- 张瑾
- PMP
- 微软认证高级项目经理
- 广东省软件协会过程改进专业委员会专家
- 希赛顾问
- 有多年项目管理经验，先后担任项目经理、质量经理和EPG过程改进经理、副总经理等职务。
- 2007年撰写了《自动化软件测试》一书，2008年撰写了《WWF工作流开发指南》并参与编写《信息系统项目管理师论文精粹》一书。《CMMI之功能点分析法》等众多文章被国内知名网站发表。
- 在软件项目管理，软件过程改进、软件自动化测试等方面有深入的研究。
- 联系方式：www.zhang-jin.net
- 软件项目管理及过程改进MSN群：group118061@msnzone.cn

单元测试基本原理

《自动化软件测试》

2008年 02月版

张瑾

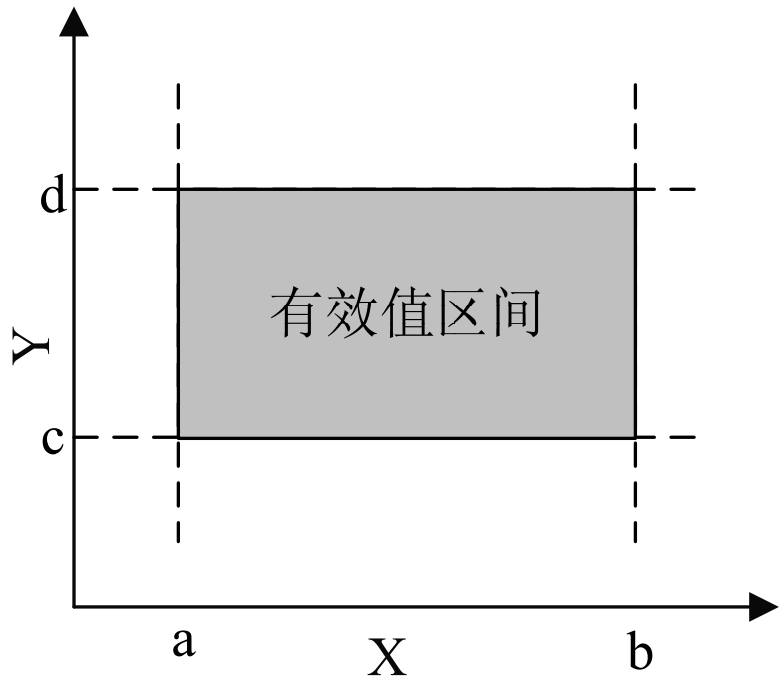
2007-12-04

培训内容

- 边界值测试
- 语句覆盖
- 判断覆盖
- 条件覆盖
- 判断—条件覆盖
- 条件组合覆盖
- 路径覆盖

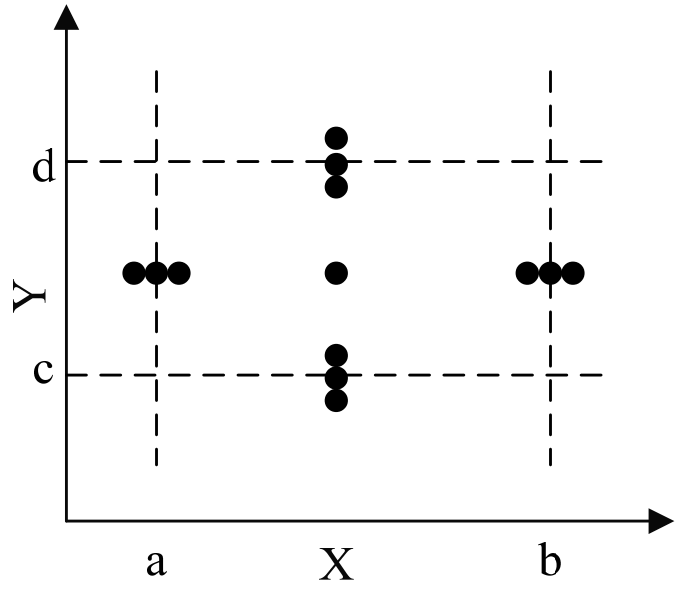
边界值

- 变量和函数都有自己的边界，能够准确找到各种边界值是软件测试人员所需要具备的基本功。对于值来说它的边界值是最大值和最小值，对于函数来说它的边界就是一个区域。
- 例如：函数F有两个输入函数x和y，如图1-5所示函数F的有效值范围就是两个输入参数x和y的有效值所组成的区间。
- a≤X≤b
- c≤y≤d



