

### 3. 错误猜测法 (error guessing)

所谓猜错, 就是猜测被测程序在哪些地方容易出错, 然后针对可能的薄弱环节来设计测试用例。显然, 它比前两种方法更多地依靠测试人员的直觉与经验。所以, 一般都先用前两种方法设计测试用例, 然后用猜错法补充一些例子作为辅助的手段。

仍以例 8.3 的程序设计要求为例, 在已经用等价分类法 (见例 8.3) 和边界值分析法 (见例 8.4) 设计测试用例的基础上, 还可用猜错法补充一些测试用例, 例如:

- ① 出生年月为“0”。
- ② 漏输“出生年月”。
- ③ 年月次序颠倒, 例如将“197512”误输为“121975”, 等等。

除了上述 3 种方法外, 因果图法也是较常用的一种黑盒测试技术。因果图 (cause-effect graph) 是一种简化了的逻辑图。当被测程序具有多种输入条件, 程序的输出又依赖于输入条件的各种组合时, 用因果图直观地表明输入条件和输出动作之间的因果关系, 能帮助测试人员把注意力集中到与程序功能有关的那些输入组合上, 比采用等价分类法有更高明的测试效率。这种方法操作步骤比较复杂, 详情就不再介绍了。

## 8.5.2 白盒测试

白盒测试以程序的结构为依据, 所以又称为结构测试。早期的白盒测试把注意力放在流程图的各个判定框, 使用不同的逻辑覆盖标准来表达对程序进行测试的详尽程度。随着测试技术的发展, 人们越来越重视对程序执行路径的考察, 并且用程序图代替流程图来设计测试用例。为了区分这两种白盒测试技术, 以下把前者称为逻辑覆盖测试 (logic coverage testing), 后者称为路径测试 (path testing)。

在本节中, 将以一个升序排序的 Pascal 程序作为引例, 分别用逻辑覆盖测试法和路径法为这一程序设计测试用例。具体代码如下:

```

LABEL
99;
CONST
n=100;
VAR
a: ARRAY[1..n] of INTEGER;
i,j,k,temp:INTEGER;
BEGIN
  READLN (k);
  FOR i:=1 TO k DO READ(a[i]);
  FOR i:=2 TO k DO
    BEGIN
      IF a[i]>=a[i-1] THEN GOTO 99;
      FOR j:=i DOWNT0 2 DO
        BEGIN
          IF a[j]>= a[j-i] THEN GOTO 99;
          temp:=a[j];

```

```

    a[j]:=a[j-1];
    a[j-1]:=temp;
  END;
99:
  END
  FOR i:=1 TO k DO WRITE(a[i])
  END

```

以上排序程序采用的是冒泡排序（bubble sorting）算法。其基本步骤是：

- ① 从一组数中取出第一个数。
- ② 取下一个数。如数已取完，则排序结束。
- ③ 如果所取数大于等于其前邻数，则重复第②步。
- ④ 如果所取数小于其前邻数，则与其前邻数交换位置。
- ⑤ 重复第④步，直至所取已无前邻数（即已交换到当前数列的第一位置），或大于等于其前邻数为止。
- ⑥ 返回第②步。

