NJUSWI
2002-2012
Anniversary
南京大學 软件学院

测试分支和循环

刘 钦

南京大学软件学院

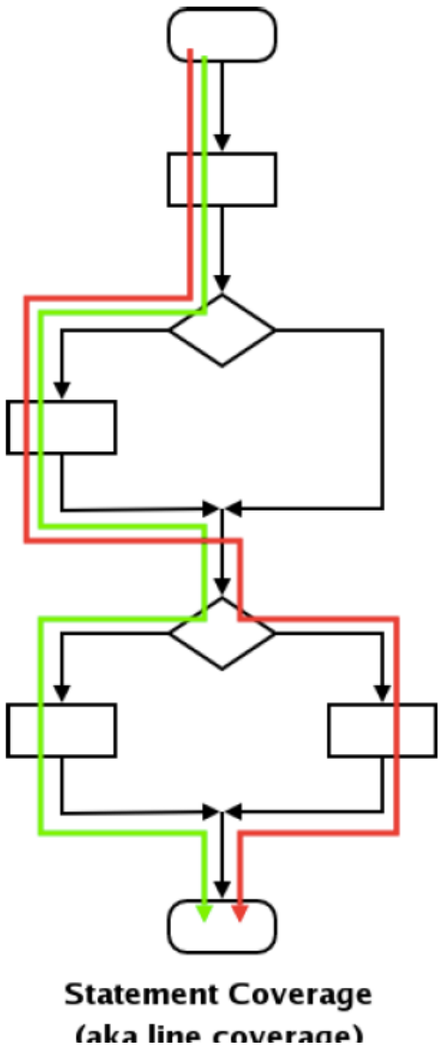
有代码就要有测试！

白盒测试

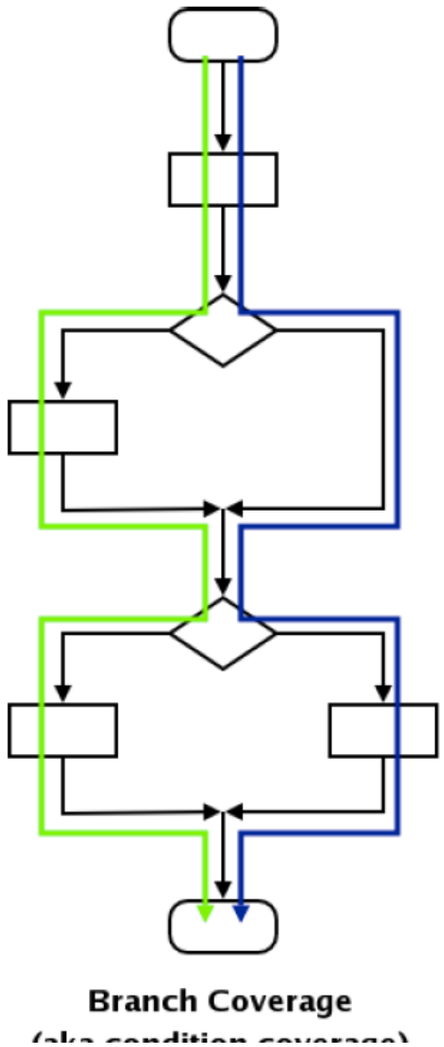
白盒测试

- 测试应用程序的内部结构或运作，而不是测试应用程序的功能
- 测试者输入数据验证数据流在程序中的流动路径，并确定适当的输出

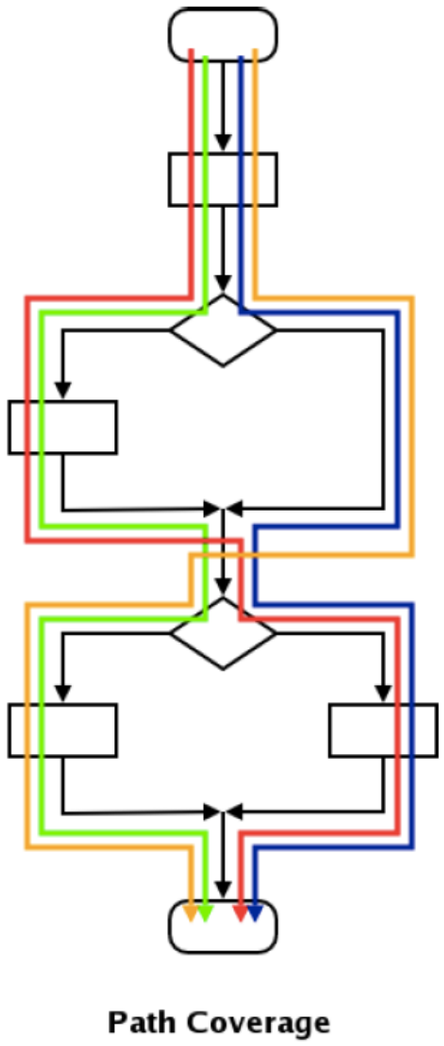
分支如何测试



Statement Coverage
(aka line coverage)



