第七章　实现

7.1　编码

7.2　软件测试基础

7.6　白盒测试技术

7.7　黑盒测试技术

7.3　单元测试

7.4　集成测试

7.5　确认测试

7.8　调试

实现

- **编码**：**编码**就是把软件设计结果翻译成用某种程序设计语言书写的程序，是对设计的进一步具体化。
- **测试**：**程序的质量主要取决于软件设计的质量**。软件**测试**是保证软件质量的关键步骤，是对软件规格说明、设计和编码的最后复审。

一、选择程序设计语言

- 程序设计语言是人和计算机通信最基本的工具，会影响人的思维和解题方式，影响人机通信的方式和质量，影响其他人阅读和理解程序的难易程度。
- 选择适宜的程序设计语言的原因：
  - 根据设计去完成编码时，困难最少；
  - 可以减少需要的程序测试量；
  - 可以得到更容易阅读和更容易维护的程序。

7.1 编码

- 理想标准：
  - 理想的模块化机制，可读性好的控制结构和数据结构；
  - 使编译程序能够尽可能多地发现程序中的错误；
- 实用标准：
  - 系统用户的要求；
  - 可以使用的编译程序和可以得到的软件工具；
  - 工程规模；
  - 程序员的知识；
  - 软件可移植性要求；
  - 软件的应用领域。

二、编码风格

代码应逻辑简明清晰、易读易懂。应该遵循下述规则。

1、程序内部的文档

- **标识符：**含义鲜明的名字、缩写规则一致、为名字加注解
- **注解：****模块开始处**简要描述模块的功能、主要算法、接口特点、重要数据以及开发简史或解释包含这段代码的必要性。**代码中间**的注解可解释这段代码的功能。
- **视觉组织：**适当的阶梯形式使程序的层次结构清晰明显。（缩进应固定距离，建议4个空格）

2、数据说明

- 数据说明的次序应该标准化

  ✓如先说明简单类型，再说明复杂类型等；

  ✓对简单类型的变量进行说明时，可先说明整型，再说明实型，再说明字符型等等。

- 当多个变量名在一个语句中说明时，应该按字母顺序排列这些变量；（不建议多个变量定义在一行上）
- 如果设计时使用了一个复杂的数据结构，则应该用注解说明实现这个数据结构的方法和特点。

3、语句构造

- 不要为了节省空间而把多个语句写在同一行；
- 不要书写太复杂的条件，嵌套的层数也不宜过多；
- 由于人的一般思维方式对逻辑非运算不太适应，因此在条件表达式中应尽量不使用否定的逻辑表示；
- 利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

哪一种风格更好？

```
if (condition)   DoSomething();
else      DoSomethingElse();
```

```
if (condition)
        DoSomething();
else
        DoSomethingElse();
```

```
if (condition) {
        DoSomething();
}else  {
        DoSomethingElse();
}
```

```
if (condition)
{
        DoSomething();
 }
 else
{
        DoSomethingElse();
}
```

4、输入输出

- 对所有输入数据都进行合法性检验；
- 保持输入格式简单；
- 使用数据结束标记，不要要求用户指定数据的数目；
- 明确提示交互式输入的请求，详细说明可用的选择或边界数值；
- 程序设计语言对格式有严格要求时，应保持输入格式一致；
- 设计良好的输出报表；
- 输出数据时要加上必要的提示信息。

5.效率

- 效率主要指处理机时间和存储器容量两方面
- 效率是性能要求，因此应该在需求分析阶段确定效率方面的要求；
- 效率是靠好设计来提高的；
- 程序的效率和程序的简单程度是一致的，**不要牺牲程序的清晰性和可读性来不必要地提高效率**。

一、软件测试的目标

- G.Myers给出的关于测试的一些规则如下：
  - 测试是为了发现程序中的错误而执行程序的过程。
  - 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案。
  - 成功的测试是发现了至今为止尚未发现的错误的测试

