软件测试

授课教师：张剑波

授课班级：**111161-2**班

2019年3月

Chapter2 软件测试技术

内容提要

- 静态测试
- 动态测试
  - 经典测试方法
  - 基于质量特征的测试
  - 基于经验的测试

两个案例引发的思考

- 案例一

某移动互联网企业项目正在匆忙的进行中。由于项目的需求，需要招聘若干测试工程师，其中小张因为白盒测试经验丰富而被录取。项目进行过程中，小张搭建了很完善的白盒测试框架并且用例覆盖度也很高，自信满满的等待着项目圆满结束。项目结束后，合作的客户企业以及用户拿到软件之后大为吃惊，产品功能和界面有多处与需求不符，故项目最终失败告终。

两个案例引发的思考（续）

- 案例二：

某企业大型**ERP**系统因新需求需要重新定制。整个项目过程中，多名测试工程师均仅采取黑盒测试的方式，项目最终因系统定制很人性化而圆满告终。但好景不长，系统在正式使用的过程中遭遇了多次数据处理错误或崩溃，原因正因为测试过程中并没有覆盖到各种实际场景，最终这套系统也因此而不得不重新开发。

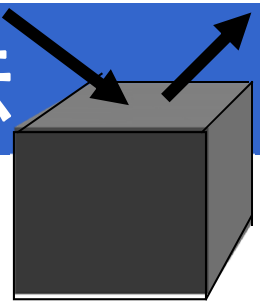
软件测试的“黑白之道”

- 在一个项目中黑盒和白盒并没有一个绝对的比例，也没有优劣之分，需要根据具体项目的背景或不同的阶段选择合适的测试策略。
- 使用黑盒白盒的策略除了与项目阶段有关，还有项目背景有关。一些相对来讲用户群很大，产品集成成本并不大的项目，黑盒测试占的比例相对就会很大。比如各种平台的应用程序、企业系统等。而相对来讲一些大型项目，比如航空航天，白盒测试就必不可少，并且颗粒度需要更细，而黑盒占的比例就会相应减少，同时大大减少了测试的成本。

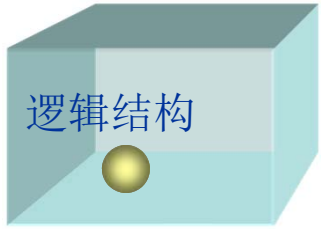
软件测试的“黑白之道”（续）

- 在项目产品拥有很多模块，而每个模块处于刚刚开发的阶段的时候，产品模块并不具有界面或完整暴露的接口，从而开发人员开发完毕只能通过白盒测试进行自测。此时的测试更多的会覆盖到模块中每段代码的逻辑以及判断路径，从而在代码结构内部保证了质量。
- 产品达到一定完成度之后，可以通过黑盒测试对于产品的界面以及模块合并之后的功能进行测试。黑盒测试侧重点则更关心产品的界面显示、功能是否与需求说明书相符，操作体验是否人性化。很多产品界面显示是否友好远远比功能重要的多。

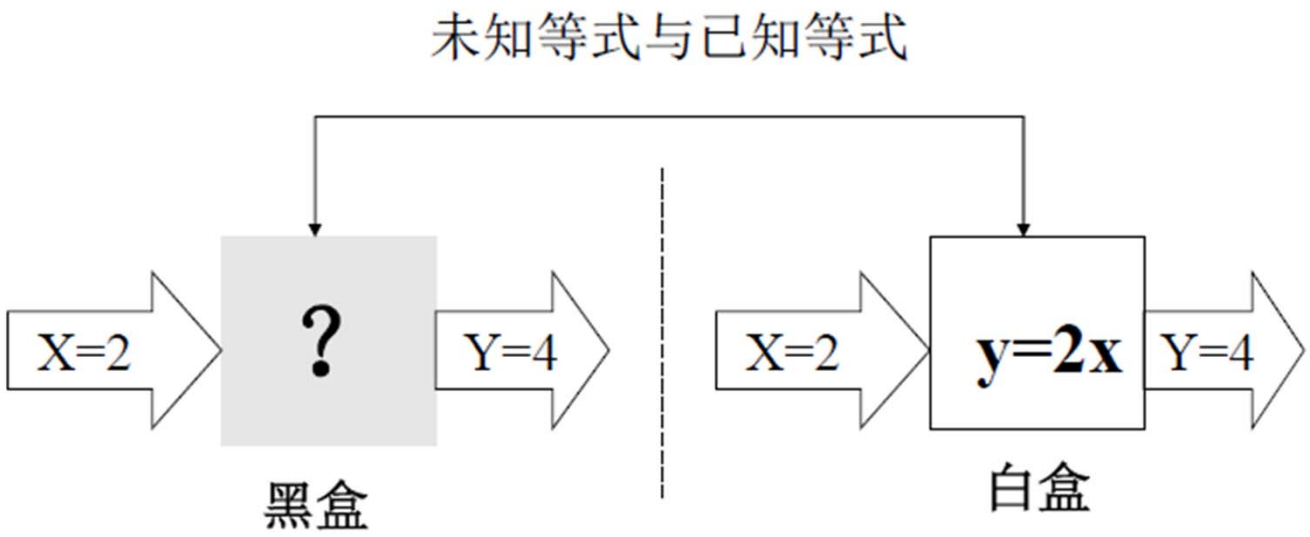
（一）经典测试方法



- 黑盒测试（功能测试，数据驱动测试）
  - 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，程序是否能适当地接收输入数据并产生正确的输出信息。
- 白盒测试（结构测试，逻辑驱动测试）
  - 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照预定要求正确工作。



黑盒测试 VS. 白盒测试



- 黑盒测试：从用户角度出发进行测试
- 白盒测试：能对程序内部的特定部位进行覆盖测试

黑盒测试的目的

- 黑盒测试方法是在程序接口上进行测试，主要是为了发现以下错误:
  - 是否有不正确或遗漏了的功能?
  - 在接口上，输入能否正确地接受? 能否输出正确的结果?
  - 是否有数据结构错误或外部信息（例如数据文件）的访问错误?
  - 性能上是否能够满足要求?
  - 是否有初始化或终止性错误?

白盒测试的目的

- 主要对程序模块进行如下的检查：
  - 对程序模块的所有独立的执行路径至少测试一次；
  - 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次；
  - 在循环的边界和运行界限内执行循环体；
  - 测试内部数据结构的有效性，等。

1、白盒测试法

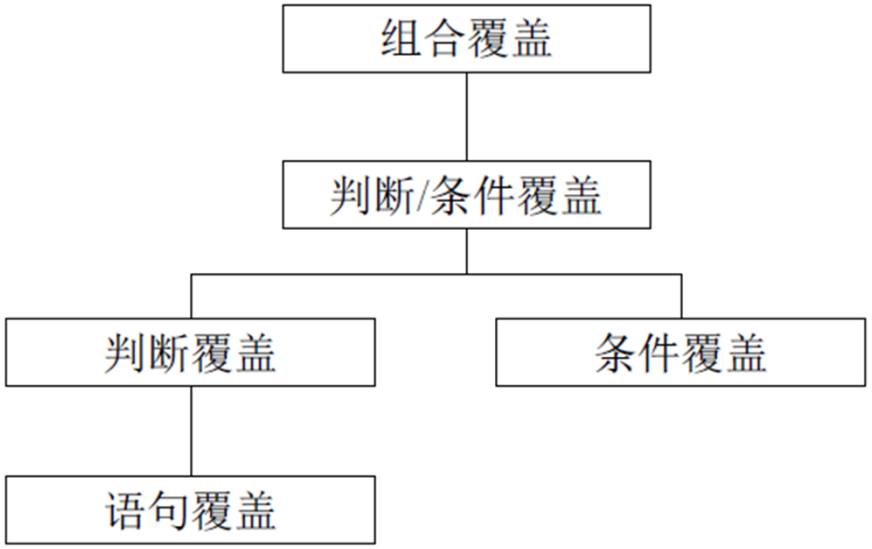
- 白盒测试常用技术：
  - 逻辑覆盖法：最常用
  - 程序插桩技术
  - 基本路径法
  - 符号测试
  - 错误驱动测试

（1）逻辑覆盖法

- 白盒法又称为逻辑覆盖法，其测试用例选择，是按照不同覆盖标准确定的。

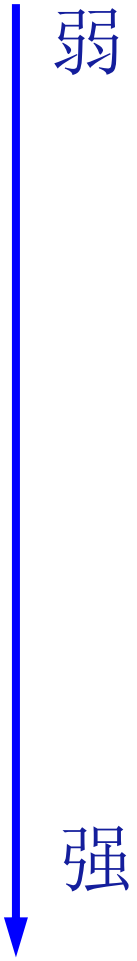
弱 → 强

语句覆盖　判定覆盖　条件覆盖　判定条件覆盖　条件组合覆盖

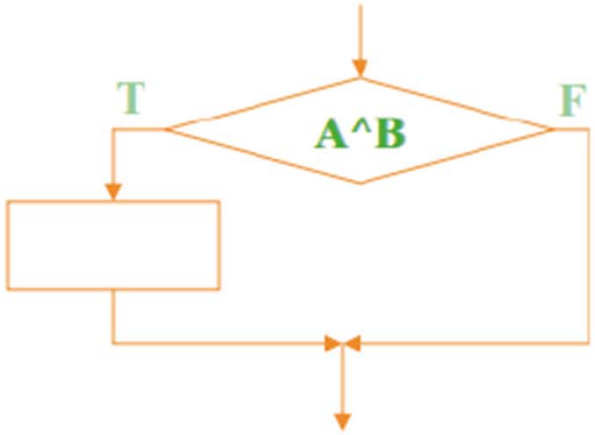
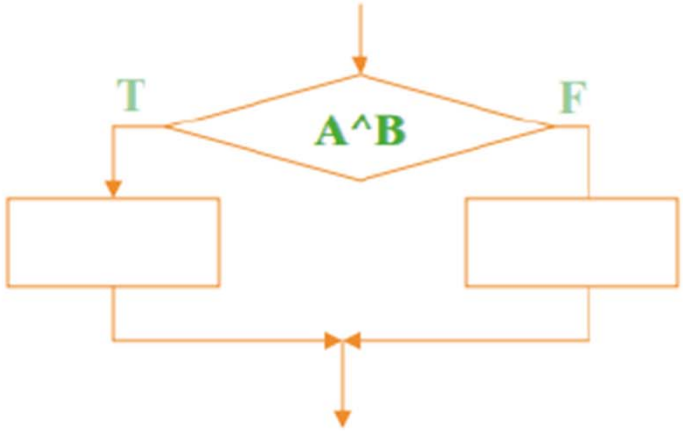


白盒测试常用的覆盖标准

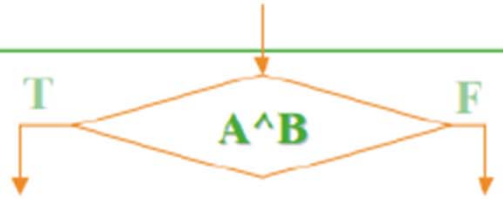
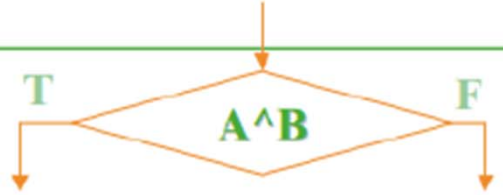
- 语句覆盖：选择足够的测试用例，使得程序中每个可执行语句至少都能被执行一次。
- 判定覆盖（分支覆盖）： 执行足够的测试用例，使得程序中每个判定至少都获得一次“真”值和“假”值。
- 条件覆盖：执行足够的测试用例，使得判定中的每个条件获得各种可能的结果。
- 判定/条件覆盖： 执行足够的测试用例，使得判定中每个条件取到各种可能的值，并使每个判定取到各种可能的结果。
- 条件组合覆盖： 执行足够的例子，使得每个判定中条件的各种可能组合都至少出现一次。



白盒测试 示例

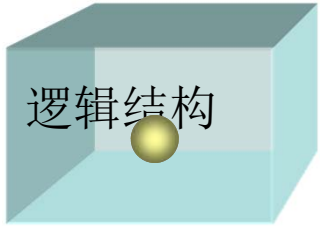
覆盖标准	程序结构举例	测试用例 应满足的条件
语句覆盖		A^B=.T.
判定覆盖		A^B=.T. A^B=.F.

