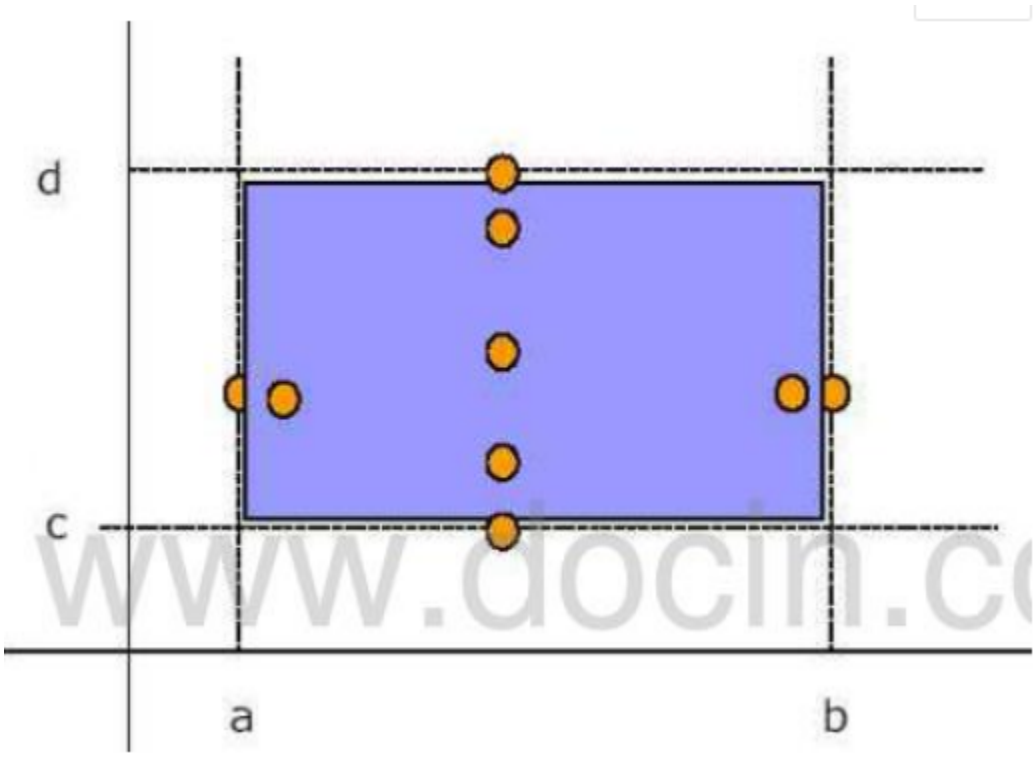


NextDate函数

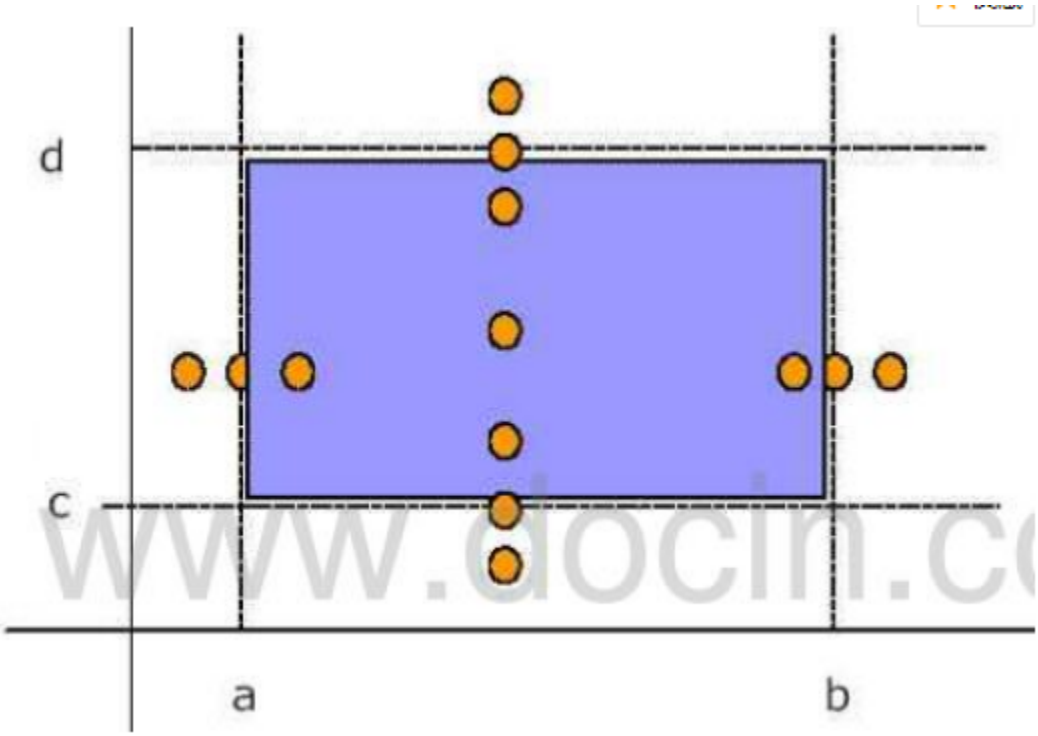


7、黑盒测试用例数的总结

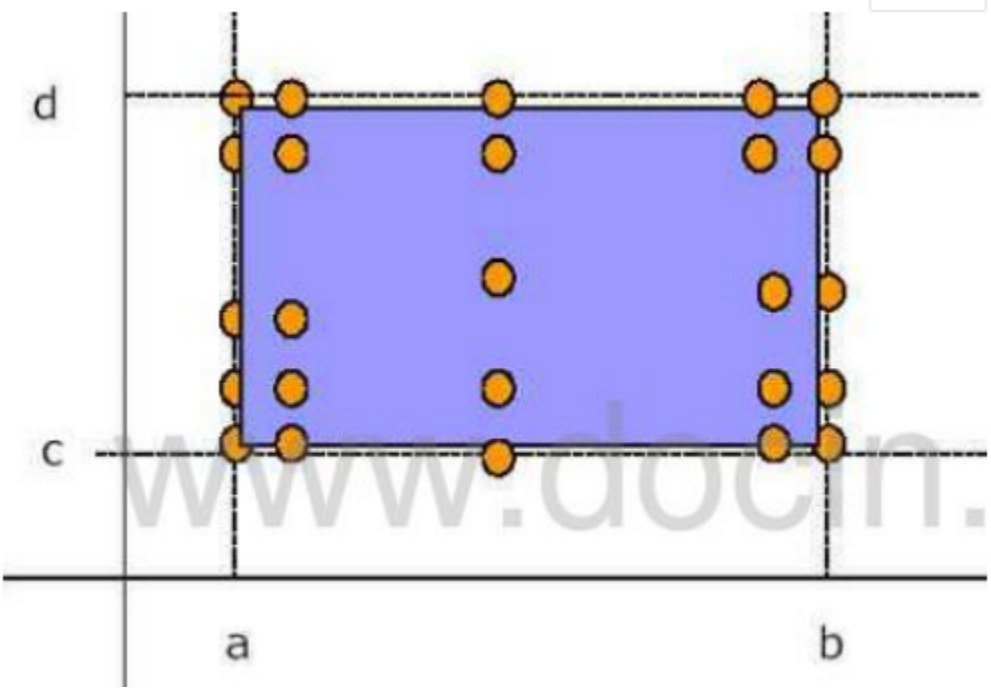
1、在边界测试中，对于有n个输入变量的程序，基本边界值分析的测试用例个数为4n+1。