测试只能证明程序中错误的存在，不能证明程序是正确的！

二、软件测试准则

- 所有测试都应该能追溯到用户需求；
- 应该远在测试开始之前就制定出测试计划；
- 把Pareto原理应用到软件测试中；测试发现的错误中80%很可能是由程序中20%的模块造成的
- 应从“小规模”测试开始，逐步进行“大规模”测试；
- 穷举测试是不可能的；
- 为了达到最佳的测试效果，应该由独立的第三方从事测试工作。

三、测试方法

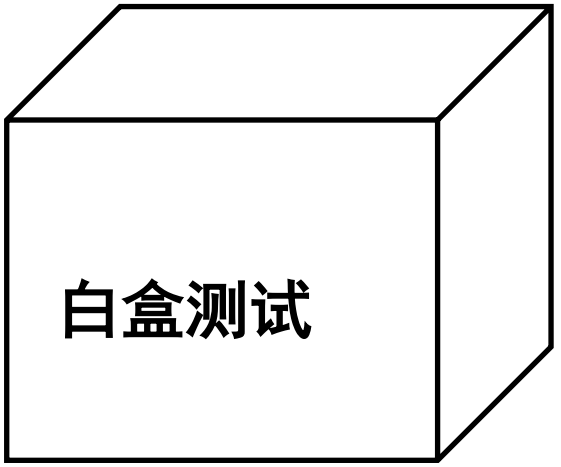
- **黑盒测试**（又称功能测试）把程序看作一个黑盒子，完全不考虑程序的内部结构和处理过程。

- 在程序接口进行的测试，只检查程序功能是否能按规格说明书的规定正常使用，程序是否能适当接收输入数据并产生正确的输出信息，程序运行过程中能否保持外部信息（例如数据库或文件）的完整性

三、测试方法

- **白盒测试**（又称结构测试）

  是把程序看成装在一个透明的白盒子里，测试者完全知道**程序的结构和处理算法**。



- 按照程序内部的逻辑测试程序，检测程序中的主要执行通路是否都能按预定要求正确工作。

四、测试步骤

- 根据第4条测试准则，测试过程也必须分步骤进行，后一个步骤在逻辑上是前一个步骤的继续。
- 大型软件系统通常由若干个子系统组成，每个子系统又由许多模块组成，因此，大型软件系统的测试过程基本上由**模块测试**、**子系统测试**、**系统测试**、**验收测试**和**平行运行**等五个步骤组成。

1.模块测试

- 设计得好的软件系统中，每个模块完成一个清晰定义的子功能，而且这个子功能和同级其他模块的功能之间没有相互依赖关系。因此，有可能把每个模块作为一个单独的实体来测试。
- 模块测试的目的是保证每个模块作为一个单元能正确运行，所以模块测试通常又称**单元测试**。在这个测试步骤中所发现的往往是编码和详细设计的错误

2.子系统测试

把经过单元测试的模块放在一起形成一个子系统来测试。模块相互间的协调和通信是这个测试过程中的主要问题，着重测试模块的接口。

3.系统测试

把经过测试的子系统装配成一个完整的系统来测试，验证系统确实能提供需求说明书中指定的功能，而且系统的动态特性也符合预定要求。在这个测试步骤中发现的往往是软件设计中的错误，也可能发现需求说明中的错误。

子系统测试和系统测试都兼有检测和组装两重含义，通常称为集成测试。

4.验收测试

- 把软件系统作为单一的实体进行测试，测试内容与系统测试基本类似，但是它是在**用户**积极参与下进行的，而且可能主要**使用实际数据**(系统将来要处理的信息)进行测试。
- 验收测试的目的是验证系统确实能够满足用户的需要，在这个测试步骤中发现的往往是系统需求说明书中的错误。验收测试也称为**确认测试**。