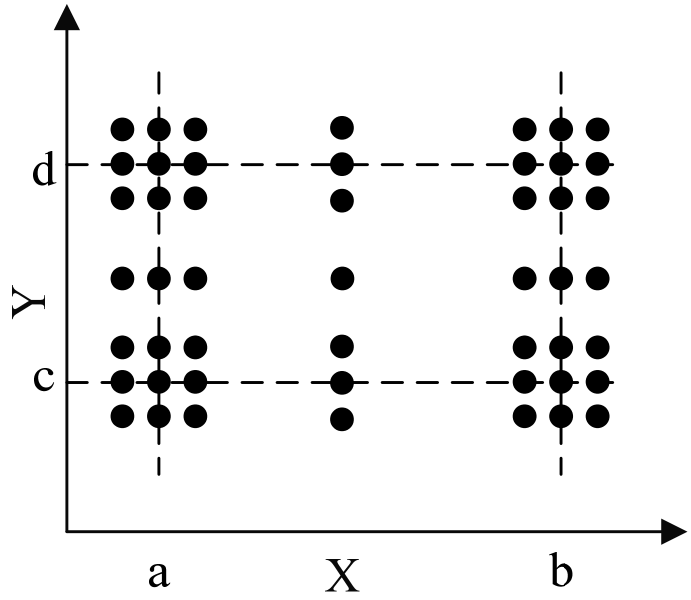
健壮性边界测试

- 健壮性边界测试就是要取最大值、最小值、正常值、略大值、略小值进行
- 测试，以验证在这五种情况下是否正确。



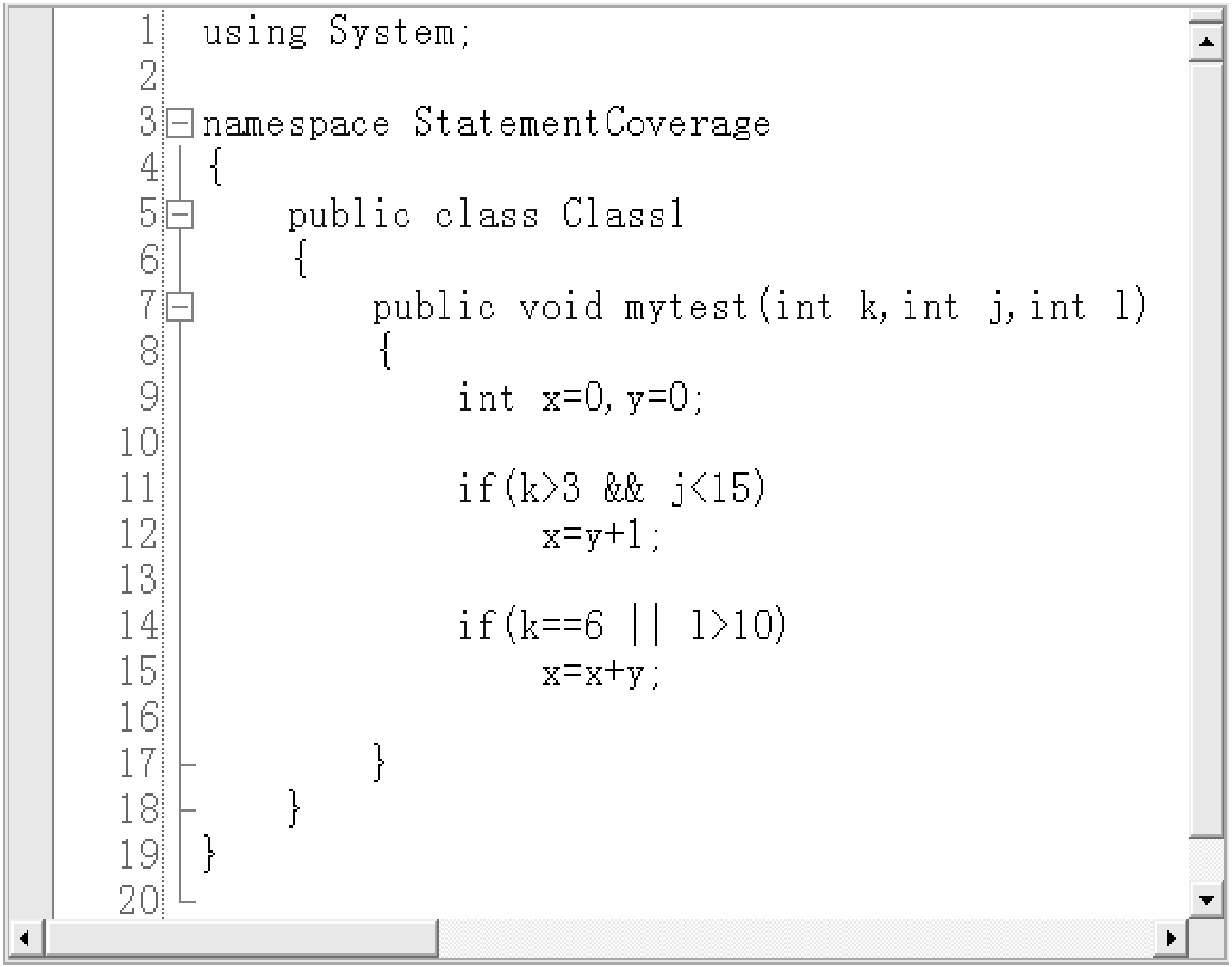
健壮性最坏情况下边界测试

- 健壮性最坏情况下边界测试除了要对以上五种境况进行测试外，还要对其进行笛卡尔积进行计算，以覆盖更多的边界情况。

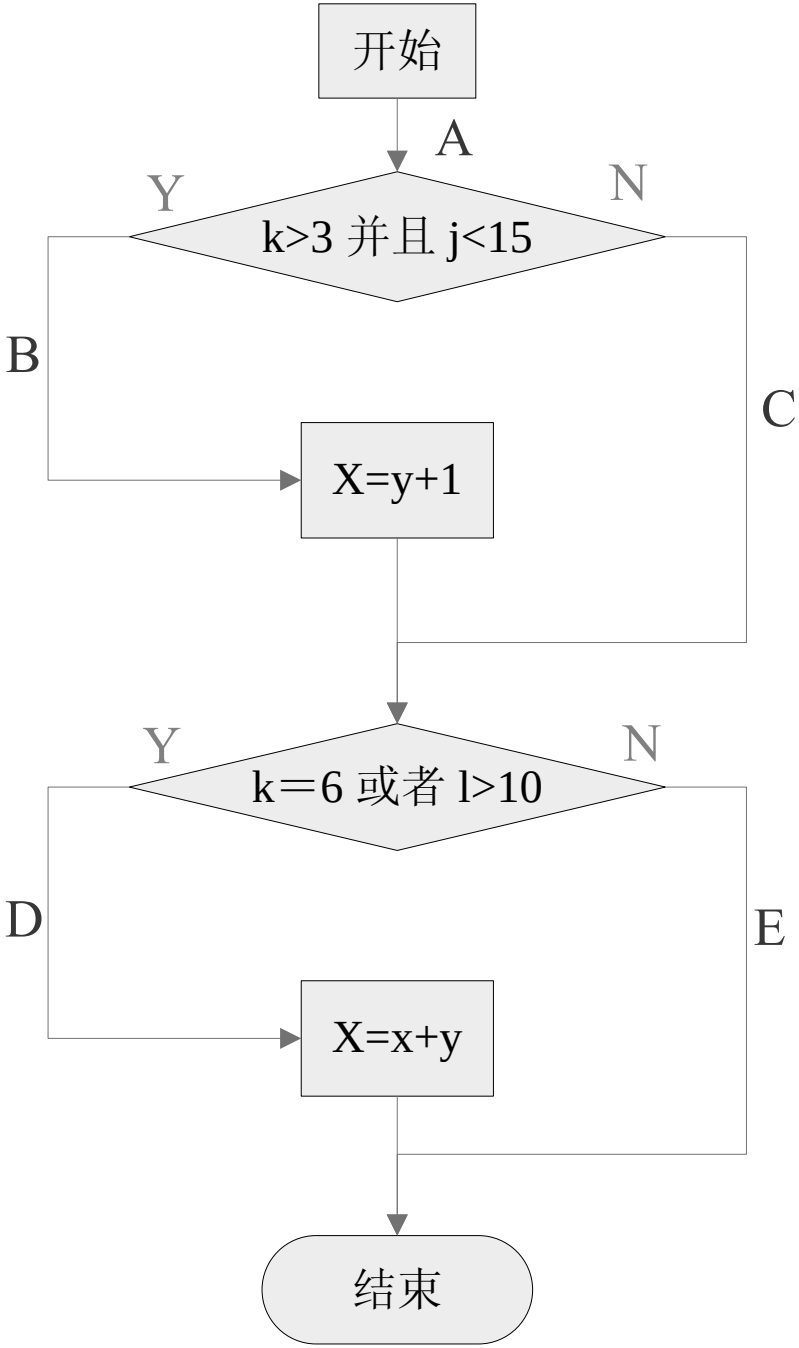


单元测试

- 白盒测试讲究的是代码覆盖的情况，目标是把所有代码和各种可能性都完全进行独立的测试。代码中间往往嵌套着各种业务逻辑，由各种流控制语句组合实现，这就可能会有多种情况发生。程序的流程图是对其逻辑进行分析的最好工具，先看一下如图1-11所示的代码范例，以及根据其逻辑画出的流程图1-12。