白盒测试 示例（续）

覆盖标准	程序结构举例	测试用例应 满足的条件
条件覆盖		A=.T. A=.F. B=.T. B=.F.
判定/条件 覆盖		A^B=.T. , A^B=.F. A=.T. A=.F. B=.T. B=.F.
条件组合 覆盖		A=.T. ^ B=.T. A=.T. ^ B=.F. A=.F. ^ B=.T A=.F. ^ B=.F.

白盒测试的步骤

- 选择逻辑覆盖标准。
- 按照覆盖标准列出所有情况。
- 选择确定测试用例。
- 验证分析运行结果与预期结果。



测试方案

- 测试方案包括
  - 具体的测试目的、应该输入的测试数据和预期的结果（测试用例）。
- 设计测试方案的基本目标：
  - 确定一组最可能发现某个错误或某类错误的测试数据。

白盒测试举例

```
void Func(double A, double B,
double X)
{
  if((A>1) && (B==0))
     X=X/A;

  if((A==2) || (X>1))
    X=X+1;
}
```

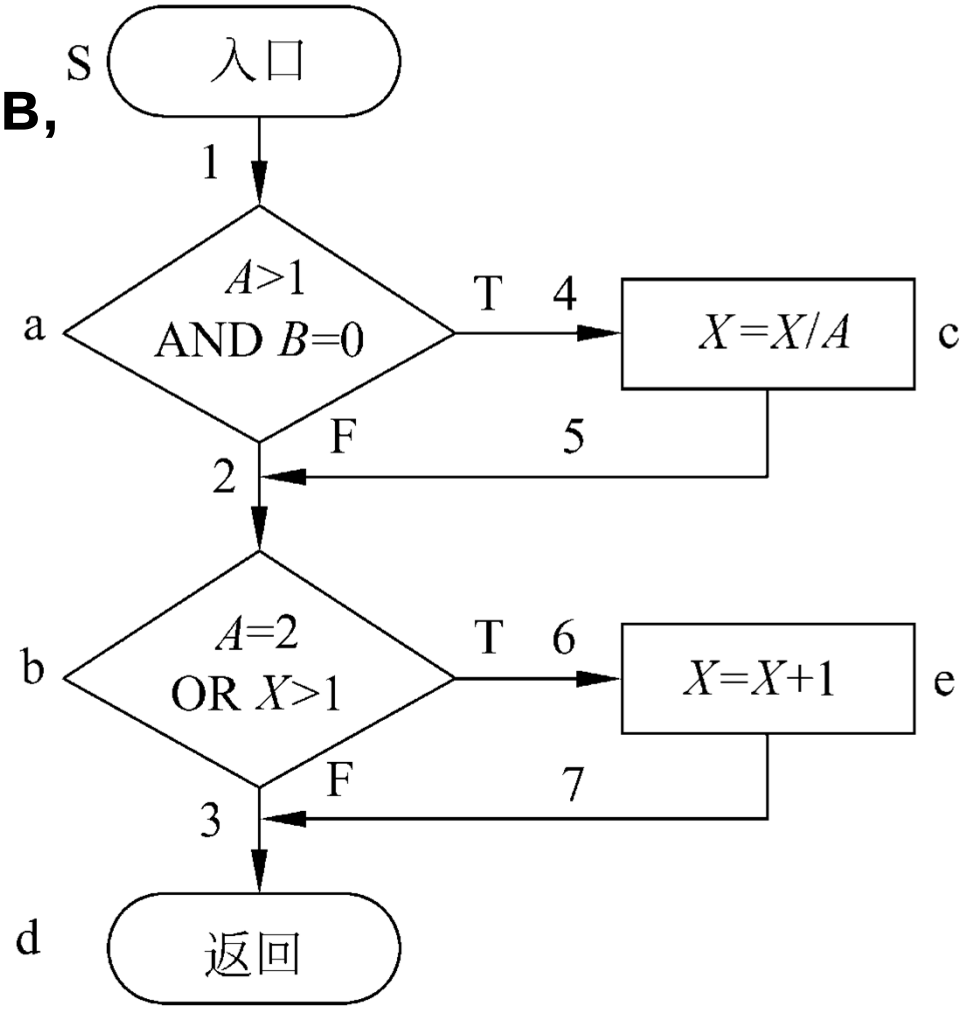
四条路径：

L1: a-c-e T T

L2: a-b-d F F

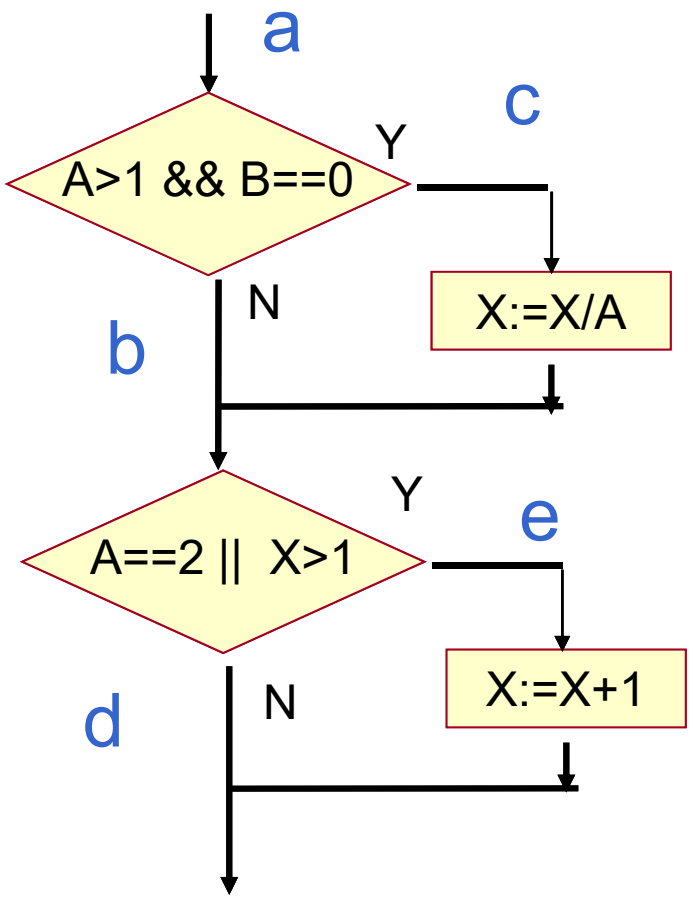
L3: a-b-e F T

L4: a-c-d T F



1）语句覆盖

❖ 设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。



满足语句覆盖的情况：

执行路径：ace

用例格式：

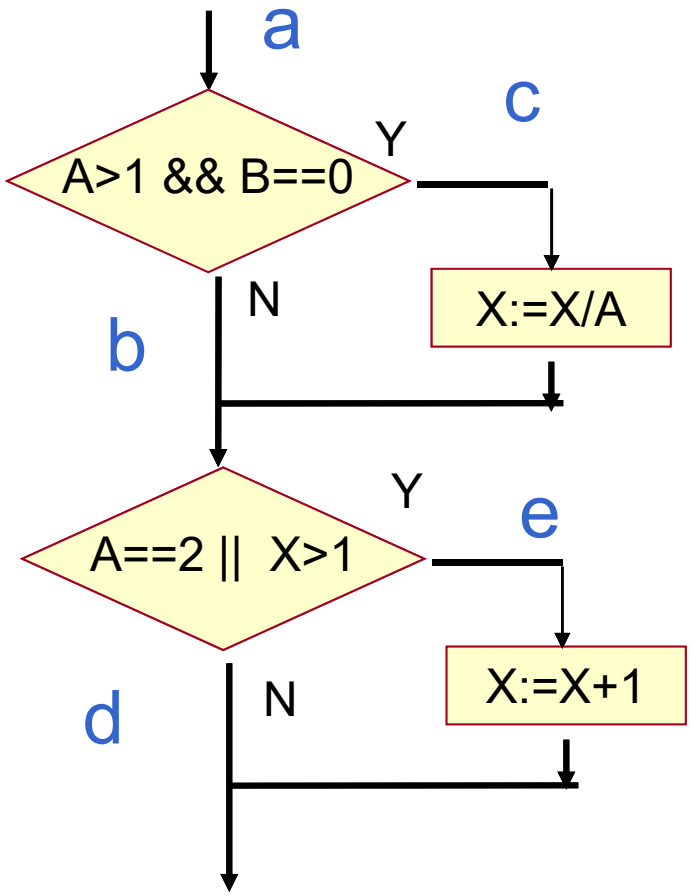
[输入(A, B, X)， 输出(A, B, X)]

选择用例：

[(2, 0, 4), (2, 0, 3)]

2）判定覆盖：分支覆盖

❖ 设计若干个测试用例，运行被测程序，使得程序中每个判定的取真分支和取假分支至少经历一次



覆盖情况：应执行路径

ace ∧ abd 或： **acd ∧ abe**

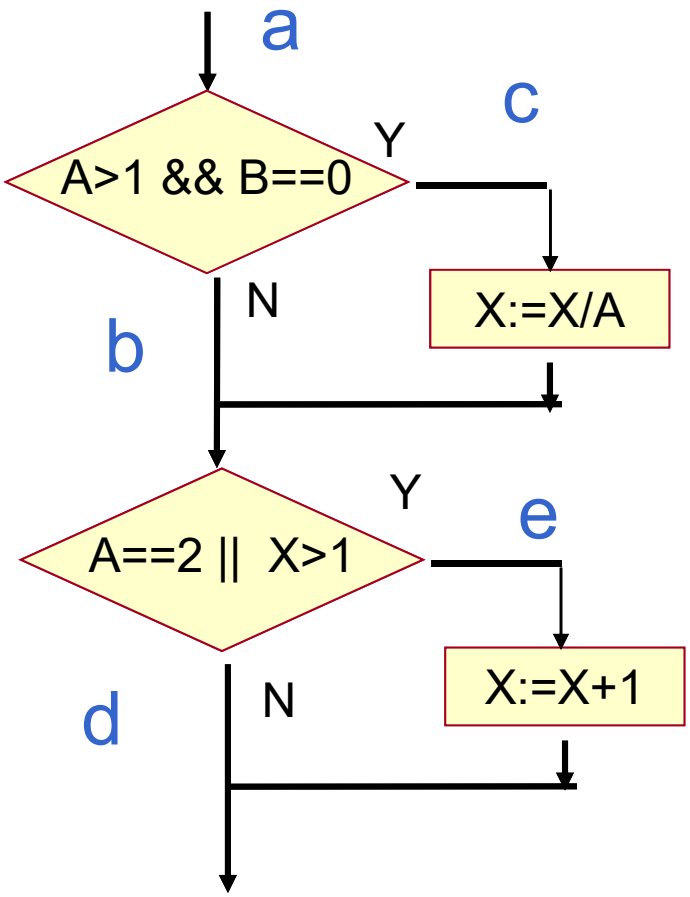
选择用例(其一）：

(1) [(2, 0, 4), (2, 0, 3)] ace
[(1, 1, 1), (1, 1, 1)] abd

(2) [(2, 1, 1), (2, 1, 2)] abe
[(3, 0, 3), (3, 1, 1)] acd

3）条件覆盖

- 设计若干个测试用例，运行被测程序，使得程序中每个判定的每个条件的可能取值至少执行一次。



1）第一个判断：

条件 **A＞1** 取真为 T_1，取假为 $\overline{T_1}$

条件 **B＝0** 取真为 T_2，取假为 $\overline{T_2}$

2）第二个判断：

条件**A＝2** 取真为 T_3，取假为 $\overline{T_3}$

条件**X＞1** 取真为 T_4，取假为 $\overline{T_4}$

条件覆盖（续1）

应满足以下覆盖情况：

判定一: A>1, A≤1, B=0, B≠0

判定二: A=2, A≠2, X>1, X≤1

测试用例	条件取值
【(2, 0, 4)，(2, 0, 3)】	$T_1T_2T_3T_4$
【(1, 1, 1)，(1,1, 1)】	$\overline{T_1}\overline{T_2}\overline{T_3}\overline{T_4}$
或	
【(1, 0, 3)，(1, 0, 4)】	$\overline{T_1}T_2\overline{T_3}T_4$
【(2, 1, 1)，(2, 1, 2)】	$T_1\overline{T_2}T_3\overline{T_4}$