Branch Coverage



Path Coverage

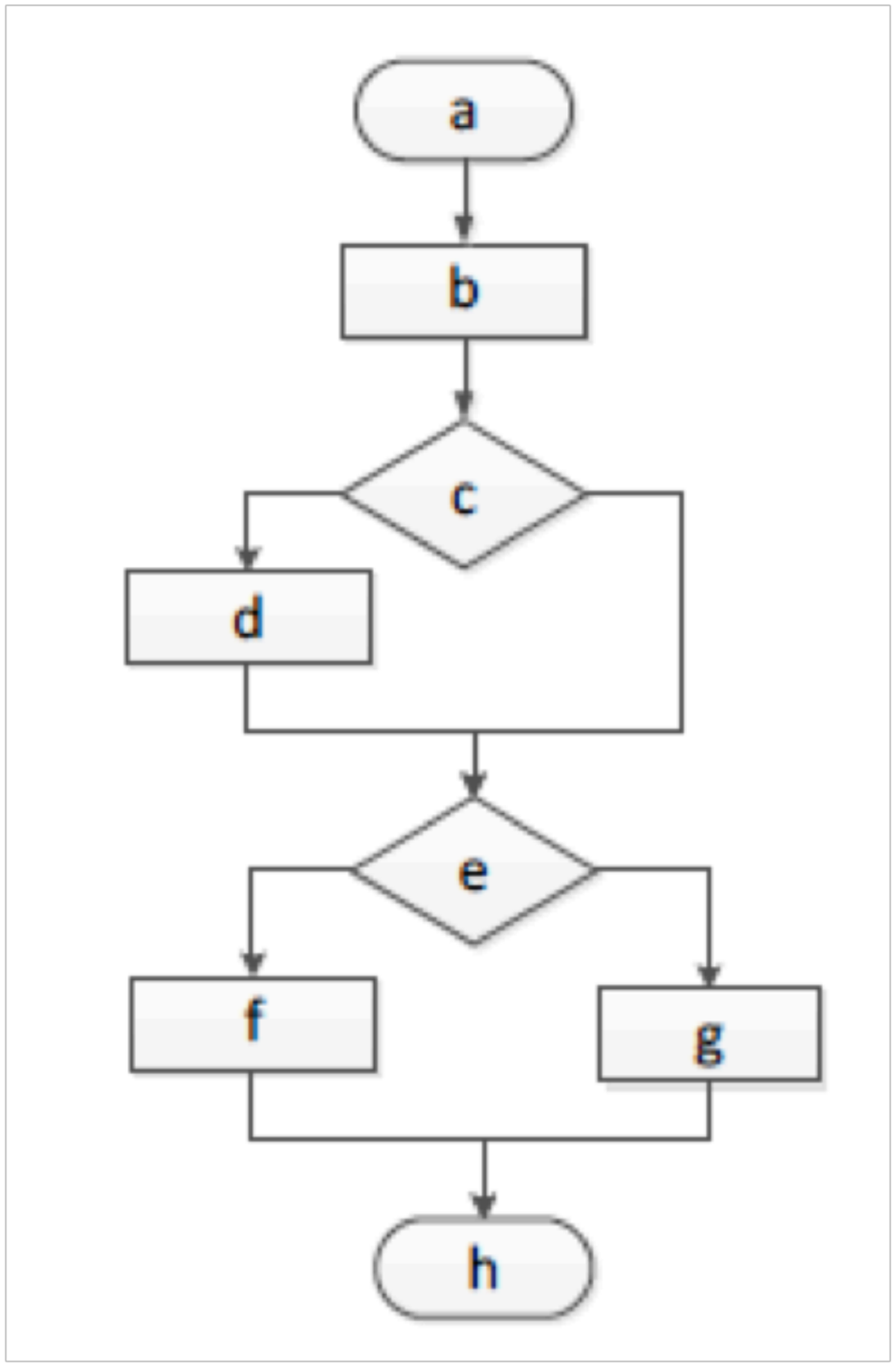
语句覆盖　分支覆盖　路径覆盖

白盒测试

白盒测试方法 - 语句覆盖

语句覆盖设计测试用例的标准是确保被测试对象的每一行程序代码都至少执行一次。

```
public Class Customer {
    int bonus;//积分额度
    ……
    //预计算消费行为后的积分额
    int getBonus ( boolean cashPayment , int consumption , bo
        //已有的 bonus 需要调整 getBonus，不能直接使用
        int preBonus=this.getBonus();
        if (cashPayment) {
            preBonus+=consumption;
        }
        if (vip) {
            preBonus*=1.5;
        } else {
            preBonus*=1.2;
        }
        return preBonus;
}
```



为了清晰地解释对语句的覆盖，图19-10为各条语句都进行了编号。可以发现如果按照

下列两条路径执行，就能够覆盖所有的语句：**a,b,c,d,e,f,h** 和 **a,b,c,d,e,g,h**

表 19-6 Customer.getBonus()的语句覆盖测试用例

ID	输入				预期输出
1	getBonus()=100	cashPayment=true	consumption=100	vip=true	300
2	getBonus()=100	cashPayment=true	consumption=100	vip=false	240