```
1  using System;
2
3  namespace StatementCoverage
4  {
5      public class Class1
6      {
7          public void mytest(int k,int j,int l)
8          {
9              int x=0,y=0;
10
11             if(k>3 && j<15)
12                 x=y+1;
13
14             if(k==6 || l>10)
15                 x=x+y;
16
17         }
18     }
19 }
20
```


开始
A
Y
N
k>3 并且 j<15
B
C
X=y+1
Y
N
k＝6 或者 l>10
D
E
X=x+y
结束

语句覆盖

- 语句覆盖（Statement Coverage）从字面上理解就是将程序中每一行代码，在白盒测试时都要检查一遍。
- 语句覆盖要求设计足够多的测试用例（当然越少越好），使得程序中每条语句至少被执行一次。但它无法实现对所有逻辑分支进行覆盖，因此它是一种最基本的覆盖原则。

表 1-1 语句覆盖的测试用例

编号	K	J	L	程序执行的路径
a	6	10	15	A—> B—>D

语句覆盖

- 提示：
- 　　将表1-1中测试数据带入程序进行运算，该用例执行了代码中的所有语句，显然白盒测试的目的达到了。但仔细研究一下程序会发现一个问题。如果将一个判断中的“&&”改为“||”，或者将第二个判断中的“||”改为“&&”，该测试用例仍可以覆盖所有的语句。
- 因此语句覆盖无法判断逻辑是否存在问题，因此语句覆盖也是最弱的一种覆盖原则。