注意：满足条件覆盖，不一定满足判断覆盖！

课堂练习

- 如果一个判定中的复合条件表达式为：

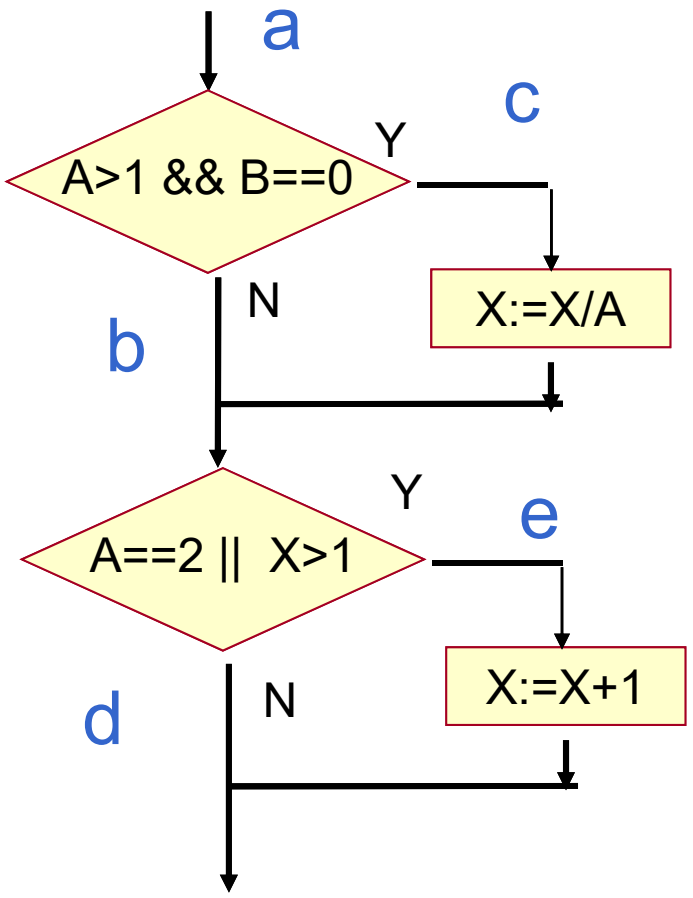
  （A > 1）or（B <= 3），则为了达到100%的条件覆盖率，至少需要设计 ____个测试用例。

  A. 1　　B. 2　　C. 3　　D. 4

  B

4）判定/条件覆盖

- 设计若干个测试用例，运行被测程序，同时满足判定覆盖和条件覆盖。



应满足以下覆盖情况：

条件: **A>1, A≤1, B=0, B≠0**

A=2, A≠2, X>1, X≤1

应执行路径

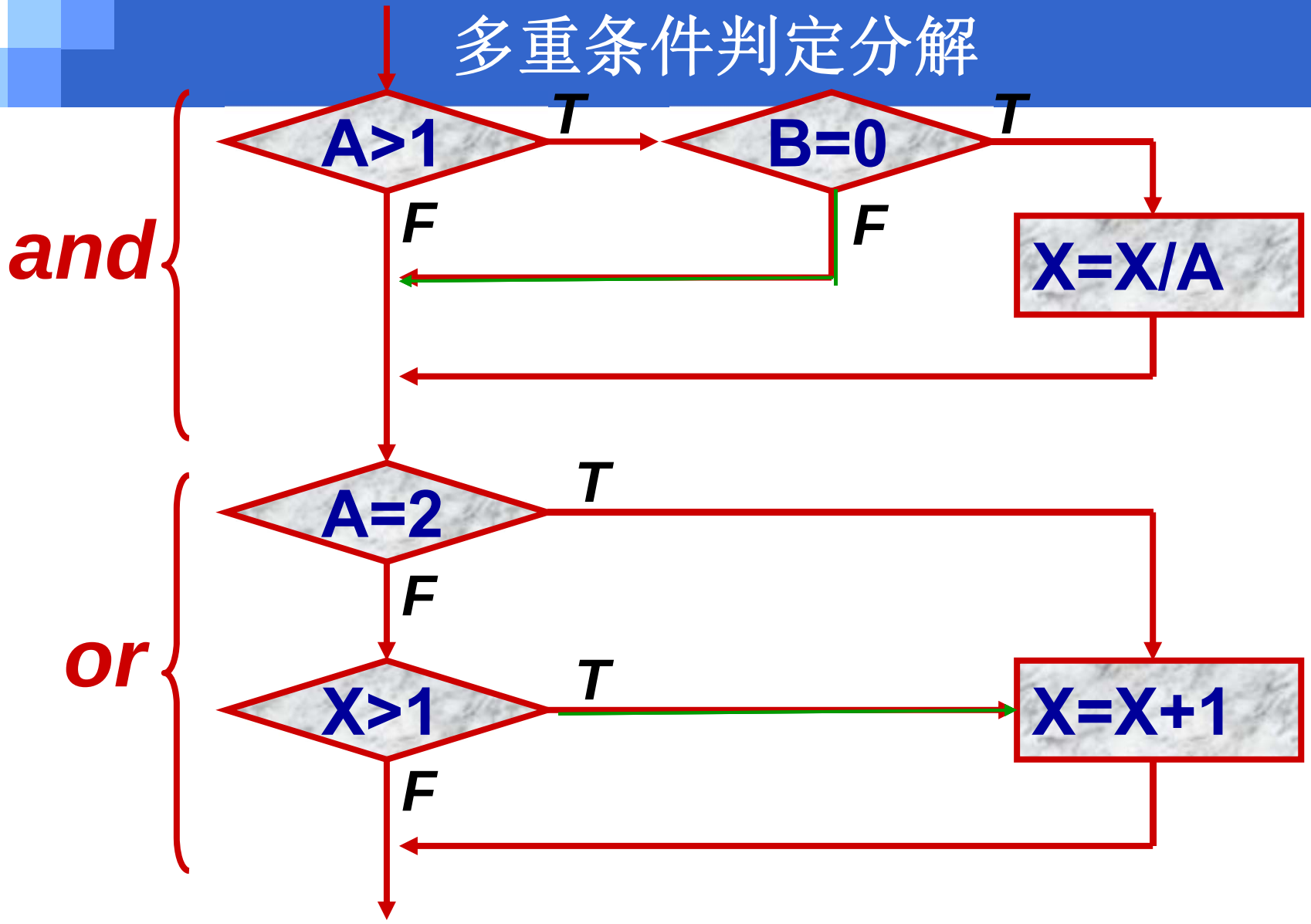
ace ∧ abd 或： **acd ∧ abe**

选择用例：

[(2, 0, 4), (2, 0, 3)] (ace)

[(1, 1, 1), (1, 1, 1)] (abd)