5.平行运行

- 同时运行新开发出来的系统和将被它取代的旧系统，以便比较两个系统的处理结果。其目的有如下几点。

(1)可在准生产环境中运行新系统而又不冒风险。

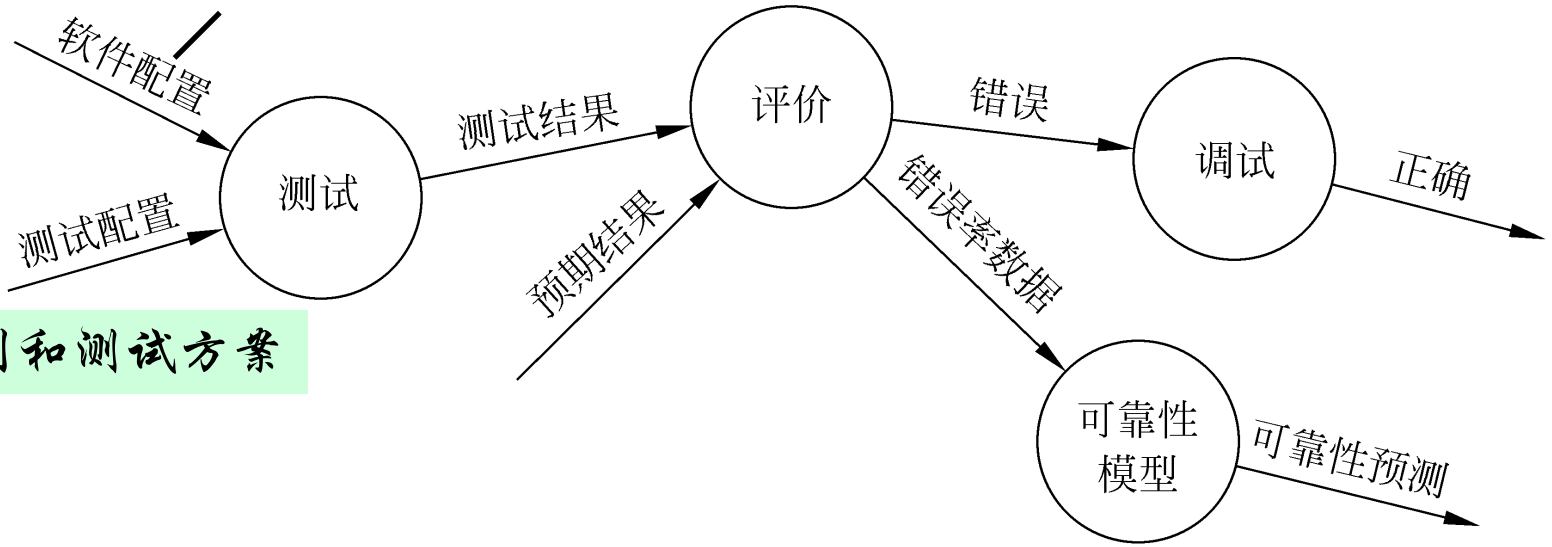
(2)用户能有一段熟悉新系统的时间。

(3)可以验证用户指南和使用手册之类的文档。

(4)能够以准生产模式对新系统进行全负荷测试，可以用测试结果验证性能指标。

五、测试阶段的信息流

需求说明书、设计说明书和源程序清单等



测试计划和测试方案

测试方案不仅仅是测试时使用的输入数据(称为测试用例)，还应该包括每组输入数据预定要检验的功能，以及每组输入数据预期应该得到的正确输出。

对测试结果进行收集和评价

- 如果经常出现要求修改设计的严重错误，那么软件的质量和可靠性是值得怀疑的，应该进一步仔细测试。
- 如果看起来软件功能完成得很正常，遇到的错误也很容易改正，则仍然应该考虑两种可能：
  - (1)软件的可靠性是可以接受的；
  - (2)所进行的测试尚不足以发现严重的错误。
- 如果经过测试一个错误也没有被发现，则很可能是因为测试配置不充分，以致不能暴露软件中潜藏的错误

7.6 白盒测试技术

- 通常把测试数据和预期输出结果称为测试用例。
- 不同的测试数据发现程序错误的能力差别很大，应该选用高效的测试数据。
- 设计测试方案的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。
- 设计测试数据的技术各有优缺点。同一种技术在不同的应用场合效果可能相差很大，因此，通常需要联合使用多种设计测试数据的技术。

7.6 白盒测试技术

- 表面看来，白盒测试是可以进行完全的测试的。只要能确定测试模块的所有逻辑路径，为每条逻辑路径设计测试用例，并评价所得到的结果，就可得到100%正确的程序。
- 实际测试中穷举法是无法实现的。