判断覆盖

- 在语句覆盖中可以发现即使所有代码都经过测试，但对逻辑却无法判断其是否正确，因此引入判断覆盖（Decision Coverage）的概念，判断覆盖也称为分支覆盖。要对每个if语句的逻辑进行独立测试，每个逻辑也就对应了语句中的if和else分支。
- 判断覆盖要求设计足够多的测试用例，使得代码中每个判断至少取一次为真值，取一次为假值，即：程序中的每个if语句的分支至少执行一次。

表 1-2　判断覆盖的测试用例

编号	K	J	L	程序执行的路径
a	6	13	10	A—> B—>D
b	2	20	10	A—> C—>E

判断覆盖

- 提示：
- 将表1-2中的两组测试数据分别带入程序运行，可以看到两个if语句分别都取了一次真值和一次假值。判断覆盖不但满足了逻辑的判断，而且还做到了语句覆盖，因此判断覆盖比语句覆盖稍强一些。
- 接下来将代码做一点改变，假如将第二个判断条件中“l>10”改成“l<10”，该测试用例仍可以执行并达到预期的测试目的。由此可见判断覆盖无法对判断条件的组合进行检查，不一定能发现某个条件是否存在错误。

条件覆盖

- 可以发现判断覆盖虽然弥补了语句覆盖的缺点，但其自身还是存在一定缺陷。条件覆盖（Condition Coverage）要求设计足够多的测试用例，使得判断中的每个条件的每种可能性至少运行一遍。通过对范例进行分析发现，它是由两个if语句构成，每个if语句包含了两个判断条件，因此可以设计一种测试用例，对这些判断条件的每种可能性进行覆盖。

表 1-3　条件覆盖的第一个判断

k>3	取真值	T1
k<=3	取假值	-T1
j<15	取真值	T2
j>=15	取假值	-T2

表 1-4 条件覆盖的第二个判断

K=6	取真值	T3
K!=6	取假值	-T3
l>10	取真值	T4
L<=10	取假值	-T4

表 1-5　条件覆盖的测试用例 1

编号	K	J	L	程序执行的路径	覆盖条件	覆盖分支
a	6	13	10	A—> B—>D	T1、T2、T3、-T4	BD
b	3	14	10	A—> C—>E	-T1、T2、-T3、-T4	CE
c	6	15	11	A—> C—>D	T1、-T2、T3、T4	CD

条件覆盖

- **提示：**
- 将表1-5中的测试数据分别带入程序进行运行。可以看出使用三个用例，将四个判断条件的八种情况全部覆盖。不但做到了语句覆盖，而且也覆盖了所有的分支，并且每个分支中每个判断条件的真、假两种可能性也全部进行了测试。

表 1-6　条件覆盖的测试用例 2

编号	K	J	L	程序执行的路径	覆盖条件	覆盖分支
a	3	13	11	A—> C—>D	-T1、T2、-T3、T4	CD
b	6	16	9	A—> C—>D	T1、-T2、T3、-T4	CD

条件覆盖

- **提示：**
- 再将表1-6中的测试数据带入程序进行运行，可以看出使用两个用例，也可以将两个if语句中四个判断条件的八种情况全部覆盖。
- 但该测试用例未能覆盖全部分支，只对第一个条件的取假的分支和第二个条件取真的分支进行覆盖，也就是说只覆盖了四个分支中的两个，因此条件覆盖也存在缺陷。

判断—条件覆盖

- 用户可以知道条件覆盖并不能取代判断覆盖，他们各有优势，这里就将他们进行结合，提供一种“判断—条件覆盖”。
- 用户需要设计足够多的测试用例，使得判断中每个条件的所有可能性至少出现一次，每个分支本身所有可能的结果也至少出现一次。判断—条件覆盖满足了判断覆盖的原则，同时也满足了条件覆盖准则，弥补了二者的不足。

判断—条件覆盖

表 1-7　判断-条件覆盖的测试用例

编号	K	J	L	程序执行的路径	覆盖条件	覆盖分支
a	6	13	15	A—> B—>D	T1、T2、T3、T4	BD
b	3	16	10	A—> C—>E	-T1、-T2、-T3、-T4	CE