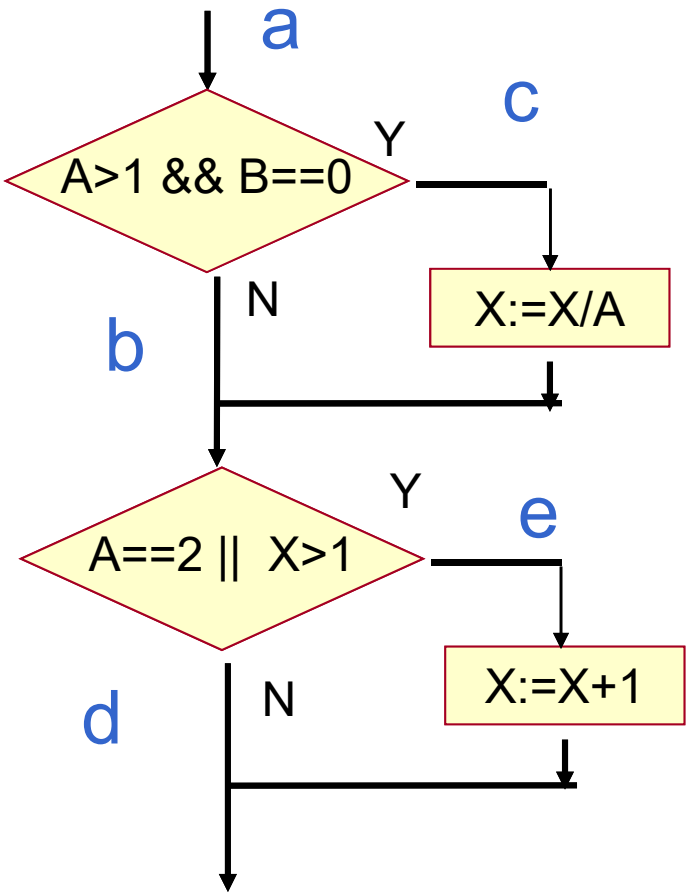
多重条件判定分解



编译系统下的执行情况：部分路径未被执行。

5）条件组合覆盖

- 设计若干个测试用例，运行被测程序，使得每个判定中条件的各种可能组合都至少出现一次。



满足以下覆盖情况：

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**

选择用例：

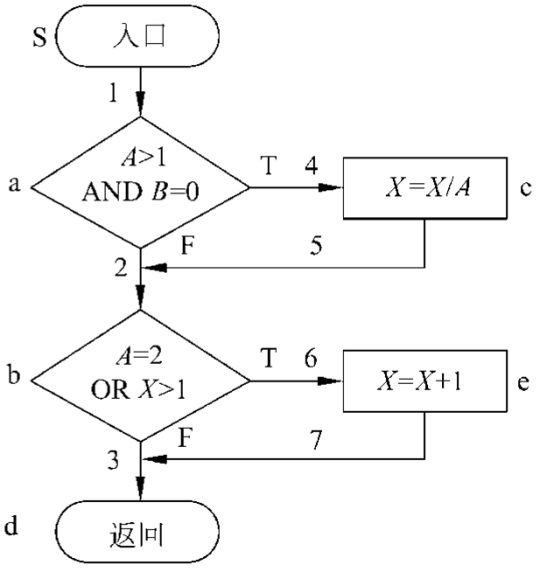
[(2, 0, 4), (2, 0, 3)]　①⑤

[(2, 1, 1), (2, 1, 2)]　②⑥

[(1, 0, 3), (1, 0, 4)]　③⑦

[(1, 1, 1), (1, 1, 1)]　④⑧

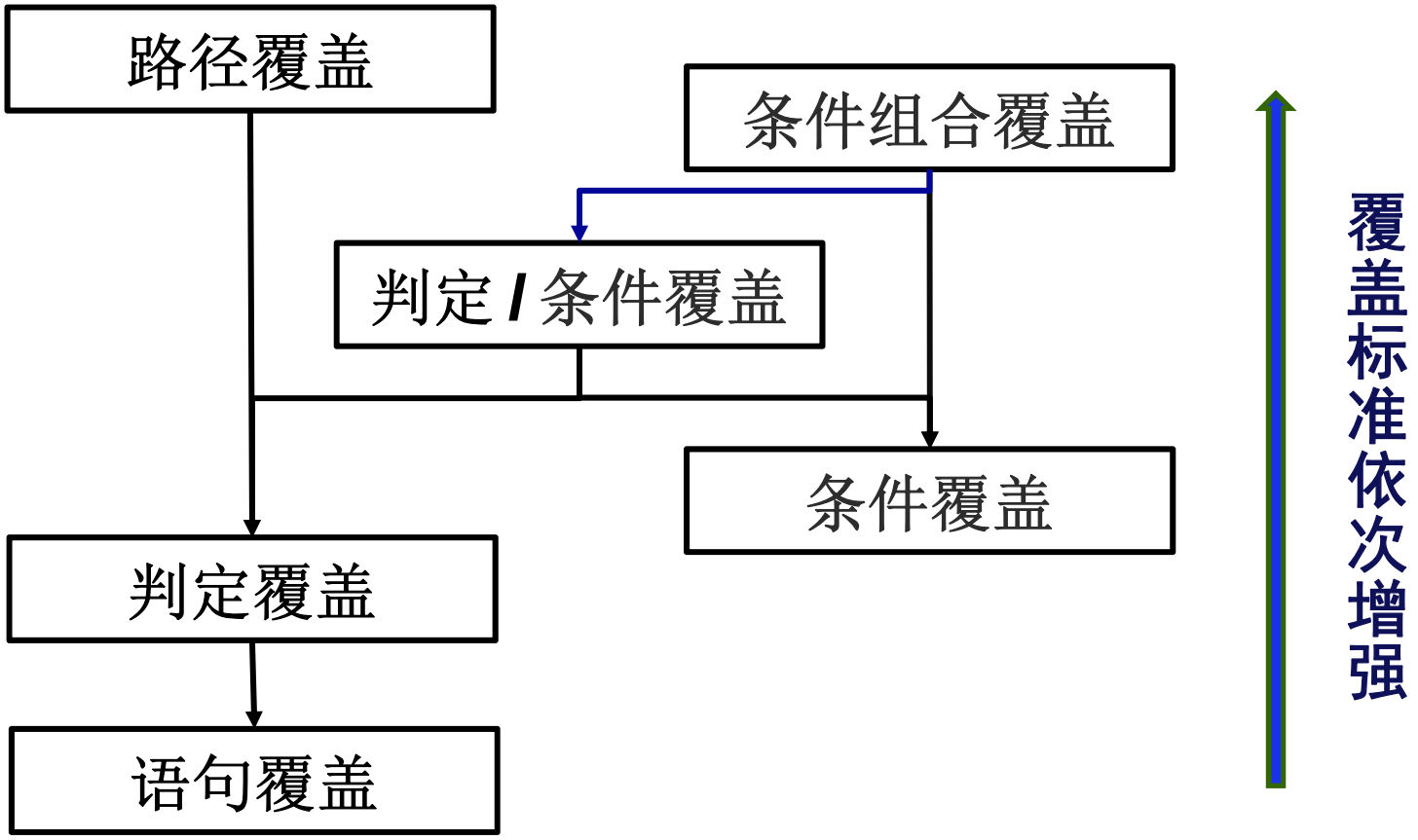
6）路径覆盖



- 路径覆盖就是设计足够的测试用例，覆盖程序中所有可能的路径。

测 试 用 例	通过路径	覆盖条件
【(2, 0, 4), (2, 0, 3)】	ace (L1)	$T_1T_2T_3T_4$
【(1, 1, 1), (1, 1, 1)】	abd (L2)	$\overline{T_1}T_2T_3T_4$
【(1, 1, 2), (1, 1, 3)】	abe (L3)	$\overline{T_1}\overline{T_2}\overline{T_3}T_4$
【(3, 0, 3), (3, 0, 1)】	acd (L4)	$\overline{T_1}\overline{T_2}T_3\overline{T_4}$

覆盖标准之间的关系



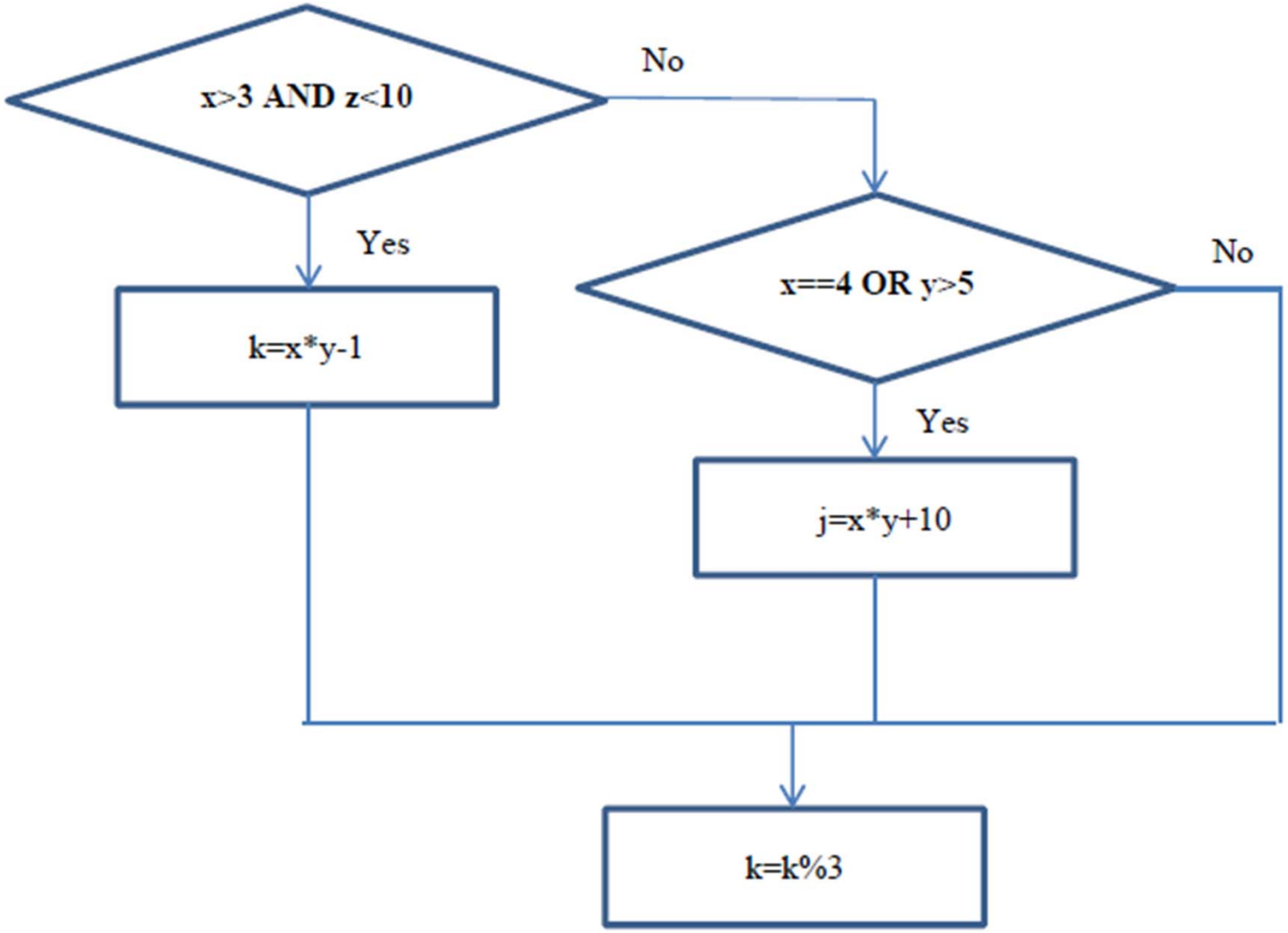
白盒测试：课题练习

```
void MyFunc(int x, int y, int z, int &j, int k)
{
    j=k=0;
    if( (x>3) && (z<10))  k=x*y-1;
    else if ((x==4) || (y>5))  j=x*y+10;
    k=k%3;
}
```

1）绘制流程图；

2）满足“条件组合覆盖”标准的最小测试用例集。

流程图



判定覆盖

- 判定组合至少出现1 次。
- ① x>3, z<10　　T
- ② x<=3 或 z>=10　　F
- ③ x==4 或y>5　　T
- ④ x!=4, y<=5　　F

合并后最小测试用例条件，例如：

①③ x==4 且 z<10　　①④ x>4且y<=5且z<10

②④ x<=3 且 y<=5　　②③ x<=3或 y>5或z>=10

条件覆盖

- 每个条件的可能取值至少执行一次
- T1： x>3　　T2： z<10
- T3： x==4　T4： y>5
- 合并后最小测试用例条件，例如：

$T_1T_2T_3T_4$ ： **x==4 且 z<10且 y>5**

$\overline{T_1T_2T_3T_4}$ ： **x<=3 且 z>=10 且 y<=5**

判定/条件覆盖

- 同时满足判定覆盖和条件覆盖。
- ① x>3, z<10　　② x<=3 或 z>=10
- ③ x==4 或y>5　　④ x!=4, y<=5
- T1：　x>3　T2：　z<10　T3：　x==4　T4：　y>5
- 合并后最小测试用例条件，例如：

① ③，$T_1T_2T_3T_4$ ，　x==4 且 z<10且y>5

$T_1T_2T_3 \ldots T_1T_2T_4$

② ④，$\overline{T_1T_2T_3T_4}$ ，　x<=3 且 z>=10 且 y<=5

$\overline{T_1T_3T_4} \ldots \overline{T_2T_3T_4}$

条件组合覆盖

- 条件的8种组合至少出现1次。

① x>3, z<10　　⑤ x==4, y>5

② x>3, z>=10　　⑥ x==4, y<=5

③ x<=3, z<10　　⑦ x!=4,y>5

④ x<=3, z>=10　　⑧ x!=4, y<=5

合并后最小测试用例条件，例如：

①⑤、②⑥、③⑦、④⑧

（2）程序插桩技术

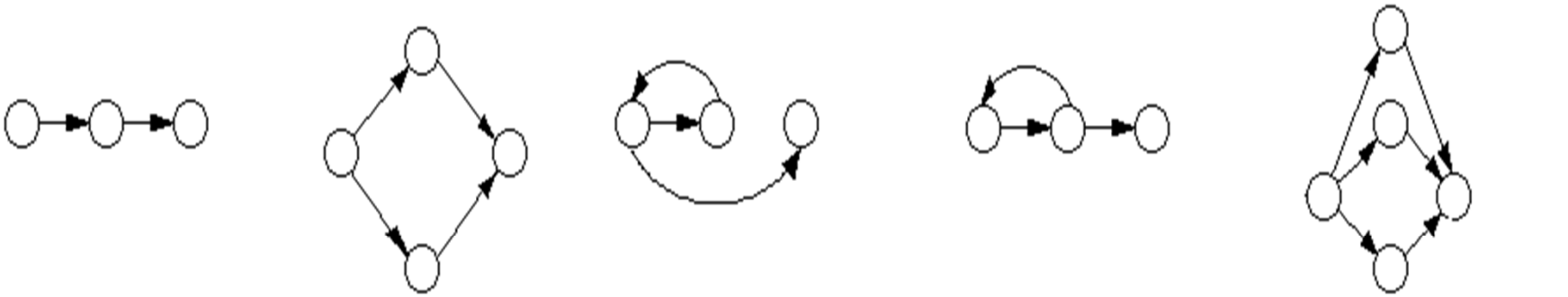
- 通过向被测程序中插入操作（语句）来达到测试的目的。
- 那些被插人的语句称为探测器或探针。
- 目的：记录程序执行过程中发生的一些重要事件，如语句执行次数、变量值的变化情况、指针的改变等。
- 需要考虑的问题：
  - 探测哪些信息？
  - 在什么位置设置探测点？
  - 需要设置多少个探测点？

（3）基本路径法

- 基本路径测试(McCabe, 1976) 确保代码模块中所有的独立路径被测试到。
  - 独立路径是指代码中任意引入至少一个新的要处理的语句集合或一个新条件的路径(Pressman, 2001)。
- 基本路径测试方法把覆盖的路径数压缩到一定限度内，程序中的循环体最多只执行一次。
- 它是在程序控制流图的基础上，分析控制构造的环路复杂性，导出基本可执行路径集合，设计测试用例的方法。
- 设计出的测试用例要保证在测试中，程序的每一个可执行语句至少要执行一次。
- 基本路径测试提供了一个测试用例的下界