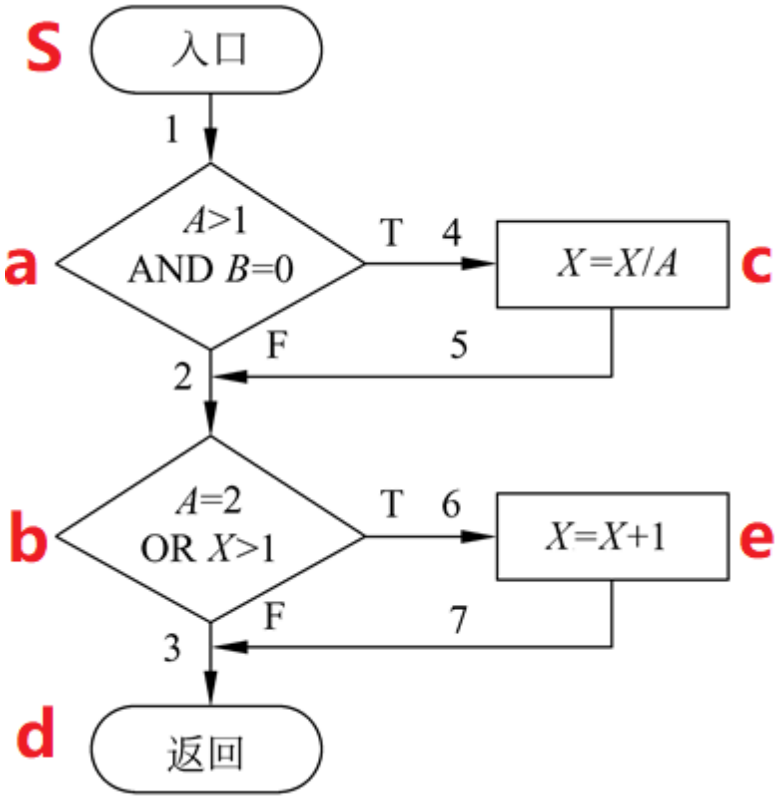
7.6.1 逻辑覆盖

- 以程序内部逻辑为基础的测试技术，它考虑测试用例对程序内部逻辑覆盖的程度。

1.语句覆盖

选择足够多的测试数据，使被测程序中每个语句至少执行一次。

为使每个语句都执行一次，程序的执行路径应该是sacbed,为此只需要输入下面的测试数据(实际上X可以是任意实数)：



被测试模块的流程图

序号	测试用例			判定		路径	覆盖标准
	A	B	X	a	b		
1	2	0	2	1	1	sacbed	语句覆盖

7.6 白盒测试技术

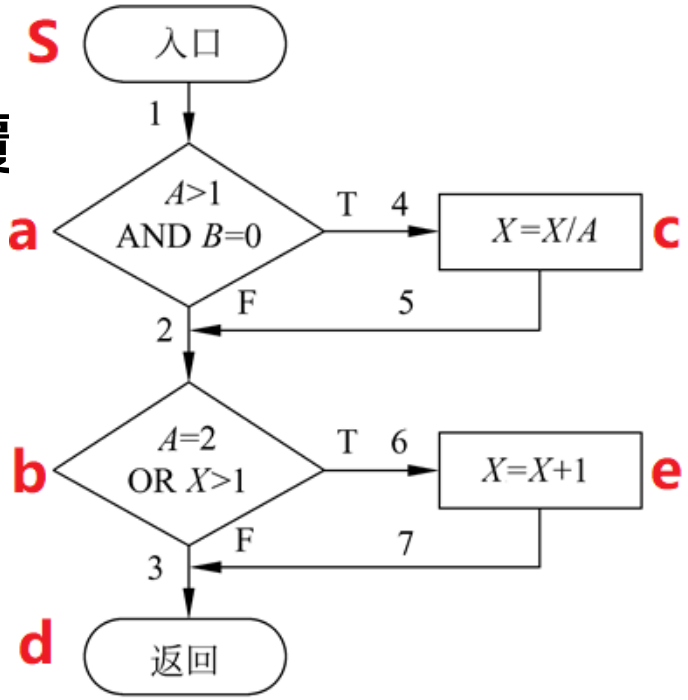
1.语句覆盖

- 对程序的逻辑覆盖很少，上例数据只测试了条件为真的情况，若实际输入条件为假时有错误显然测试不出来。
- 只关心判定表达式的值，而没有分别测试判定表达式中每个条件取不同值的情况。为执行sacbed路径以测试每个语句，只需两个判定表达式(A>1)AND(B=0)和(A=2)OR(X>1)都取真，因此使用上述一组测试数据就够了。