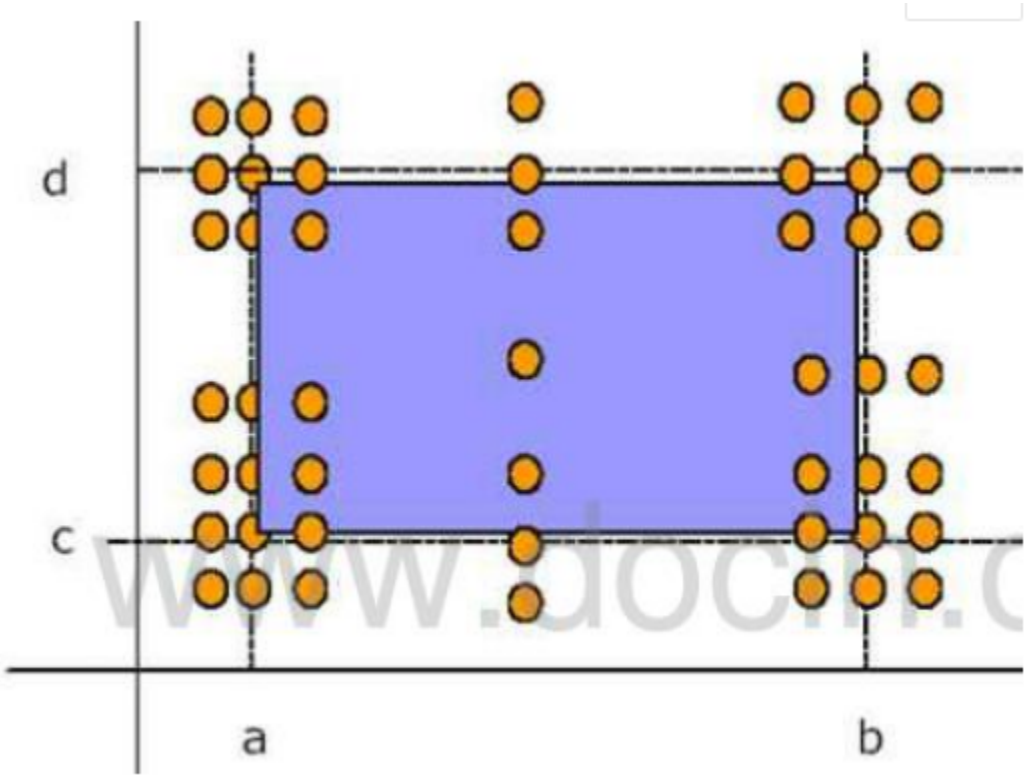
2、在健壮性测试中，对于有n个输入变量的程序，健壮性测试的测试用例个数为6n+1。



3、对于有n个输入变量的程序，最坏情况测试的测试用例个数为5^n。



4、对于有n个输入变量的程序，健壮最坏情况测试的测试用例个数为7^n。



8、基于组合技术和组合优化的方法（判定表/决策表方法 、因果图法）

判定表/决策表方法

一个判定表由“条件和活动”两部分组成，也就是列出了一个测试活动执行所需的条件组合，所有可能的条件组合定义了一系列的选择，而测试活动需要考虑每一个选择。

判定表元素：

1. 条件桩：列出问题的所有条件
2. 动作桩：列出可能针对问题所采取的操作
3. 条件项：针对所列条件的具体赋值
4. 动作项：列出在条件项（各种取值）组合情况下应该采取的动作。
5. 规则：任何一个条件组合的特定取值及其相应要执行的操作。

判定表方法步骤：

1. 列出所有的条件桩和动作桩；
2. 填入条件项；
3. 填入动作项，制定初始判定表；
4. 简化、合并相似规则或者相同动作

问题说明：“某货运站收费标准如下：如果收件地点在本省，则快件每公斤5元，慢件每公斤3元；如果收件地点在外省，则在20公斤以内（含20公斤）快件每公斤7元，慢件每公斤5元，而超过20公斤时，快件每公斤9元，慢件每公斤7元。”

		1	2	3	4	5	6	
条件桩	收件地址在本省？	Y	Y	N	N	N	N	条件项
	邮件重量<20公斤？	–	–	Y	Y	N	N	
	快慢件？	Y	N	Y	N	Y	N	
动作桩	3元/公斤		X					动作项
	5元/公斤	X			X			
	7元/公斤			X			X	
	9元/公斤					X		

因果图法

多种输入条件的组合，产生多种结果设计测试用例。

设计方法：

分析软件规格说明文档描述的哪些是原因（输入条件），哪些是结果（输出条件），给每个原因和结果赋予一个标识符

找出原因与结果，原因与原因之间的对应关系，划出因果图

在因果图上标上哪些不可能发生的因果关系，表明约束或限制条件

根据因果图，创建判定表，将复杂的逻辑关系和多种条件组合很具体明确的表示出来

把判定表的每一列作为依据设计测试用例。

因果图法总结:

优点 :

1、因果图法能够帮助我们按照一定步骤，高效的选择测试用例，设计多个输入条件组合用例

2、因果图分析还能为我们指出，软件规格说明描述中存在的问题

3、可以依据因果图检验需求的逻辑和程序未来应包含的函数或方法。

缺点 :

1、输入条件与输出结果的因果关系，有时难以从软件需求规格说明书得到。

2、即时得到了这些因果关系，也会因为因果关系复杂导致因果图非常庞大，测试用例数目极其庞大。

测试分类：白盒测试

白盒测试主要是检查程序的内部结构、逻辑、循环和路径。常用测试用例设计方法有：

1、逻辑覆盖：以程序的内部逻辑结构为基础，分为语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖等。

2、基本路径测试：在程序控制流程的基础上，分析控制构造的**环路复杂性**，导出基本可执行路径集合，从而设计测试用例。

逻辑覆盖 vs. 路径覆盖：

1、逻辑覆盖：以程序或系统的内部逻辑结构为基础，分为语句覆盖、判定覆盖、判定-条件覆盖、条件组合覆盖等。

2、基本路径测试：在程序或业务控制流程的基础上，分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例。

白盒测试基本原则(68页)