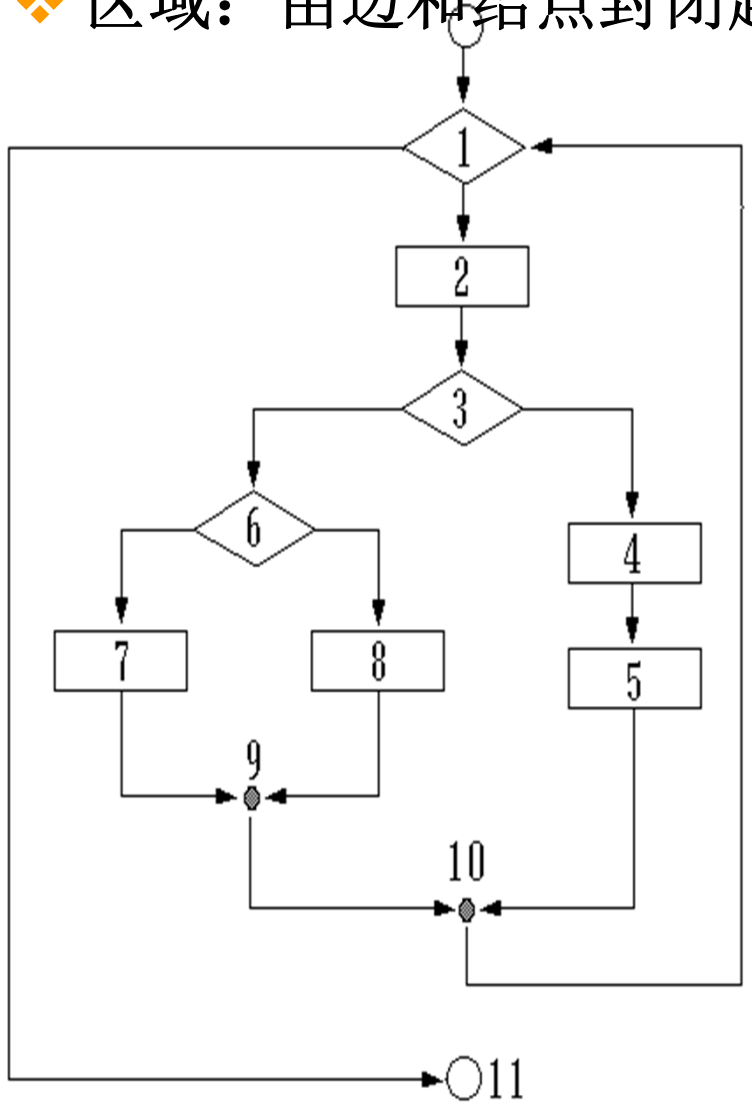
1） 程序的控制流图

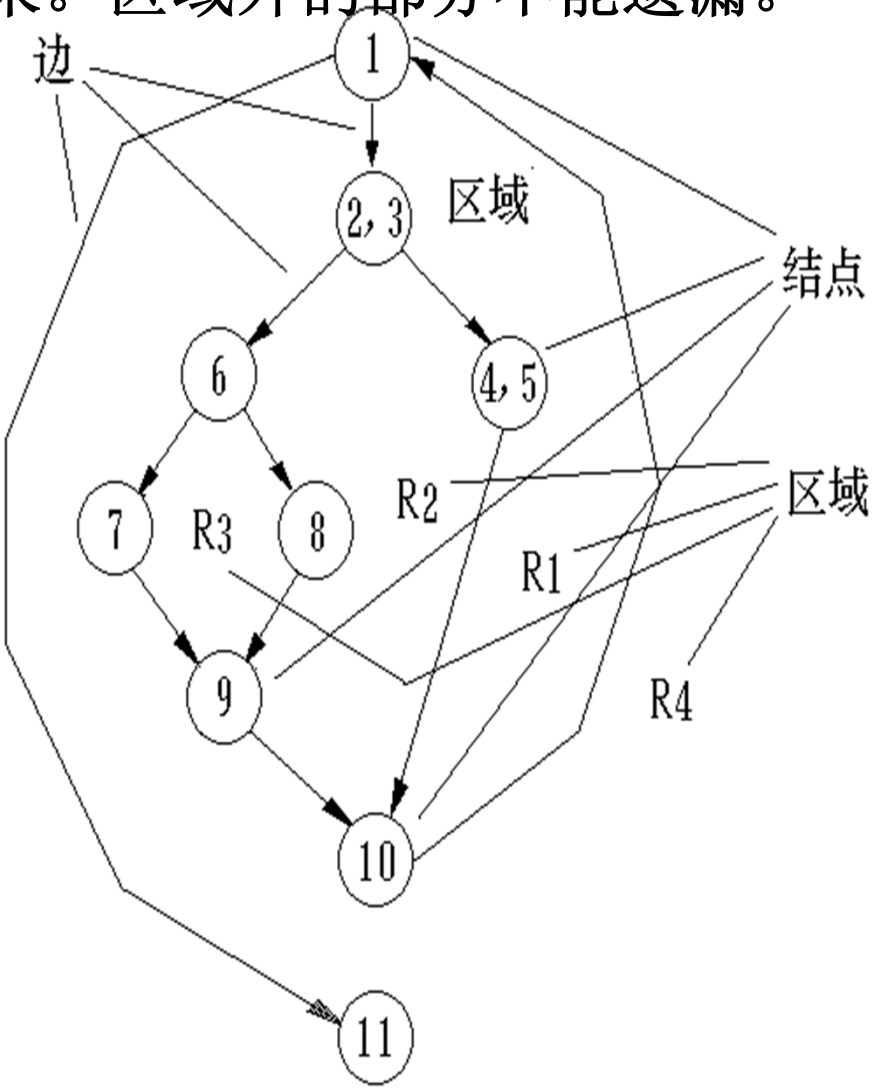
- 符号○为控制流图的一个结点，表示一个或多个无分支的PDL语句或源程序语句。箭头为边，表示控制流的方向。



程序流程图　　控制流程图

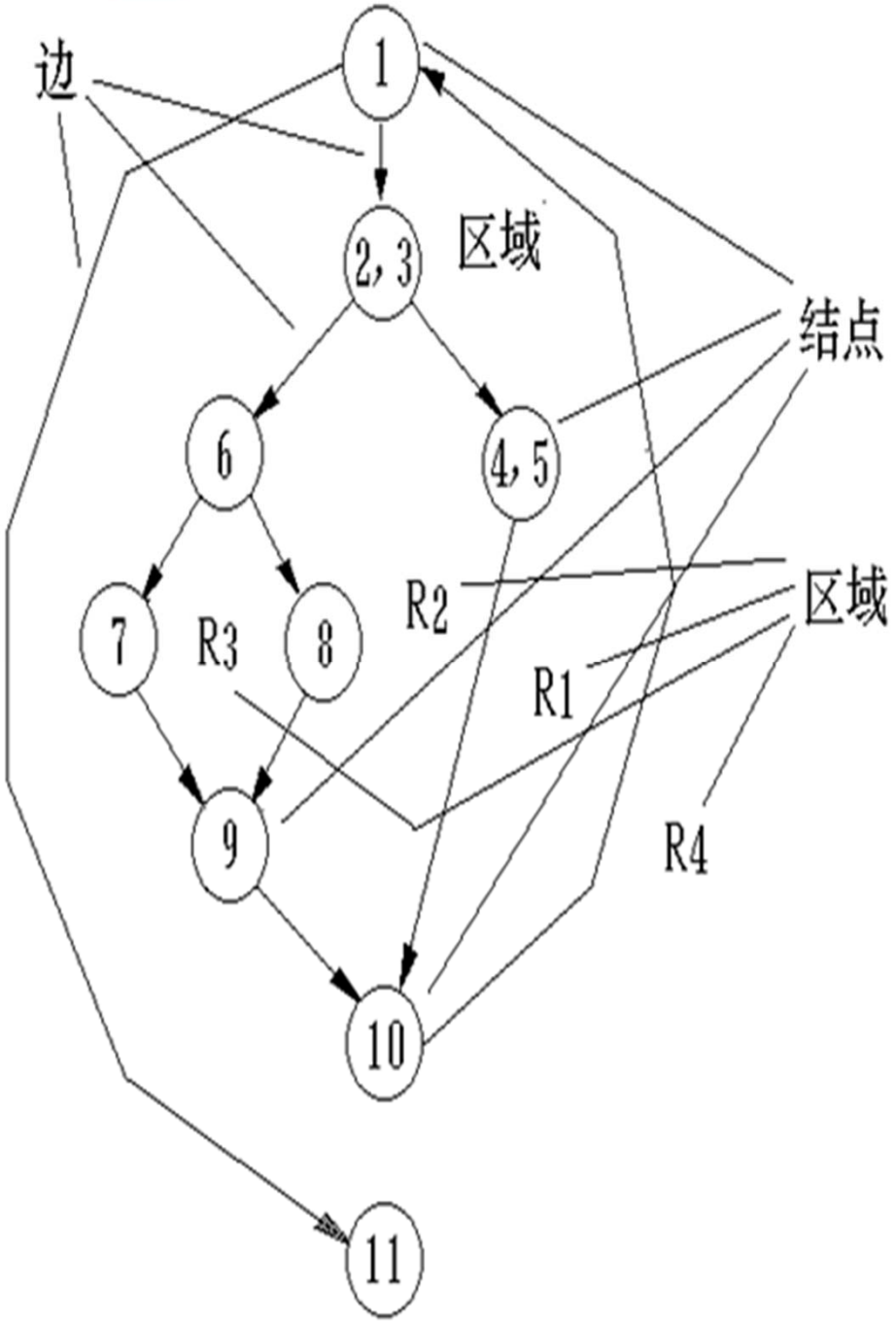
- 区域：由边和结点封闭起来。区域外的部分不能遗漏。





2） 程序环路复杂性

- 程序的环路复杂性：给出了程序基本路径集中的独立路径条数，确保程序中每个可执行语句至少执行一次。
- 从控制流图来看，一条独立路径是至少包含有一条在其它独立路径中从未有过的边的路径。



独立路径

- **path1：1 - 11**
  path2：1 - 2 - 3 - 4 - 5 - 10 - 1 - 11
  path3：1 - 2 - 3 - 6 - 8 - 9 - 10 - 1 - 11
  path4：1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11
- **路径 path1，path2，path3，path4组成了控制流图的一个基本路径集。**

程序环路（圈）复杂性的度量

- 常用的三种方法：
- 1、流图中区域的数量对应于环型的复杂性；
- 2、给定流图G的圈复杂度V(G)，定义为：

$$V(G) = E-N+2$$

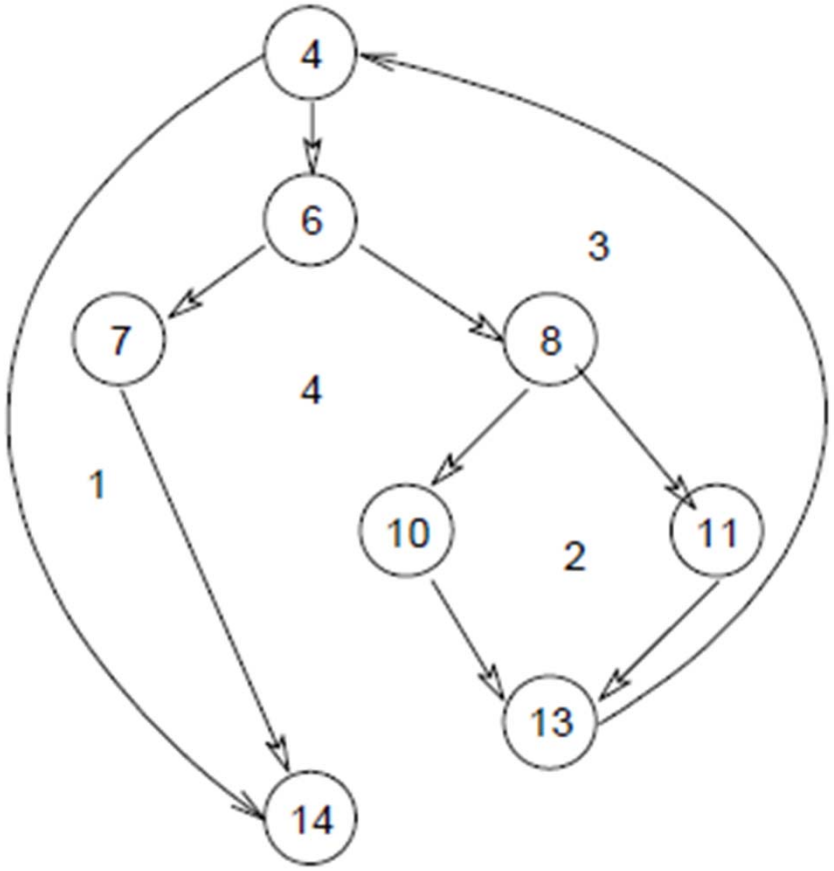
E是流图中边的数量，N是流图中结点的数量

- 3、给定流图G的圈复杂度V(G)，定义为：

$$V(G)=P+1,$$

P是流图G中判定结点的数量。

环路（圈）复杂性的度量示例



- 1：4个区域
- 2：V(G) = 10 – 8 + 2 = 4
- 3：V(G) = 3 + 1 = 4

3）导出测试用例

- 导出测试用例，确保基本路径集中的每一条路径的执行。
- 根据判断结点给出的条件，选择适当的数据以保证某一条路径可以被测试到 — 用逻辑覆盖方法。
- 每个测试用例执行之后，与预期结果进行比较。如果所有测试用例都执行完毕，则可以确信程序中所有的可执行语句至少被执行了一次。