但如果把第一个判定表达式中的AND错写成OR，或把第二个判定表达式中的X>1误写成X<1，使用上面的测试数据并不能查出这些错误

语句覆盖是很弱的逻辑覆盖标准

2.判定覆盖

- 又叫分支覆盖，不仅每个语句必须至少执行一次，而且每个判定的每个分支都至少执行一次。
- 能分别覆盖路径sacbed和sabd，或者覆盖路径sacbd和sabed的两组测试数据，都满足判定覆盖。



a和b各取1和0一次

序号	测试用例			判定		路径	覆盖标准
	A	B	X	a	b		
1	3	0	1	1	0	sacbd	判定覆盖
2	2	1	1	0	1	sabed	

判定覆盖比语句覆盖强，但仍然覆盖不全。

上面的数据只覆盖了全部路径的一半

3.条件覆盖

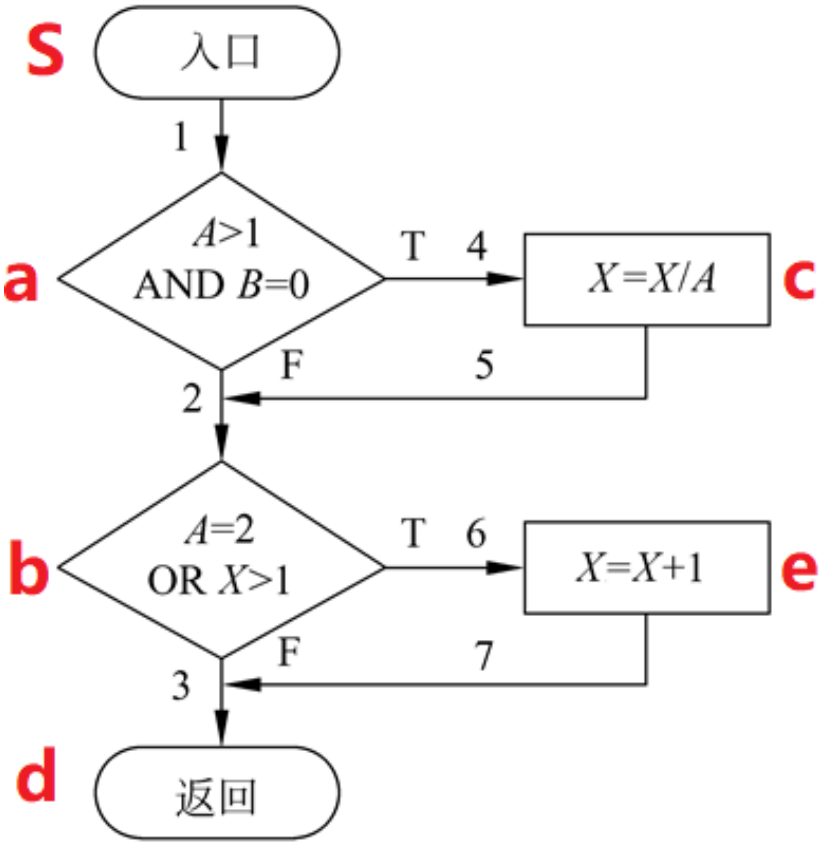
- 不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。
- 两个判定表达式，每个表达式中有两个条件。