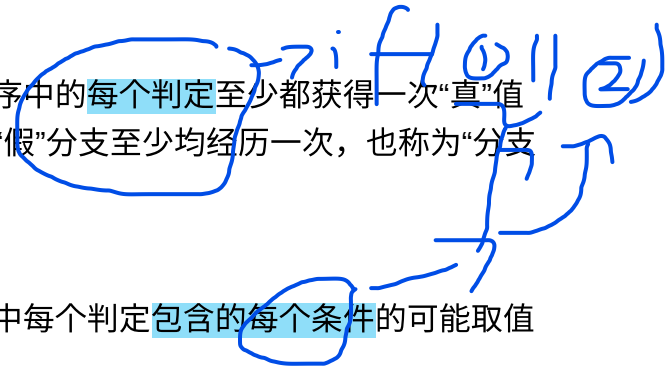
1. 所有独立路径至少测试一次
2. 所有逻辑值真假两种情况
3. 检查内部数据结构，保证结构有效性
4. 在上下边界及可操作范围内运行所有循环

1、逻辑覆盖测试

根据覆盖目标的不同，逻辑覆盖又可分为语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、组合覆盖和路径覆盖。

1、**语句覆盖**：选择足够多的测试用例，使得程序中的每个可执行语句至少执行一次。

2、**判定覆盖**：通过执行足够的测试用例，使得程序中的每个判定至少都获得一次“真”值和“假”值， 也就是使程序中的每个取“真”分支和取“假”分支至少均经历一次，也称为“分支覆盖”。



3、**条件覆盖**：设计足够多的测试用例，使得程序中每个判定包含的每个条件的可能取值（真/假）都至少满足一次。

4、**判定/条件覆盖**：设计足够多的测试用例，使得程序中每个判定包含的每个条件的所有情况（真/假）至少出现一次，并且每个判定本身的判定结果（真/假）也至少出现一次。

——满足判定/条件覆盖的测试用例一定同时满足判定覆盖和条件覆盖。

5、**组合覆盖**：通过执行足够的测试用例，使得程序中每个判定的所有可能的条件取值组合都至少出现一次。

——满足组合覆盖的测试用例一定满足判定覆盖、条件覆盖和判定/条件覆盖。

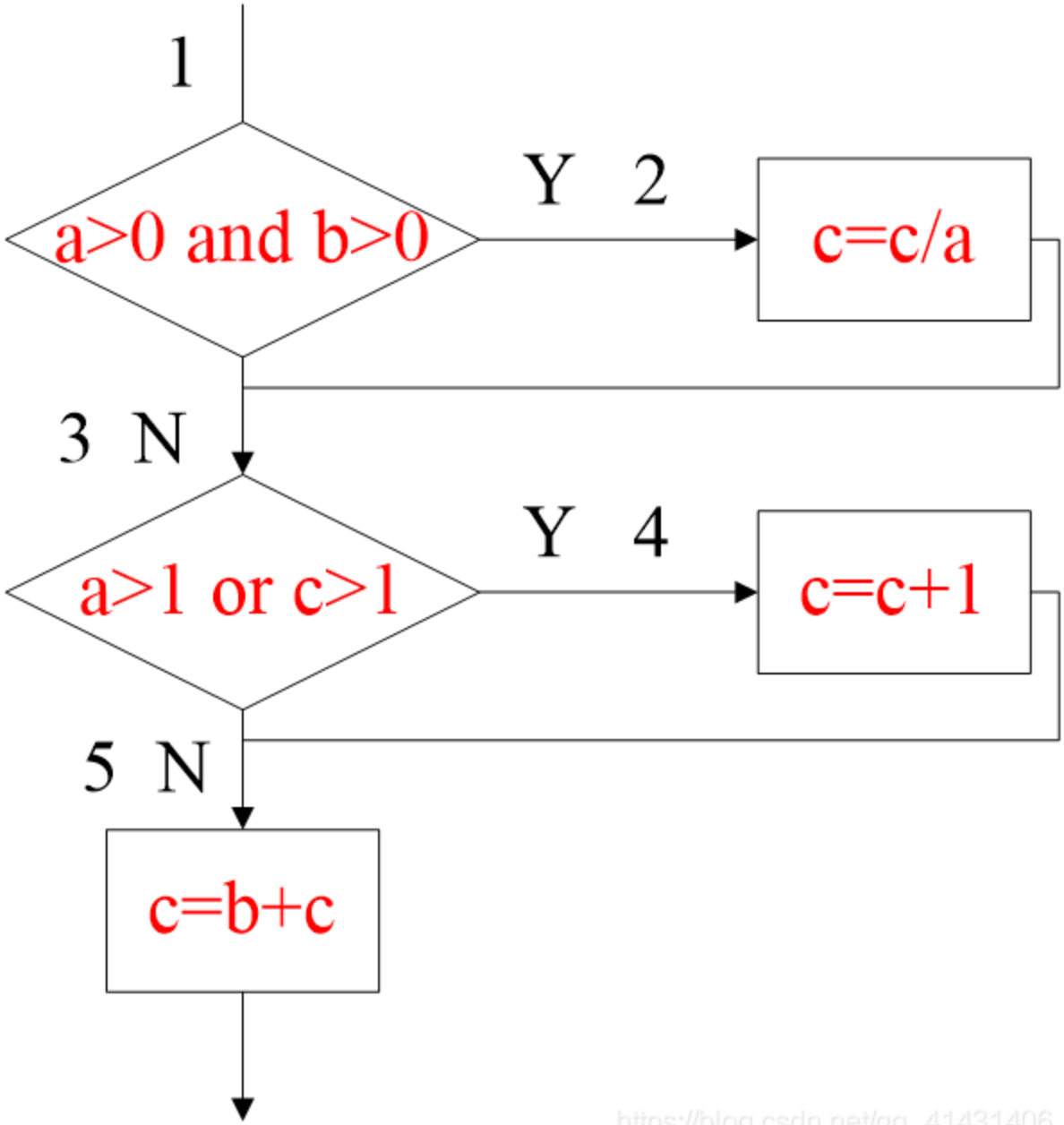
6、**路径覆盖**：设计足够多的测试用例，要求覆盖程序中所有可能的路径。

逻辑覆盖详解

1、**语句覆盖：使程序中的每个可执行语句至少被执行一次**

如果是顺序结构，就是让测试从头执行到尾

如果有分支、条件和循环，需要利用下面的方法，执行足够的测试覆盖全部语句



只需设计一个测试用例:　a=2，b=1，c=6；即达到了语句覆盖。

【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，但对于隐藏的条件是无法测试的。如在多分支的逻辑运算中无法全面的考虑。**语句覆盖是最弱的逻辑覆盖。**