（4）符号测试

- 基本思想：允许程序的输入不仅可以是数值数据，也可以包括符号值。
- 符号：可以是符号变量，也可以是包含这些符号变量的一个表达式。
- 例如：如果原来测试某程序时要从输入数据X的取值范围1~200中选取一个进行数值计算，现在可以用X1作为输入数据，代入程序进行代数运算，这样所得的结果是含有X1的代数表达式。
- 在执行被测程序的过程中，符号的计算就代替了普通测试执行中对测试用例的数值计算，所得的结果是符号公式或符号谓词。

（5）错误驱动测试

- 现实的解决办法是将搜索错误的范围尽可能地缩小，以利于测试某类错误是否存在。基于这一思想，出现了错误驱动测试方法。
- 错误驱动测试方法：可以把目标集中在对软件危害最大的可能错误上，而暂时忽略对软件危害较小的可能错误。可以取得较高的测试效率，并能降低测试的成本。
- 程序变异方法是一种错误驱动测试
  - 程序强变异
  - 程序弱变异

程序变异

- DeMillo认为：当程序被开发并经过简单测试后，残留在程序中的错误不再是很严重的错误，而是一些难以发现的小错误。
- 例如：遗漏了某个操作、分支谓词规定的边界有位移、运行时的内存错误等。即使是一些较复杂的错误，也可能由这些简单错误组合而成。
- 程序变异的目标：查出这些简单的错误及其组合错误。
- 程序P的变异因子m(P)也是一个程序，它是对P进行微小改动而得到的。因而，也可以说m（P）是P的一个变换。如果程序P是正确的，则m（P）是一个几乎正确的程序；如果P不正确，则P的某一个变异因子m(P)也可能是正确的。

程序变异的方法：替换

◆ 一些常用的变异操作如下：

- 常量之间的替换；
- 数组之间的替换；
- 将常量替换为数组分量；
- 将数组分量替换成常量；
- 同维数的数组名之间的替换；
- 算术运算符之间的替换；
- 关系运算符之间的替换；
- 逻辑运算符之间的替换；
- 插入绝对值符号；
- 插入单目运算符；
- 语句分解；
- 语句删除；
- 循环终止条件变换。

2、黑盒测试法

- 黑盒测试，力图发现下述类型的错误：
  - ①功能不正确或遗漏了功能；
  - ②界面错误；
  - ③数据结构错误或外部数据库访问错误；
  - ④性能错误；
  - ⑤初始化和终止错误。

黑盒测试方案的设计

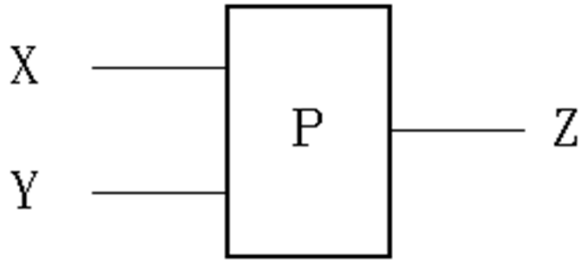
- 应考虑如下问题：
  - 怎样测试功能的有效性？
  - 哪些类型的输入可构成好的测试用例？
  - 系统是否对特定的输入值特别敏感？
  - 怎样划定数据类的边界？
  - 系统能够承受什么样的数据率和数据量？
  - 数据的特定组合将对系统运行产生什么影响？

黑盒测试的误区

- 用黑盒测试发现程序中的错误，必须在所有可能的输入条件和输出条件中确定测试数据，来检查程序是否都能产生正确的输出——但这是不可能的。

- 假设一个程序*P*有输入量*X*和*Y*及输出量*Z*。在字长为32位的计算机上运行。若*X*、*Y*取整数，按黑盒方法进行穷举测试：
- 可能采用的测试数据组：

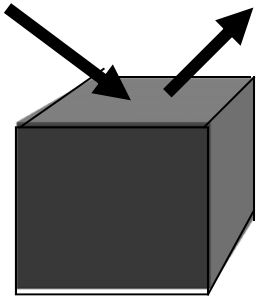
$$2^{32} \times 2^{32} = 2^{64}$$



- 如果测试一组数据需要1毫秒，一年工作*365*× *24*小时，完成所有测试需5亿年。

黑盒测试的方法

- 不考虑程序的内部结构与特性，只根据程序功能或程序的外部特性设计测试用例。



- 等价划分法
- 边值分析法
- 错误推测法
- 因果图法

实际案例

- 手机号码输入框：**153XXXX6360**

- 测试用例：
- **1**：输入英文空格：
- **2**：输入中文空格：半角输入、全角输入
- **3**：输入特殊字符
- **4**：输入超长**11**位字符
- **5**：输入英文字符
- **6**：输入中文字符

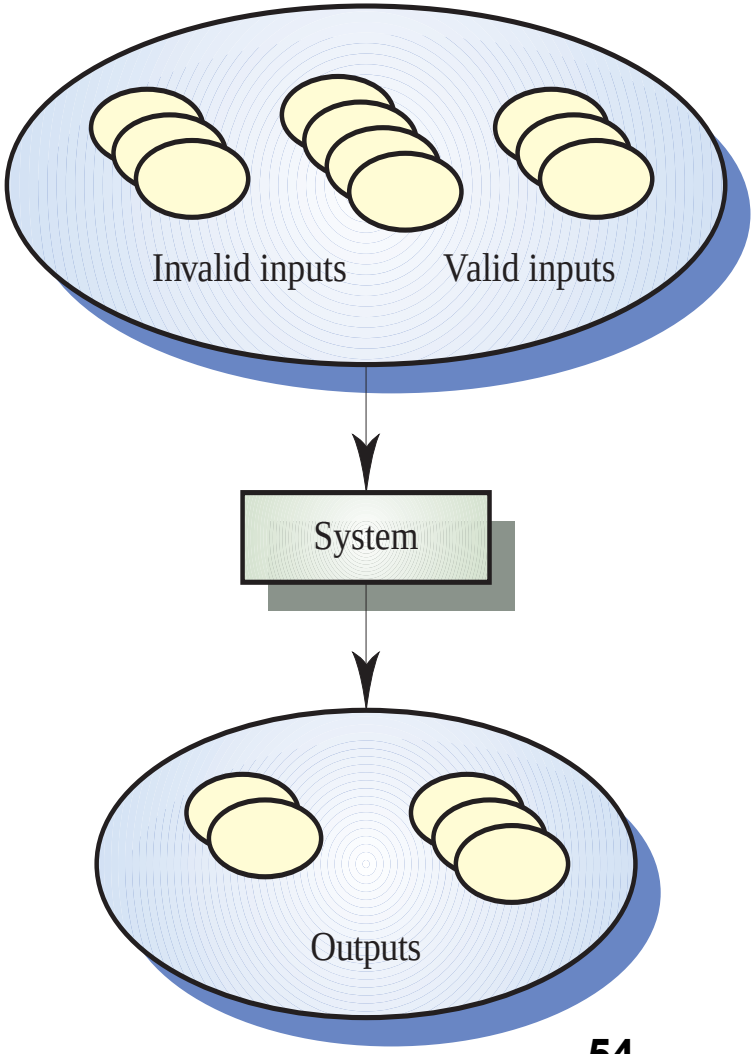
1、等价划分法

❖ 等价划分把程序的输入域划分成若干个数据类，据此导出测试用例。

❖ 一个理想的测试用例能独自发现一类错误。

❖等价类分为：

- 有效等价类：对于程序的规格说明，是合理的、有意义的输入数据构成的集合。
- 无效等价类：对于程序的规格说明，是不合理的、没有意义的输入数据构成的集合。



划分等价类的标准

- 有效等价类：可以检验程序是否实现预先规定的功能和性能。
- 无效等价类：用来检查程序中功能和性能的实现是否不符合规格说明要求。
- 等价类划分的标准
  - 完备测试、避免冗余
    - 划分为互不相交的一组子集
    - 集的并是整个集合

等价划分步骤

❖① 划分“等价类”

- 应按照输入条件（如输入值的范围，值的个数，值的集合，输入条件必须如何）划分为有效等价类和无效等价类。
- 例如：每个学生可选修1-3门课程
  - 可以划分一个有效等价类：选修1-3门课程。
  - 可以划分两个无效等价类：未选修课，选修课超过3门。

❖② 选择测试用例

- 为每个等价类编号；
- 使一个测试用例尽可能覆盖多个有效等价类；
- 特别要注意：一个测试用例只能覆盖一个无效等价类。

等价类划分的启发规则

❖ 1、如果规定了输入值的范围，则可划分出一个有效的等价类和两个无效的等价类；

例如：… … 序号值可以从 1到999 … …

一个有效等价类：1≤ 序号值 ≤999

两个无效等价类：序号值 <1

序号值 >999

❖2、如果规定了输入数据的个数，则类似地可划分出一个有效的等价类和两个无效的等价类；

等价类划分的启发规则（续1）

- 3、如果输入条件规定了输入值的集合，或者是规定了“必须如何”的条件，这时可划分出一个有效等价类和一个无效等价类。

> 例如：在 C 语言中对变量标识符规定为 “以字母打头的 …… 串”。 所有以字母打头的构成为有效等价类； 而不在此集合内(不以字母打头)归于无效等价。

- 4、如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。

等价类划分的启发规则（续2）

❖5、如果规定了输入数据的一组值，而且程序对不同输入值分别进行处理，则每个允许的输入值是一个有效的等价类，此外还有一个无效的等价类（它是所有不允许输入值的集合）。

例如：在教师分房方案中规定对教授、副教授、讲师和助教分别计算分数，做相应的处理。因此可以确定4个有效等价类为教授、副教授、讲师和助教，以及 1个无效等价类，它应是所有不符合以上身份的人员的输入值的集合。

❖6、如果规定了输入数据为整型，则可以划分出正整数、零和负整数三个有效类。

等价类划分的启发规则（续3）

❖7、如果规定了输入数据必须遵循的规则，则可以划分出一个有效的等价类和若干个无效的等价类。

例如：在C 语言中规定了“一个语句必须以分号‘；’作为结束”，这时，可以确定一个有效等价类，以“；”结束，而若干个无效等价类应以“：，、”等结束。

❖8、如果程序的处理对象是表格，则应该使用空表以及含一项或多项的表。

等价类划分举例

对**FORTRAN**编译系统中的**DIMENSION**语句进行测试。

语句格式为：**DIMENSION** ad[,ad] …　　ad为数组描述符，

形式为 n（d[,] …　其中：n一数组名，字母打头的字母数字串，长6。

D为界偶（1-7个）：[ld:]nd　ld 和 nd 的值为1-65535，　ld缺省为1。

输入条件	合理的等价类	不合理的等价类
数组描述的个数	1个（1）、多于1个（2）	没有数组描述（3）
数组名的字符数	1—6个（4）	0（5），>6（6）
数组名	有字母（7）有数字（8）	有其他字符（9）
数组名的第1个字符为字母	是（10）	不是（11）
维数	1—7（12）	0（13），　>7（14）
上界	常数（15）	数组元素名（16

40 个等价类

2、边界值分析

❖基本思想：选择等价类的边缘值作为测试用例，让每个等价类的边界都得到测试，选择测试用例既考虑输入亦考虑输出。

❖分析步骤：

- 先划分等价类。
- 选择测试用例，测试等价类边界。

❖边界选择原则：

- 按照输入值范围的边界，如[-32768,32767] 16位整数
- 按照输入/输出值个数的边界。
- 输出值域的边界。
- 输入/输出有序集的边界，如a[0]、a[n-1]