在a点有下述各种结果出现：

A>1,A≤1,B=0,B≠0；

在b点有下述各种结果出现：

A=2,A≠2,X>1,X≤1；



7.6 白盒测试技术

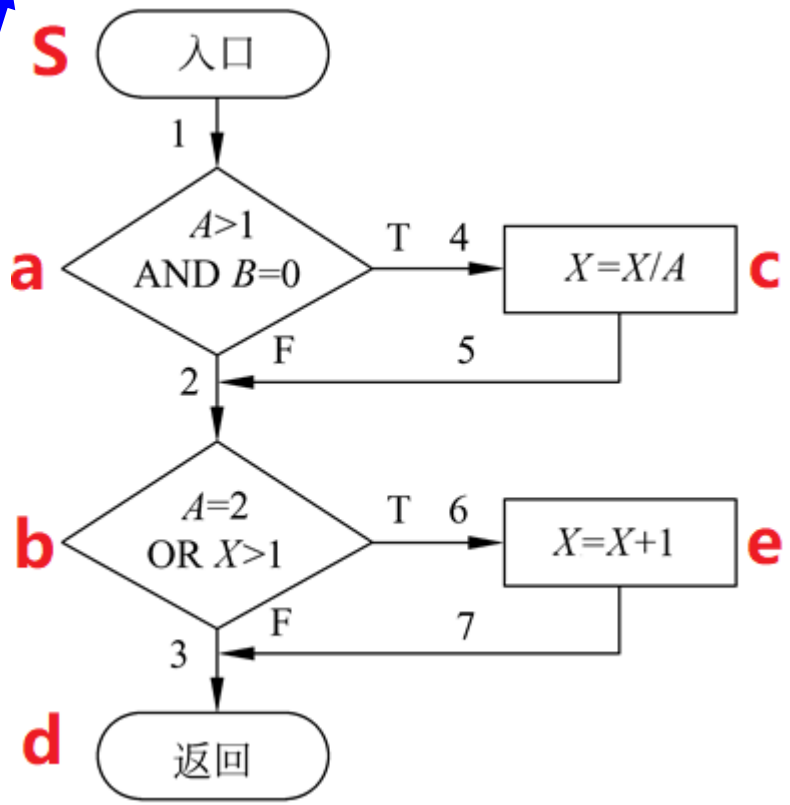
a1、a2、b1、b2各取1和0一次

序号	测试用例			条件取值				判定		路径	覆盖标准
	A	B	X	a1	a2	b1	b2	a	b		
1	2	0	4	1	1	1	1	1	1	sacbed	条件覆盖
2	1	1	1	0	0	0	0	0	0	sabd	