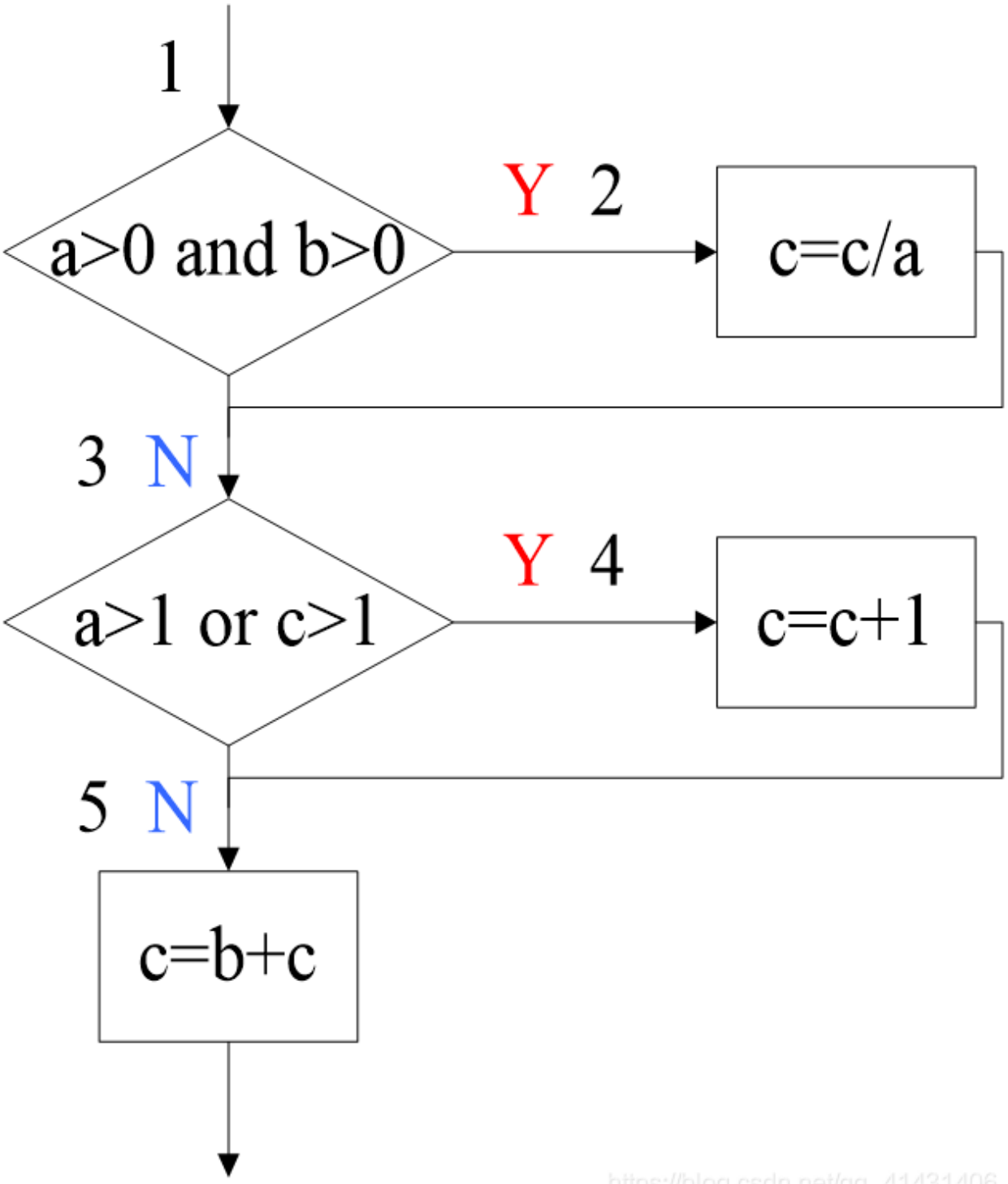
2、**判定覆盖：每个判断的取真分支和取假分支至少经历一次，即判断真假值均曾被满足。 一个判定往往代表着程序的一个分支，所以判定覆盖也被称为分支覆盖。**

a=2，b=1 ，c=6可覆盖判断M的Y分支和判断N的Y分支；

a=-2，b=-1 ，c=-3可覆盖判断M的N分支和判断N的N分支 。 这两组测试用例可覆盖所有判定的真假分支。

a=1，b=1 ，c=-3 可覆盖判断M的Y分支和判断N的N分支 ；

a=1，b=-2 ，c=3可覆盖判断M的N分支和判断N的Y分支 ； 同样的这两组测试用例也可覆盖所有判定的真假分支。



【优点】：判定覆盖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**，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。判定覆盖仍是弱的逻辑覆盖。

3、**条件覆盖：要使每个判断中每个条件的可能取值至少满足一次。**

判断M表达式:

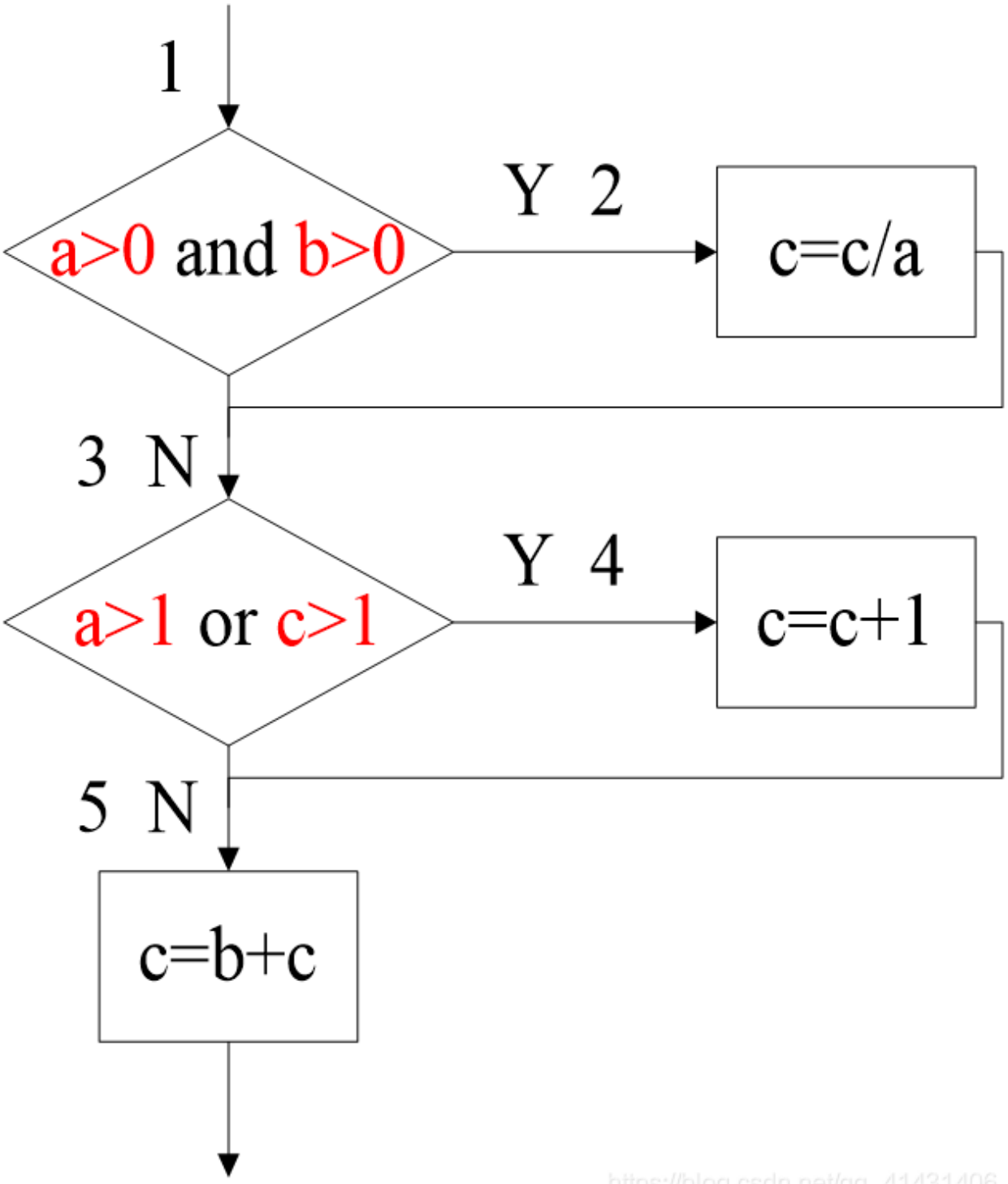
设条件 a>0 取真 记为 T1　　取假 记为 F1

设条件 b>0 取真 记为 T2　　取假 记为 F2

判断N表达式:

设条件 a>1 取真 记为 T3　　取假 记为 F3

设条件 c>1 取真 记为 T4　　取假 记为 F4

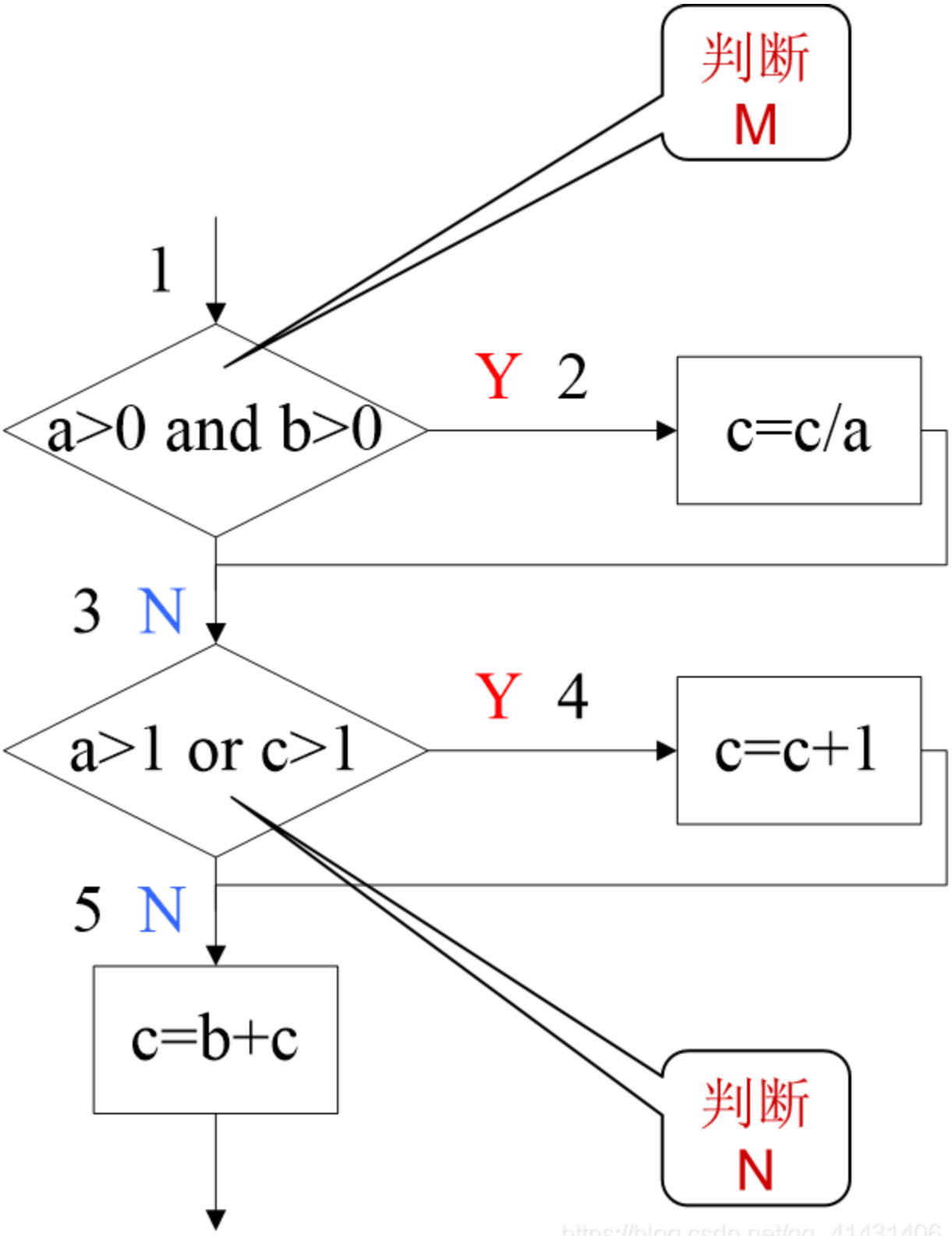


它覆盖了判定M的N分支和判断N的Y分支。

测试用例	覆盖条件	具体取值条件
a=2,b=-1,c=-2	T1, F2, T3, F4	a>0,b<=0, a>1,c<=1
a=-1,b=2,c=3	F1, T2, F3, T4	a<=0,b>0, a<=1,c>1

我们用条件覆盖设计的思想就是让测试用例能覆盖 T1、T2、T3、T4、F1、F2、F3、F4

4、**判定条件覆盖：判定和条件覆盖设计方法的交集，使得判断条件中的所有条件可能取值至少执行一次，同时，所有判断的可能结果至少执行一次。**



按照判定－条件覆盖的要求，我们设计的测试用例要**满足如下条件**：

1.**所有条件可能**至少执行一次取值；

2.**所有判断的可能**结果至少执行一次。

测试用例	取值条件	具体取值条件	判定条件	通过路径
输入：a=2，b=1，c=6 输出：a=2，b=1，c=5	T1，T2，T3，T4	a>0，b>0, a>1，c>1	M=.T. N=.T.	P1（1-2-4）
输入：a=-1，b=-2，c=-3 输出：a=-1，b=-2，c=-5	F1，F2，F3，F4	a<=0, b<=0, a<=1, c<=1	M=.F. N=.F.	P4（1-3-5）