边界值分析法举例

■ A、按照输入值范围的边界：刚刚达到或超越范围

例如：输入值的范围是-1.0至1.0，

则可选择用例: –1.0、1.0、-1.001、1.001。

■ B、按照输入/输出值个数的边界：最大、最小、+/- 1

例如：输入文件可有1-255个记录，

则 设计用例：文件的记录数为 0个、1个、255个、256个。

■ C、输出值域的边界。

例如：检索文献摘要，最多4篇。

设计用例：可检索0篇、1篇、4篇，和5篇（错误）。

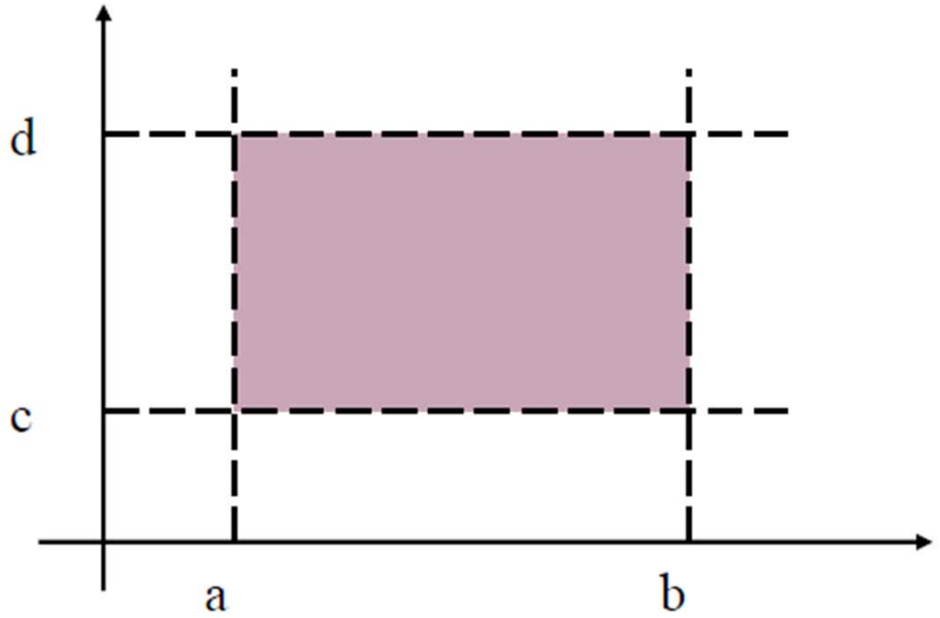
■ D、输入/输出有序集（如顺序文件、线性表）的边界。

应选择第一个元素和最后一个元素。

边界值分析示例

$a \le x_1 \le b$

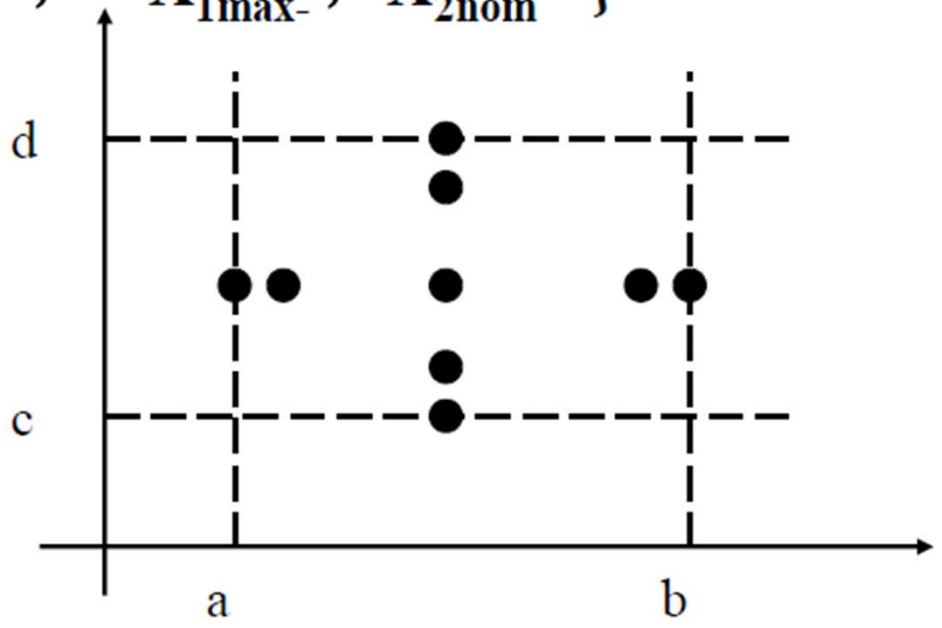
$c \le x_2 \le d$



基本思想：取输入变量在最小值、略高于最小值、正常值、略低于最大值、最大值。

边界值分析示例（续）

$\{<X_{1nom}, X_{2min}>, <X_{1nom}, X_{2min+}>, <X_{1nom}, X_{2nom}>,$
$<X_{1nom}, X_{2max}>, <X_{1nom}, X_{2max-}>,$
$<X_{1min}, X_{2nom}>, <X_{1min+}, X_{2nom}>, <X_{1nom}, X_{2nom}>,$
$<X_{1max}, X_{2nom}>, <X_{1max-}, X_{2nom}>\}$



3、错误推测法

❖错误推测法：靠经验和直觉推测程序中可能存在的各种错误，从而有针对性地编写检查这些错误的例子。

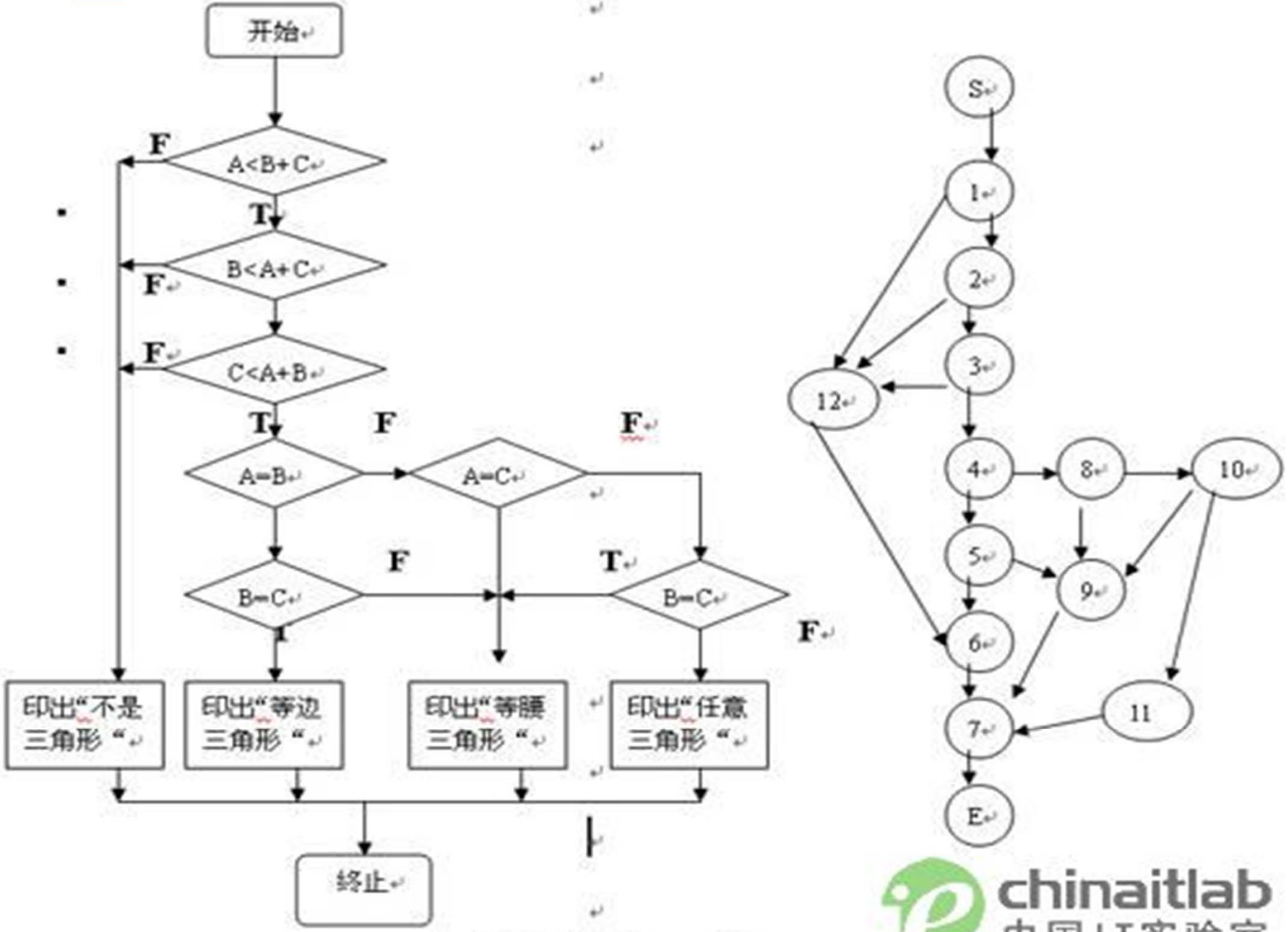
❖基本思想：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。

❖例如：线性表排序算法

- 输入的线性表为空表；
- 表中只含有一个元素；
- 输入表中所有元素已排好序；
- 输入表已按逆序排好；
- 输入表中部分或全部元素相同。

黑盒测试用例设计案例

【例】假设现有以下的三角形分类程序。该程序的功能是，读入代表三角形边长的3个整数，判定它们能否组成三角形。如果能够，则输出三角形是等边、等腰或任意三角形的分类信息。图1显示了该程序的流程图和程序图。为以上的三角形分类程序设计一组测试用例。



三角形程序的流程图和程序图

【解】

■第一步：确定测试策略。在本例中，对被测程序的功能有明确的要求，即：

（1）判断能否组成三角形；

（2）识别等边三角形；

（3）识别等腰三角形；

（4）识别任意三角形。

因此可首先用黑盒法设计测试用例，然后用白盒法验证其完整性，必要时再进行补充。

■ 第二步：根据本例的实际情况，在黑盒法中首先可用等价分类法划分输入的等价类，然后用边界值分析法和错误推测法作补充。

- 等价分类法：
- 有效等价类

输入3个正整数：

(1)3数相等

(2)3数中有2个数相等，比如AB相等

(3)3数中有2个数相等，比如BC相等

(4)3数中有2个数相等，比如AC相等

(5)3数均不相等

(6)2数之和不大于第3数，比如最大数是A

(7)2数之和不大于第3数，比如最大数是B

(8)2数之和不大于第3数，比如最大数是C

无效等价类：

(9)含有零数据

(10)含有负整数

(11)少于3个整数

(12)含有非整数

(13)含有非数字符

- 边界值法：

(14)2数之和等于第3数

- 错误推测法：

(15)输入3个零

(16)输入3个负数

■ 第三步：提出一组初步的测试用例，如下表所示：

序号	测试内容	测试数据			期望结果
		A	B	C	
1	等边	5,5,5			等边三角形
2	等腰	4,4,5	5,5,4	4,5,4	等腰三角形
3	任意	3,4,5			任意三角形
4	非三角形	9,4,4	4,9,4	4,4,9	不是三角形
5	退化三角形	8,4,4	4,8,4	4,4,8	
6	零数据	0,4,5	4,0,5	4,5,0	
7		0,0,0			
8	负数据	-3,4,5	3,-4,5	3,4,-5	运行出错
9		-3,-4,-5			
10	遗漏数据	3,4			
11	非整数	3.3,4,5			
12	非数字符	A,4,5			

■ 第四步：用白盒法验证第三步产生的测试用例的充分性。结果表明，上表中的前8个测试用例，已能满足对被测程序图的完全覆盖，不需要再补充其他的测试用例。

4、因果图

- 把输入条件视为“因”，把输出条件视为“果”，
- 将黑盒看成是从因到果的网络图
- 采用逻辑图的形式来表达功能说明书中输入条件的各种组合与输出的关系。
- 根据这种关系可选择高效的测试用例。
- 因果图是一种形式化语言，是一种组合逻辑网络图。

因果图的适用范围

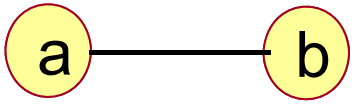
- 如果在测试时必须考虑输入条件的各种组合，可使用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来设计测试用例，这就需要利用因果图。
  - if-then-else逻辑很突出
  - 输入变量之间存在逻辑关系
  - 涉及输入变量子集的计算
  - 输入与输出之间存在因果关系
  - 很高的圈(McCabe)复杂度
- 因果图方法最终生成的就是判定表(或决策表）。它适合于检查程序输入条件的各种组合情况。

因果图的基本符号

- 0 – 表示“不出现”　　1 – 表示“出现”