图 8.5 显示了该程序排序部分的流程图。由图可知：

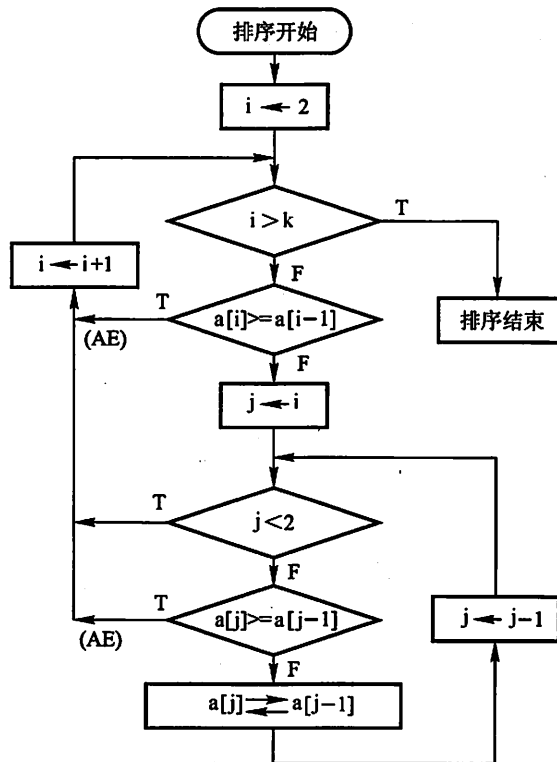


图 8.5 冒泡程序中排序部分的流程图

① 除第一个数之外，每取出一新数，便加到前面已取出的数列末尾重新排序。 $k$ 个数重复排序 $k-1$ 次。

② 每次排序，将所取数由下向上依次与它的上一个数比较。只要它小于上一个数，就把它移到上一个数的上面。恰如水中气泡轻者上浮，从水底不断冒向水面一般。

### 1. 逻辑覆盖测试法

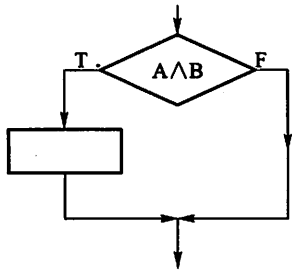
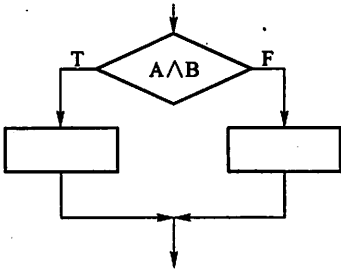
逻辑覆盖测试法通用流程图来设计测试用例，它考察的重点是图中的判定框（菱形框）。因为这些判定若不是与选择结构有关，就是与循环结构有关，因此是决定程序结构的关键成分。

按照对被测程序所作测试的有效程度，逻辑覆盖测试可由弱到强区分为5种覆盖标准，如表8.7所示。表8.8显示了实现各种覆盖标准的简单示例。

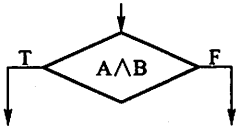
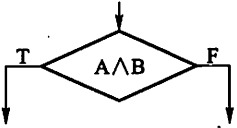
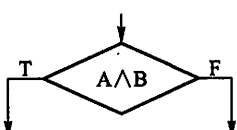
表 8.7 逻辑覆盖测试的5种标准

发现 错误的 能力	弱 ↓ 强	语句覆盖	每条语句至少执行一次
		判定覆盖	每一判定的每个分支至少执行一次
		条件覆盖	每一判定中的每个条件，分别按“真”、“假”至少各执行一次
		判定/条件覆盖	同时满足判定覆盖和条件覆盖的要求
		条件组合覆盖	求出判定中所有条件的各种可能组合值，每一可能的条件组合至少执行一次

表 8.8 5种覆盖标准的示例

覆 盖 标 准	程序结构举例	测试用例应满足的条件
语句覆盖		$A \wedge B = T$
判定覆盖		$A \wedge B = T, A \wedge B = F$

续表

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		$A \wedge B=.T., A \wedge B=.F.$ $A=.T., A=.F.$ $B=.T., B=.F.$
条件组合覆盖		$A=.T. \wedge B=.T.$ $A=.T. \wedge B=.F.$ $A=.F. \wedge B=.T.$ $A=.F. \wedge B=.F.$

以下结合引例，说明按照不同覆盖标准设计测试用例的方法。

(1) 对引例作逻辑覆盖测试

[例 8.5] 试按表 8.7 中的不同标准，为引例中的排序程序设计测试用例。

[解] 从冒泡排序程序代码和图 8.5 可知，排序程序具有双重嵌套循环结构。其内外层循环体各包含一条选择语句，用于在条件满足时提前退出循环。程序中的 4 个判断是测试时考察的重点。以下分别列出按不同覆盖标准设计的测试用例：