测试用例	覆盖条件	覆盖判断
a=2,b=1,c=6	T1, T2, T3, T4	M的Y分支和N的Y分支
a=-1,b=-2,c=-3	F1, F2, F3, F4	M的N分支和N的N分支

【优点】 ：能同时满足判定、条件两种覆盖标准。

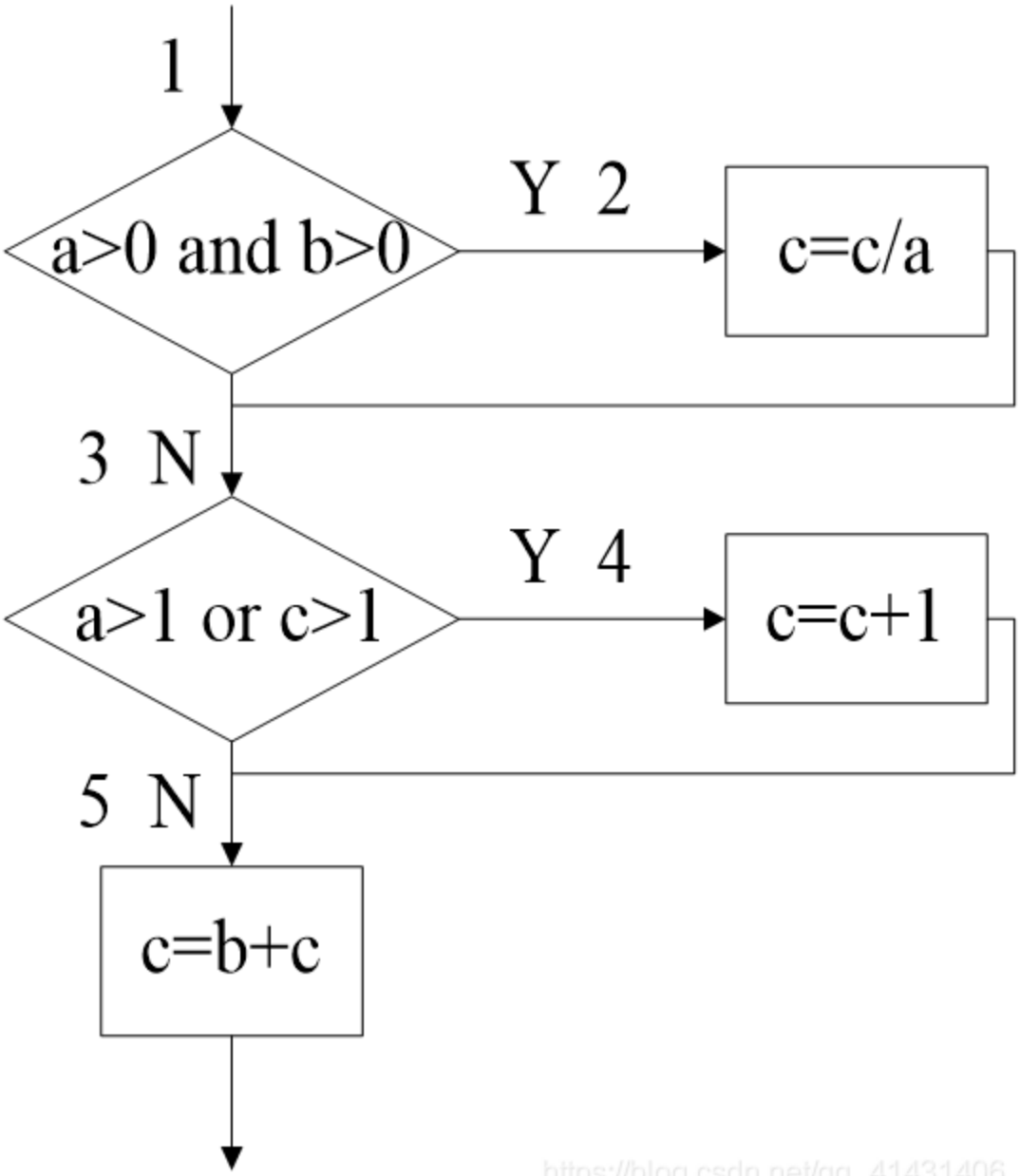
【缺点】 ：判定/条件覆盖准则的缺点是未考虑条件的组合情况。

5、**条件组合覆盖：每个判断的所有可能的条件取值组合都至少出现一次**

它与条件覆盖的差别是它不是简单地要求每个条件都出现“真”与“假”两种结果，而是要求让这些结果的所有可能组合都至少出现一次。

按照条件组合覆盖的基本思想，对于前面的例子，我们把每个判断中的所有条件进行组合，设计组合条件如表所示，而我们设计的测试用例就要包括所有的组合条件。

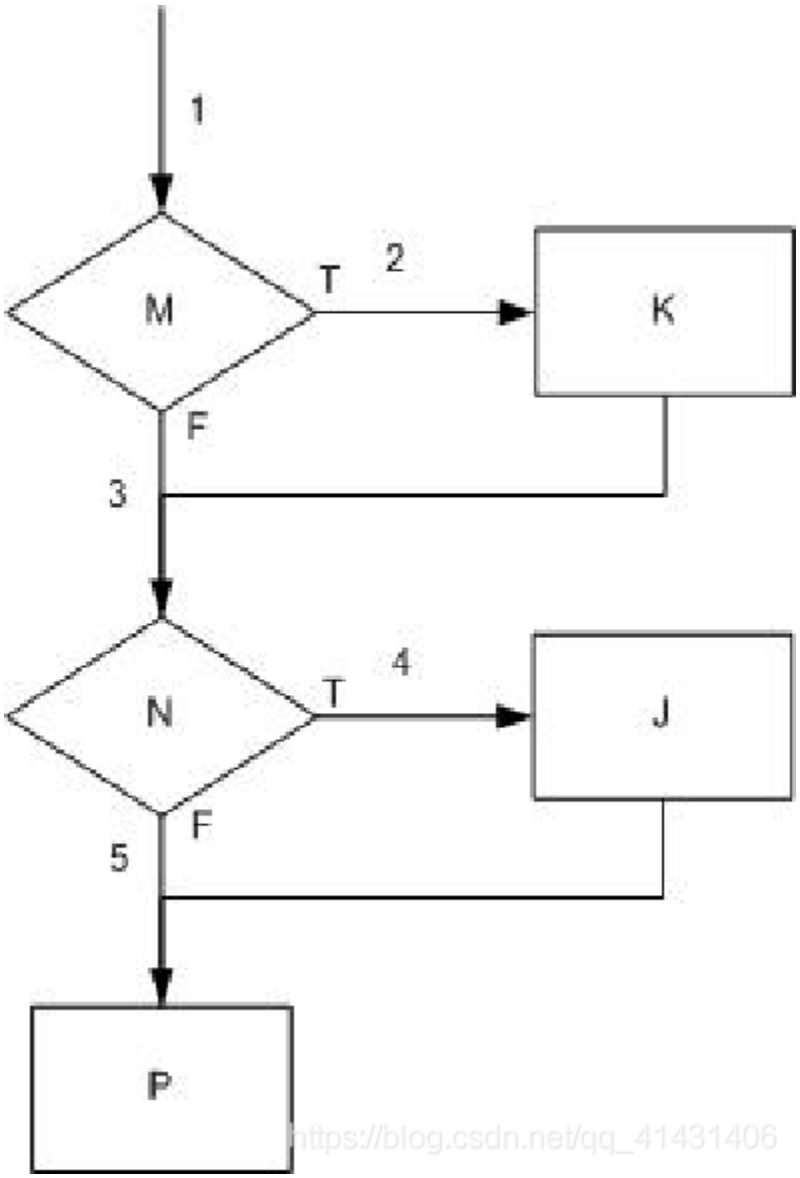
组合编号	覆盖条件取值	判定条件取值	判定-条件组合
1	T1，T2	M=.T.	a>0，b>0，M取真
2	T1，F2	M=.F.	a>0，b<=0，M取假
3	F1，T2	M=.F.	a<=0，b>0，M取假
4	F1，F2	M=.F.	a<=0，b<=0，M取假
5	T3，T4	N=.T.	a>1，c>1，N取真
6	T3，F4	N=.T.	a>1，c<=1，N取真
7	F3，T4	N=.T.	a<=1，c>1，N取真
8	F3，F4	N=.F.	a<=1，c<=1，N取假



测试用例	覆盖条件	覆盖判断	覆盖组合
a=2,b=1,c=6	T1, T2, T3, T4	M取Y分支，Q取Y分支	1，5
a=2,b= -1,c= -2	T1, F2, T3, F4	M取N分支，Q取Y分支	2，6