- 条件覆盖通常比判定覆盖强，但满足条件覆盖的测试数据不一定满足判定覆盖。

上面两组测试数据是否满足判定覆盖标准？

满足

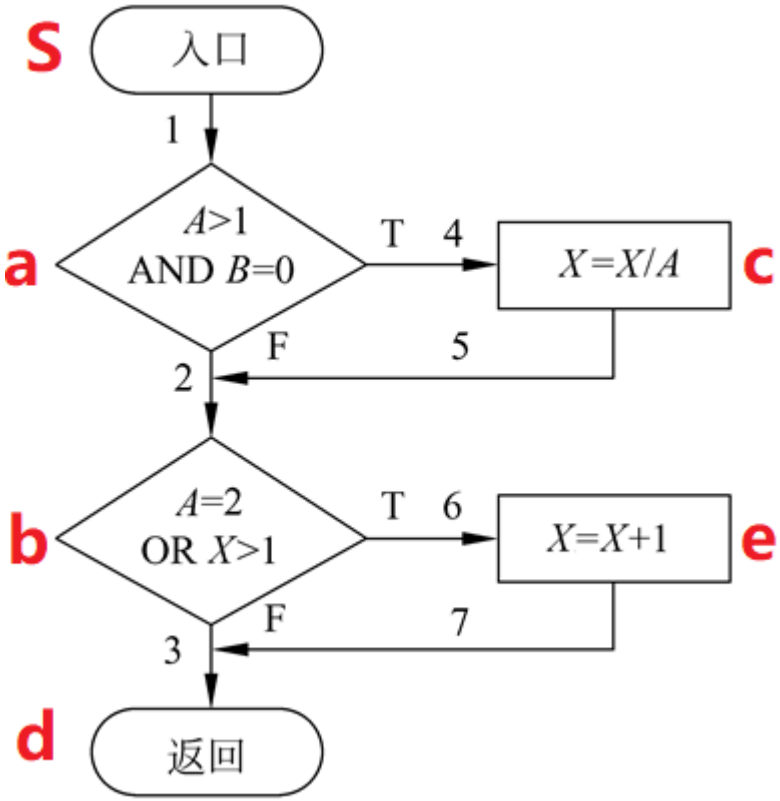


7.6 白盒测试技术

序号	测试用例			条件取值				判定		路径	覆盖标准
	A	B	X	a1	a2	b1	b2	a	b		
1	2	0	1	1	1	1	0	1	1	sacbed	条件覆盖
2	1	1	2	0	0	0	1	0	1	sabed	

是否满足判定覆盖标准?

- 上面两组测试数据只满足条件覆盖标准并不满足判定覆盖标准(第二个判定表达式的值总为真)
- 条件覆盖和判定覆盖互不相互包容。需要兼顾条件和分支，这就是判定/条件覆盖。



4.判定/条件覆盖

- 能同时满足判定覆盖和条件覆盖的逻辑覆盖。选取足够的测试数据，使判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行一次。

序号	测试用例			条件取值				判定		路径	覆盖标准
	A	B	X	a1	a2	b1	b2	a	b		
1	2	0	4	1	1	1	1	1	1	sacbed	判定/条件覆盖
2	1	1	1	0	0	0	0	0	0	sabd	

- 但这两组测试数据也就是为了满足条件覆盖标准最初选取的两组数据，因此有时判定/条件覆盖也并不比条件覆盖更强。

- 判定/条件覆盖也有缺陷。从表面来看，它测试了所有条件的取值。但实际并不是这样。
- 对判定/条件覆盖，逻辑表达式中的错误不一定能查出。因为其未考虑条件的组合情况。