白盒测试方法
- 条件覆盖

条件覆盖设计测试用例的标准是确保程序中每个判断的每个结果都至少满足一次

表 19-7 Customer.getBonus()的条件覆盖测试用例

ID	输入				预期输出
1	getBonus()=100	cashPayment=true	consumption=100	vip=true	300
2	getBonus()=100	cashPayment=false	consumption=100	vip=false	120

白盒测试方法
- 路径覆盖

路径覆盖测试用例的标准是确保程序中每条独立执行路径都至少执行一次

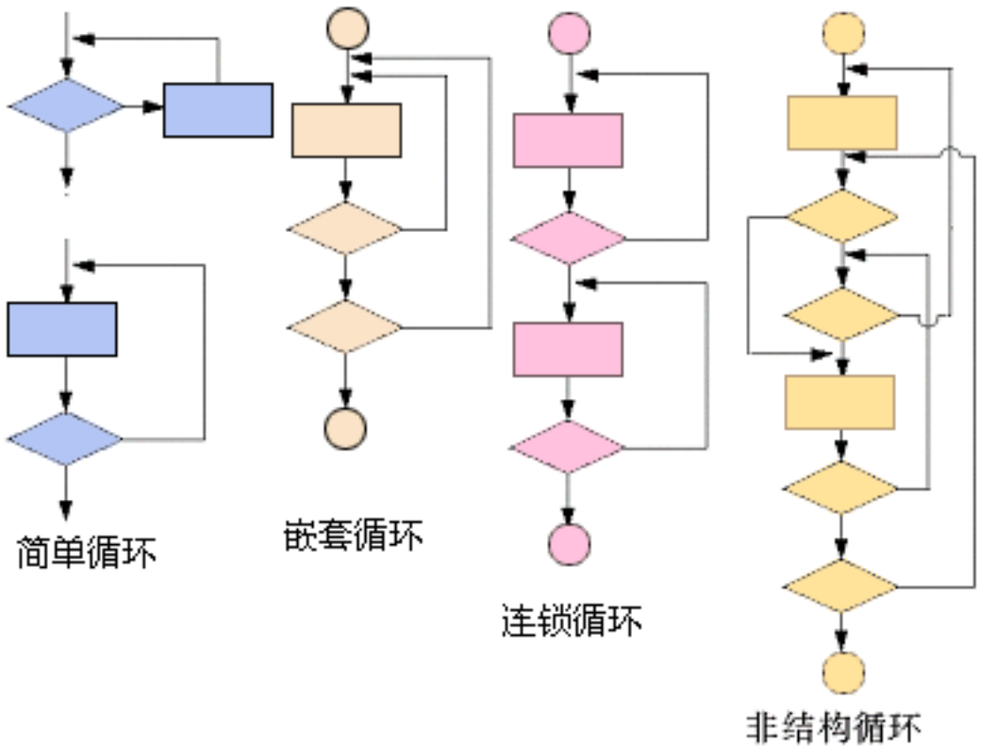
表 19-8 Customer.getBonus()的路径覆盖测试用例

ID	输入				预期输出
1	getBonus()=100	cashPayment=true	consumption=100	vip=true	300
2	getBonus()=100	cashPayment=false	consumption=100	vip=false	120
	getBonus()=100	cashPayment=true	consumption=100	vip=false	240
	getBonus()=100	cashPayment=false	consumption=100	vip=true	150

循环如何测试？

循环的分类

- 简单循环
- 嵌套循环
- 连锁循环
- 不规则循环



简单循环

- 下列测试集用于简单循环，其中n表示允许通过循环的最大次数。
  - 整个跳过循环。
  - 只有一次通过循环。
  - 两次通过循环。
  - m次通过循环，其中m<n。
  - n-l次通过循环。
  - n次通过循环。

案例

- void function(int number)　　// number 是大于0，小于100的数。
- {
- 　if (number % 2) == 0)
- 　　printf("even \n");
- 　for (;number < 9; number++){
- 　　printf("number is %d\n", number);
- 　}
- }
- 测试用例：
- function(11);　//整个跳过循环。
- function(8);　//只有一次通过循环。
- function(7);　//两次通过循环。
- function(6);　//m次通过循环，其中m<n。
- function(1);　//n-1次通过循环
- function(0); n次通过循环。

嵌套循环

- 除最内层的循环外，所有其他层的循环变量置为最小值，对最内层的循环进行简单循环的全部测试；
- 逐步外推，对其外面一层循环进行测试。测试时，所有外层循环的循环变量取最小值，所有其他嵌套内层循环的循环变量取"典型"值；
- 反复进行，直到所有各层循环测试完毕；
- 对全部各层循环同时取最小循环次数，或者最大循环次数。

连锁循环

- 对于连锁循环，如果各个循环是相互独立的，则可以用与简单循环相同的方法进行测试；
- 但是如果两个循环串接起来，而第一个循环是第二个循环的初始值，则这两个循环并不是独立的，则推荐使用嵌套循环的方法进行测试。

不规则循环

- 不规则循环不能进行测试，尽量重新设计结构化的程序后再进行测试。