a=-1,b=2,c=3	F1, T2, F3, T4	M取N分支，Q取Y分支	3，7
a= -1,b= -2,c= -3	F1, F2, F3, F4	M取N分支，Q取N分支	4，8

要满足1、2、3、4、5、6、7、8条件组合

测试用例	覆盖条件	覆盖路径	覆盖组合
输入：a=2，b=1，c=6 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 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 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 输出：a=-1，b=-2，c=-5	F1，F2，F3，F4	P4（1-3-5）	4，8



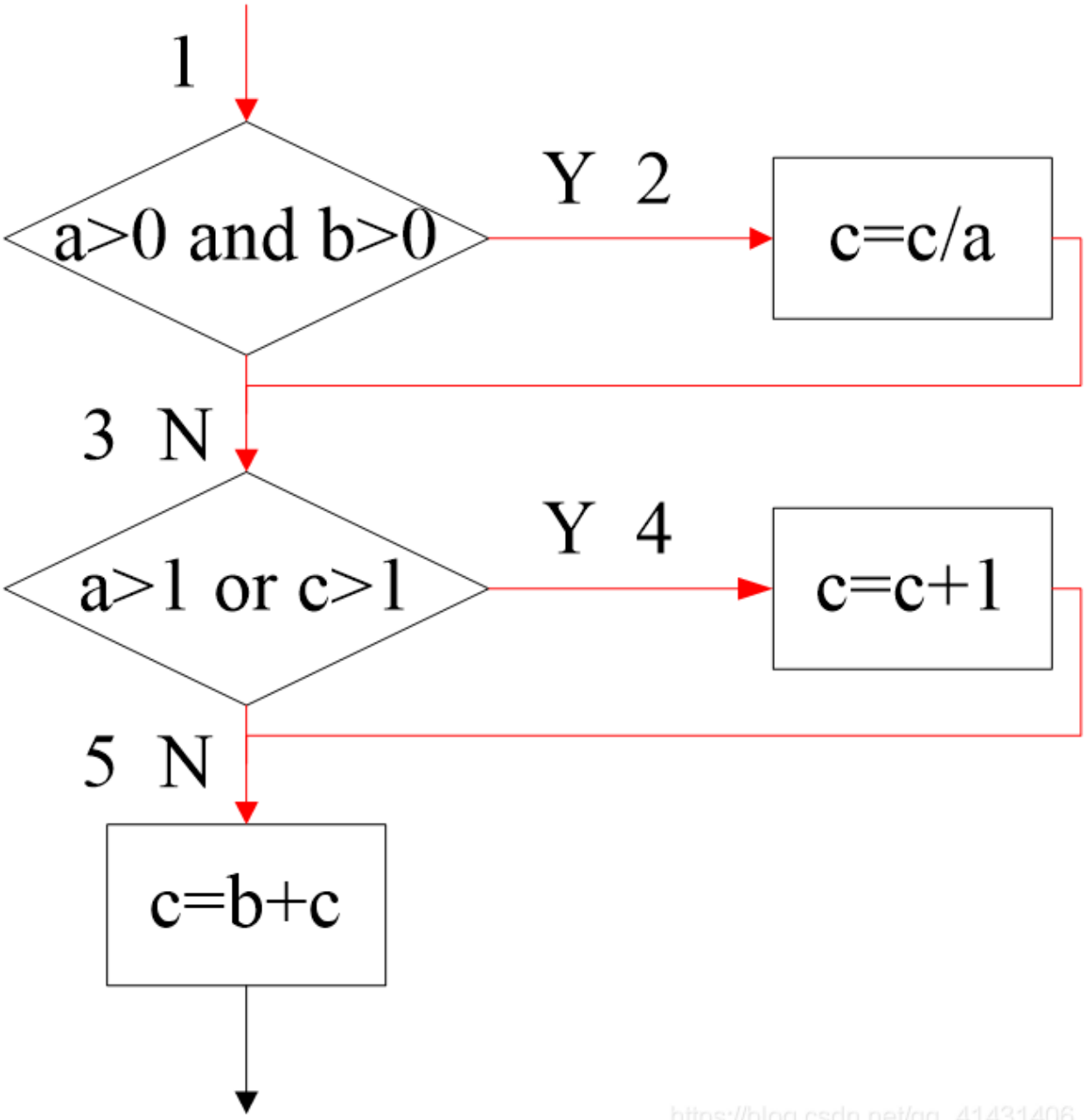
覆盖了所有组合，但覆盖路径有限，1-2-5 没被覆盖

【优点】：条件组合覆盖准则满足判定覆盖、条件覆盖和判定/条件覆盖准则。

【缺点】：线性地增加了测试用例的数量。

6、**路径覆盖：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。**

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



测试用例	覆盖组合	覆盖路径
a=2,b=1,c=6	1，5	1-2-4
a=1,b=1,c=-3	1，8	1-2-5
a=-1,b=2,c=3	4，7	1-3-4
a=-1,b=-2,c=-3	4，8	1-3-5