5.条件组合覆盖

- 更强的逻辑覆盖标准，要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。

7.6 白盒测试技术

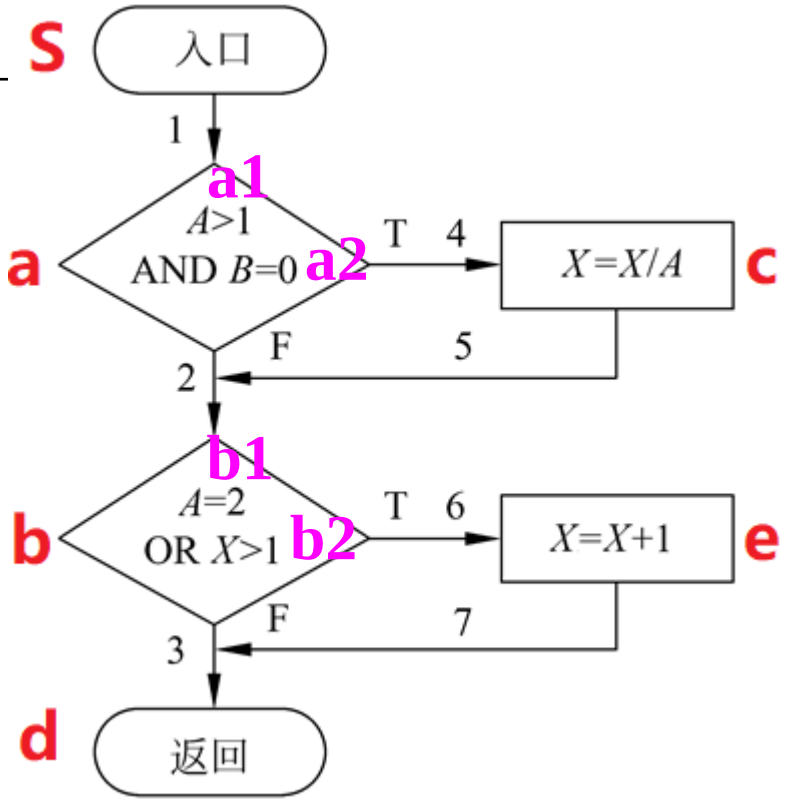
- 上例共有8种可能的条件组合：

(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



a1和a2、b1和b2各取11、10、01、00一次

序号	测试用例			条件取值				判定		路径	覆盖标准
	A	B	X	a1	a2	b1	b2	a	b		
1	2	0	4	1	1	1	1	1	1	sacbed(针对(1)和(5))	条件组合覆盖
2	2	1	1	1	0	1	0	0	1	sabed(针对(2)和(6))	
3	1	0	2	0	1	0	1	0	1	sabed(针对(3)和(7))	
4	1	1	1	0	0	0	0	0	0	sabd(针对(4)和(8))	

- 显然，满足条件组合覆盖标准的测试数据，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此**条件组合覆盖是前述几种覆盖标准中最强的**。但是，满足条件组合覆盖标准的测试数据并不一定能使程序中的每条路径都执行到，例如，上述4组测试数据都没有测试到路径sacbd。
- 因此，还**必须对程序路径的覆盖程度进行分析**，这就是点覆盖、边覆盖和路径覆盖。

6.点覆盖

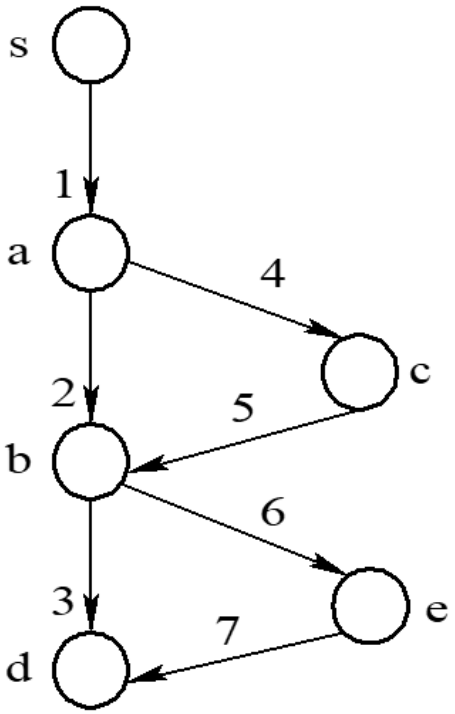
- 图论中点覆盖的定义如下：如果连通图G的子图G′是连通的，而且包含G的所有结点，则称G′是G的点覆盖。
- 正常情况下，程序流程图是连通的有向图。满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过流图的每个结点一次，由于流图的每个结点与一条或多条语句相对应，显然，**点覆盖标准和语句覆盖标准是相同的**。

7.边覆盖和路径覆盖

- 图论中**边覆盖**的定义是：如果连通图G的子图G″是连通的，而且包含G的所有边，则称G″是G的边覆盖。边覆盖的测试标准是要求选取足够多测试数据，使得**程序执行路径至少经过流图中每条边一次**。**通常边覆盖和判定覆盖是一致的**。
- **路径覆盖**的含义是，选取足够多测试数据，**使程序的每条可能路径都至少执行一次**(如果程序图中有环，则要求每个环至少经过一次)。

7.6 白盒测试技术

- 路径覆盖相对来说是比较强的逻辑覆盖标准
- 但路径覆盖并没有检验表达式中条件的各种组合情况，而只考虑每个判定表达式的取值
- 可把路径覆盖和条件覆盖组合使用。



- 共有四条可执行的路径：

1-2-3；1-2-6-7；1-4-5-3；1-4-5-6-7

a和b取11、10、01、00一次

序号	测试用例			条件取值				判定		路径	覆盖标准
	A	B	X	a1	a2	b1	b2	a	b		
1	1	1	1					0	0	1-2-3，即sabd	路径覆盖
2	1	1	2					0	1	1-2-6-7，即sabed	
3	3	0	1					1	0	1-4-5-3，即sacbd	
4	2	0	4					1	1	1-4-5-6-7，即sacbed	