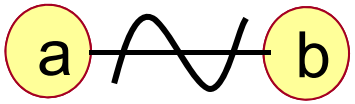
- 恒等

  若a为1，则b为1，否则b为0。



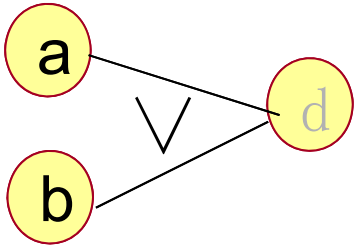
- “非”函数

  若a为1，则b为0，否则b为1。



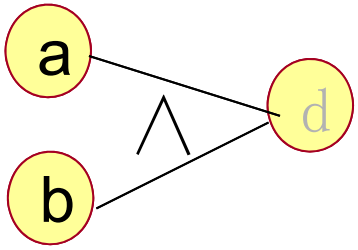
- “或”函数

  若a或b为1，则d为1，否则d为0。



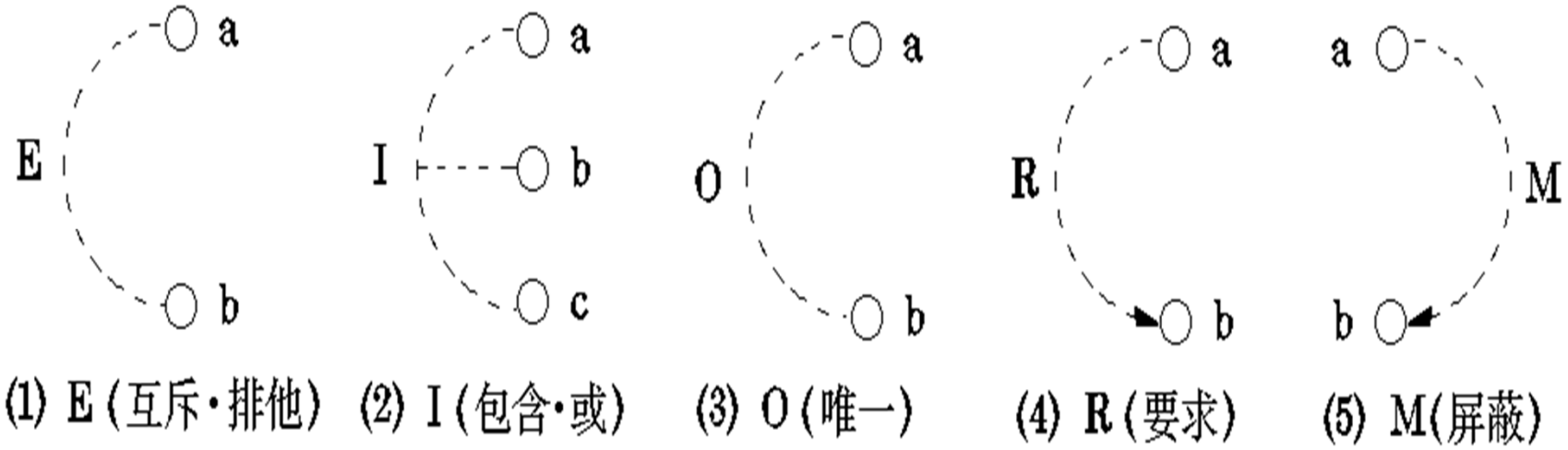
- “与”函数

  若a与b同为1，则d为1，否则d为0。



- 表示约束条件的符号

  为了表示原因与原因之间，结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。



(1) E(互斥·排他) (2) I(包含·或) (3) O(唯一) (4) R(要求) (5) M(屏蔽)

因果图法的步骤

- (1) 分析软件规格说明描述，找出
  - 哪些是原因（即输入条件或输入条件的等价类），
  - 哪些是结果（即输出条件），
  - 给每个原因和结果赋予一个标识符。
- (2) 分析软件规格说明描述中的语义，找出
  - 原因与结果之间，原因与原因之间对应的是什么关系?
  - 根据这些关系，画出因果图。

因果图法的步骤（续1）

- (3) 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。
- (4) 把因果图转换成判定表。
- (5) 把判定表的每一列拿出来作为依据，设计测试用例。

因果图法应用示例

- 例如，有一个处理单价为5角钱的饮料的自动售货机软件测试用例的设计。其规格说明如下：
  1）若投入5角钱或1元钱的硬币，押下〖橙汁〗或〖啤酒〗的按钮，则相应的饮料就送出来。
  2）若售货机没有零钱找，则一个显示〖零钱找完〗的红灯亮，这时在投入1元硬币并押下按钮后，饮料不送出来而且1元硬币也退出来；若有零钱找，则显示〖零钱找完〗的红灯灭，在送出饮料的同时退还5角硬币。

(1) 分析这一段说明，列出原因和结果

原因: 1. 售货机有零钱找

2. 投入1元硬币

3. 投入5角硬币

4. 押下橙汁按钮

5. 押下啤酒按钮

建立中间结点，表示处理中间状态

11. 投入1元硬币且押下饮料按钮

12. 押下〖橙汁〗或〖啤酒〗的按钮

13. 应当找5角零钱并且售货机有零钱找

14. 钱已付清

结果： 21. 售货机〖零钱找完〗灯亮

22. 退还1元硬币

23. 退还5角硬币

24. 送出橙汁饮料

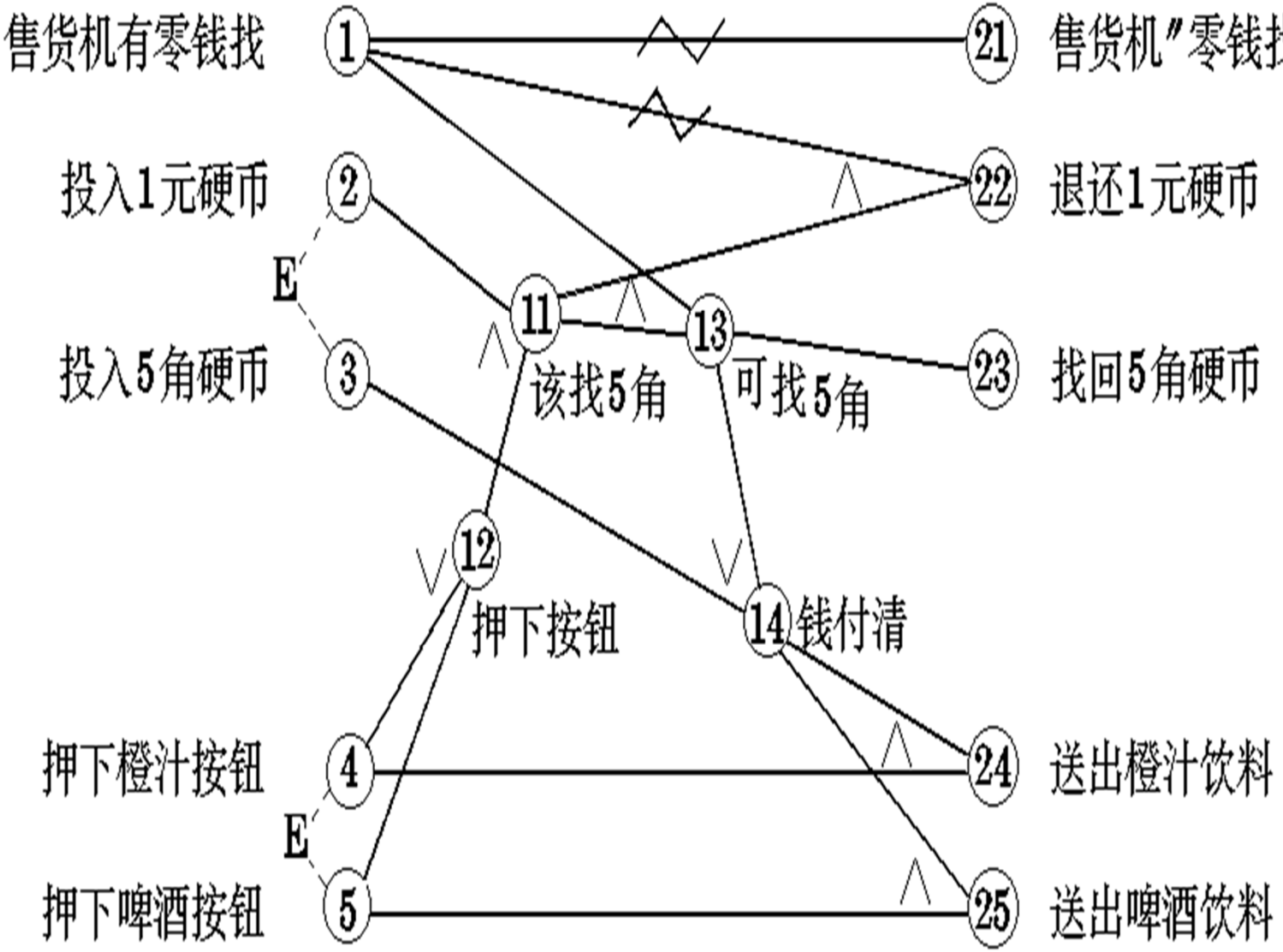
25. 送出啤酒饮料

(2) 画出因果图。所有原因结点列在左边，所有结果结点列在右边。

(3) 由于 2 与 3，4 与 5 不能同时发生，分别加上约束条件E。

(4) 因果图

(5) 转换成判定表


售货机有零钱找
1
21
售货机"零钱找完"灯亮
投入1元硬币
2
22
退还1元硬币
E
11
13
投入5角硬币
3
该找5角
可找5角
23
找回5角硬币
12
押下按钮
14
钱付清
押下橙汁按钮
4
24
送出橙汁饮料
E
押下啤酒按钮
5
25
送出啤酒饮料

因果图

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

其他黑盒测试方法

- Behave Like a Dumb User
- Look for Bugs Where You've Already Found Them
  - the more bugs you find, the more bugs there are
  - Many programmers tend to fix only the specific bug you report
- Think like a Hacker
- Follow Experience, Intuition, and Hunches

黑盒测试：课堂练习

- 假设一个软件系统的用户日期输入，限定在1990年1月~2049年12月，并规定日期由6位数字字符组成（前4位表示年，后2位表示月）。
- 请使用等价类划分法进行黑盒测试，每一类注明等价类内容并设计相应的测试用例，来测试该系统的 "日期检查功能"。
- 测试用例格式：编号、测试数据、 期望结果、覆盖的等价类编号
- 提示：日期的类型及长度

  年份范围

  月份范围

课堂练习（续1）

输入等价类	有效等价类	无效等价类
日期的类型及长度	(1)	(4) (5) (6)
年份范围	(2)	(7) (8)
月份范围	(3)	(9) (10)

答案

输入等价类	有效等价类	无效等价类
日期的类型及长度	（1）6位数字字符	（4）有非数字字符 （5）少于6位数字字符 （6）多于6位数字字符
年份范围	（2）在1990~2049之间	（7）小于1990 （8）大于2049
月份范围	（3）在01~12之间	（9）小于00 （10）大于12

测试策略小结

测试策略

1、在任何情况下都应该使用边界值分析的方法。

2、必要时用等价类划分法补充测试方案。

3、必要时再用错误猜测法补充测试方案。

4、对照程序逻辑，检查已经设计出来的测试方案。可以根据对程序可靠性的要求采用不同的逻辑覆盖标准，若现有测试方案的逻辑程度没有达到要求的覆盖标准，则应再补充一些测试方案。

3、测试工具简介

- 白盒测试工具
  - 动态测试主要采用“插桩”的方式，即向代码生成的可执行文件中插入一些监测代码，运行框架程序，统计程序运行时的数据，可以针对所有类的成员函数进行测试，也可以只针对类的公共接口函数进行测试。
  - 商业性的白盒测试工具，比较有代表性的如：compuware公司的Numega系列工具和ParaSoft的JavaSolution以及C/C++ Solution系列。
  - 非商业性的白盒测试工具，主要以Xunit系列为代表的测试框架工具。
  - 集成环境的工具

- 黑盒测试工具，
  - 功能测试的工具：
  - Rational公司的TeamTest、Robot，
  - Compuware公司的QACenter，
  - 性能测试的工具：
  - LoadRunner，
  - Radview公司的WebLoad、
  - Microsoft公司的WebStress等工具。

本节小结

- 黑盒测试（功能测试，数据驱动测试）
  - 等价类划分
  - 边界值分析
  - …
- 白盒测试（结构测试，逻辑驱动测试）
  - 逻辑覆盖法
  - …