【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。

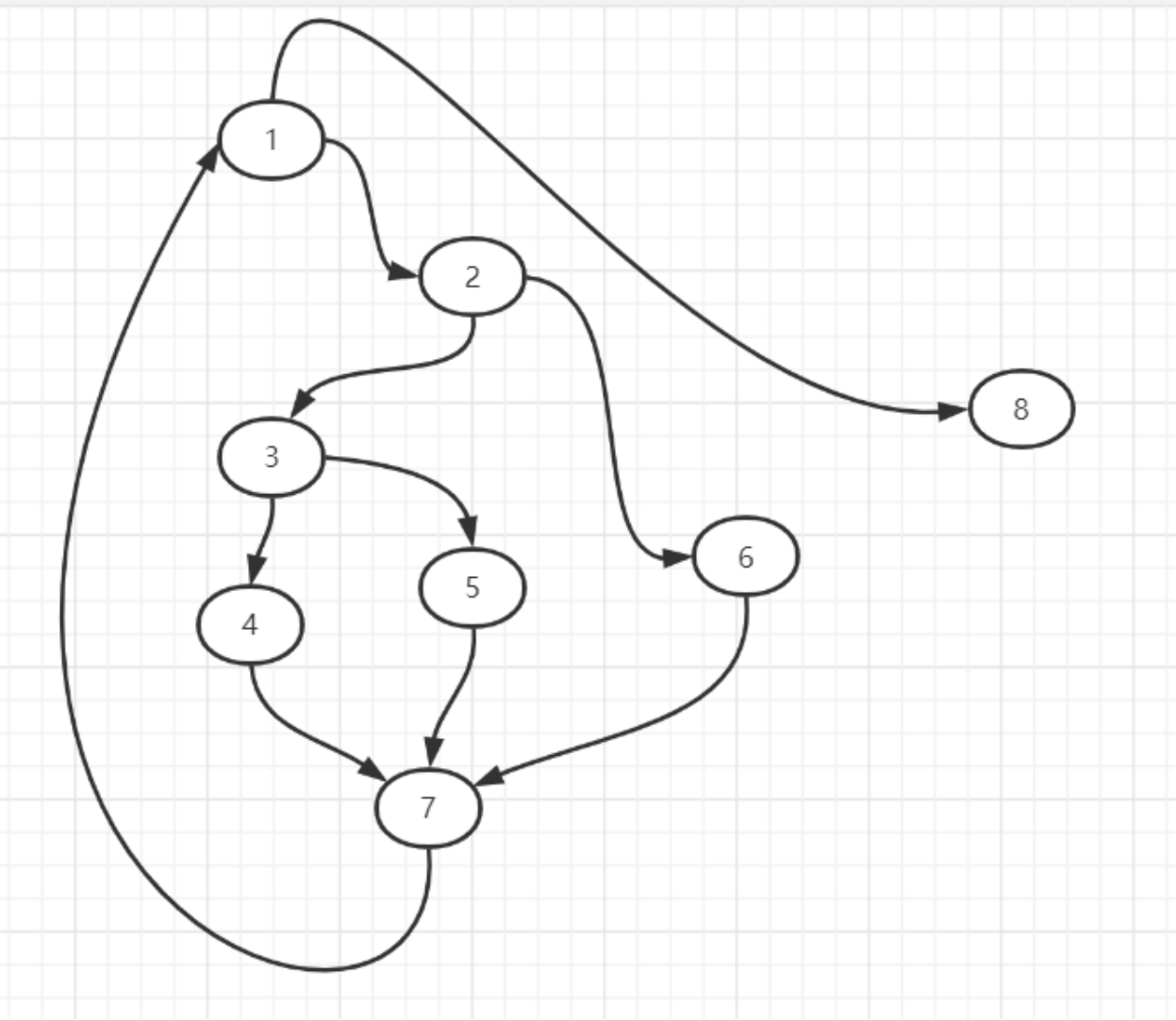
【缺点】 ：需要设计大量、复杂的测试用例，使得**工作量呈指数级增长，不见得把所有的条件组合都覆盖。**

2、基本路径测试

基本路径测试并不是测试所有路径的组合，仅仅保证每条基本路径被执行一次！

画程序流程图，控制流图，计算环形复杂度

做出程序的控制流图：



测试源程序：

```
public class Testing {
    int Test(int i_count, int i_flag) {
        int i_temp = 0;
        while(i_count>0) {
            if(0==i_flag) {
                i_temp = i_count + 100;
                break;
            }
            else {
                if(1==i_flag) {
                    i_temp = i_temp + 10;
                }
                else {
```

```
                    i_temp = i_temp + 20;
                }
            }
            i_count--;
        }
        return i_temp;
    }
}
```

测试代码：

```
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.runner.JUnitCore;

//基本路径测试，每条可能的路径都执行一次
public class Testteing {

    Testing t = new Testing();

    @Test
    public void testTest() {

        //i_count<=0的情况（取<）
        assertEquals(0, t.Test(-1, 1));
        //i_count<=0的情况（取=）
        assertEquals(0, t.Test(0, 0));
        //i_count>0 且 0==i_flag
        assertEquals(110, t.Test(10, 0));
        //i_count>0 且 0!=i_flag 且 1!=i_flag
        assertEquals(400, t.Test(20, 20));
        //i_count>0 且 1==i_flag 的情况
        assertEquals(300, t.Test(30, 1));
    }
}
```

（三）selenium3自动化测试

控制浏览器后退、前进：

```
from selenium import webdriver

# 设置浏览器驱动
driver = webdriver.Chrome()

#访问百度首页
first_url = "http://www.baidu.com"
print("now %s" %(first_url))
driver.get(first_url)

#访问新闻页面
second_url = 'http://news.baidu.com'
print("now %s" %(second_url))
driver.get(second_url)

print("back to %s" %(first_url))
driver.back()

#前进到新闻页
print("forward to %s" %second_url)
driver.forward()
```

WebDriver常用方法：

```
from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException

driver = webdriver.Chrome()
driver.get("https://passport.bilibili.com/login?from_spm_id=333.1007.top_bar.login")

driver.find_element_by_id("login-username").send_keys("******")
driver.find_element_by_id("login-passwd").send_keys("******")
driver.find_element_by_id("geetest-wrap").click()
```