- **提示：**
- 将表1-7中两组测试数据带入程序进行运算。可以看出不但满足了分支覆盖的目的，而且也满足了各分支中每个条件的所有可能性。
- 如果对代码再进行认真分析会得到在第一个表达式中，只有两个条件均满足的情况下才为真，因此当K＝3表达式为假时，J的值就不再起作用。但是对于第二个表达式，只要有一个条件满足时，该表达式就为真。因此当K＝6表达式为真时，L的值就不再起作用。
- 因此，“判断—条件”覆盖不一定能检查出表达式中的条件的错误。

条件组合覆盖

- 既然“判断—条件覆盖”都无法达到最终的效果，那就将“条件覆盖”、“判断覆盖”、“判断—条件覆盖”的目的进行组合，推出另一种覆盖方式“条件组合覆盖”。
- 用户需要设计足够多的测试用例，使得每个判断中各条件结果可能性的组合至少出现一次。
- 各位对代码进行分析，有两个if语句，每个if语句都包含两个判断条件，因此对同一if语句中的判断条件的可能性进行组合，每个判断可以得到四种组合方式。

表 1-8 条件组合覆盖的第一个判断

条件组合 1#	k>3，j<15	T1，T2	第一个判断为真
条件组合 2#	k>3，j>=15	T1，-T2	第一个判断为假
条件组合 3#	k<=3，j<15	-T1，T2	第一个判断为假
条件组合 4#	k<=3，j>=15	-T1，-T2	第一个判断为假

表 1-9 条件组合覆盖的第二个判断

条件组合 5#	k= 6，l>10	T3，T4	第二个判断为真
条件组合 6#	k=6，l<=10	T3，-T4	第二个判断为真
条件组合 7#	k!=6，l>10	-T3，T4	第二个判断为真
条件组合 8#	k!=6，l<=10	-T3，-T4	第二个判断为假

表 1-10 条件组合覆盖的测试用例：

编号	K	J	L	程序执行的路径	覆盖条件	覆盖组合	覆盖分支
A	6	13	15	A—> B—>D	T1、T2、T3、T4	#1、#5	BD
B	6	16	10	A—> C—>D	T1、-T2、T3、-T4	#2、#6	CD
C	3	13	12	A—> C—>D	-T1、T2、-T3、T4	#3、#7	CD
D	3	15	10	A—> C—>E	-T1、-T2、-T3，-T4	#4、#8	CE

条件组合覆盖

- **提示：**
- 将表1-10中的四组测试数据带入程序进行运算。可以看到所有的条件组合均被覆盖。该程序有四条路径，但本用例覆盖了所有条件分支，也覆盖了各种条件的组合，但却少了一条路径A—> B—> E。路径能否完全覆盖，在白盒测试中却是一个非常重要的问题。

路径覆盖

- 可以看出以上五种覆盖方式各有千秋。路径是软件测试最需要关注的地方，因此路径覆盖也是一种常用的覆盖方式。用户需要设计足够多的测试用例，使其覆盖程序中所有可能的路径，该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要全面。
- 如表1-11所示对该测试代码进行分析，其一共有四条路径
- (1)A－> B－>D
- (2)A－> C－>E
- (3)A－> C－>D
- (4)A－> B－> E

路径覆盖

表 1-11 路径覆盖的测试用例

编号	K	J	L	程序执行的路径	覆盖条件
a	6	13	15	A—> B—>D	T1、T2、T3、T4
b	4	16	10	A—> C—>E	T1、-T2、-T3、-T4
c	3	13	12	A—> C—>D	-T1、T2、-T3、T4
d	4	13	10	A—> B—> E	T1、T2、-T3，T4

- 提示：
- 在实际测试工作中，路径其实是一个非常庞大的数字，要想全部覆盖，往往不太现实。特别是在以盈利为目的的项目中，项目进度、成本与质量之间需要进行权衡。