① 语句覆盖。稍作分析便不难看出，只要输入前大后小的两个数，程序执行时就可以遍历流程图中的所有框。因此，仅需选用一组测试数据如

$\{a=\{8,4\}, k=2\}$

就能够实现语句覆盖。这类覆盖发现错误的能力不强，例如若将程序中的两个“ $\geq$ ”均误写为“ $=$ ”，用上述的测试数据就不能发现。

② 判定覆盖。选用上述的测试数据，内、外层循环都是从正常的循环出口退出的。要实现判定覆盖，还需在语句覆盖的基础上，增加两个能使程序从非正常出口（在图 8.5 中用 AE 标志）退出的测试数据。例如，用以下两组数据

$\{a=\{8,4,9\}, k=3\}$

$\{a=\{8,4,4\}, k=3\}$

或

$\{a=\{8,4,8,4\}, k=4\}$

则程序将在满足  $(a[i] = a[i-1])$  或  $(a[j] = a[j-1])$  的条件下通过非正常出口，也能实现判定覆盖。但又可能出现另一种偏向，掩盖把“ $\geq$ ”误写为“ $=$ ”的错误，造成更加严重的测试漏洞。

③ 条件覆盖。从以上分析很容易想到，必须选取足够的测试，使多个条件中每个条件

分别按“真”、“假”出现一次，才能克服前述的缺点，进一步提高发现错误的能力。这就是条件覆盖的由来。就本例而言，如果使用测试数据

$\{a=\{8,4,9,6\}, k=4\}$

$\{a=\{8,4,8,4\}, k=4\}$

就能对程序实现条件覆盖。此时  $a[i]$  (或  $a[j]$ ) 大于、等于或小于  $a[i-1]$  (或  $a[j-1]$ ) 的 3 种情况将分别至少出现一次，无论把“ $\geq$ ”误写为“ $>$ ”或“ $=$ ”，都可用这两组数据检查出来。

④ 其他覆盖。本例中的两个条件  $a[i] \geq a[i-1]$  及  $a[j] \geq a[j-1]$ ，其组成条件都不是互相独立的。如果其中有一个条件 (例如  $a[i] > a[i-1]$ ) 为真，则另一个条件 (如  $a[i] = a[i-1]$ ) 必然为假。所以就本例来说，判定条件覆盖及条件组合覆盖都没有实际意义，可以不必讨论。

由此可见，本例宜选择条件覆盖，以便得到较强的查错能力。测试数据可选择

$\{a=\{8,4,9,6\}, k=4\}$

$\{a=\{8,4,8,4\}, k=4\}$

或合成一组

$\{a=\{8,4,8,4,9,6\}, k=6\}$

## (2) 关于覆盖标准的讨论

在表 8.7 的 5 种覆盖中，语句覆盖发现错误的能力最弱，一般不单独采用。判定覆盖与条件覆盖的差别在于：前者把判定看成一个整体，后者则着眼于其中的一个条件。当一个判定只含一个条件时，判定覆盖也就是条件覆盖。但如果一个判定含有一个以上的条件 (称其为复合条件)，采用判定覆盖有可能出现例如下述的漏洞，即判定中有些条件得到测试，另一些条件却被忽略，从而掩盖程序的错误。条件覆盖要求对每一条件进行单独的检查，一般说来它的查错能力比判定覆盖更强，但也并不尽然。在图 8.6 中，如果由条件 A、B 的 4 种逻辑值组成内容为 {A 真, B 假} 和 {A 假, B 真} 的两组测试数据，则无论判定包含的条件是“A and B”或“A or B”，都只覆盖一个分支，另一个分支未被覆盖。把判定覆盖和条件覆盖的要求汇集于一身的判定/条件覆盖，正是为了弥补条件覆盖的这一不足之处。

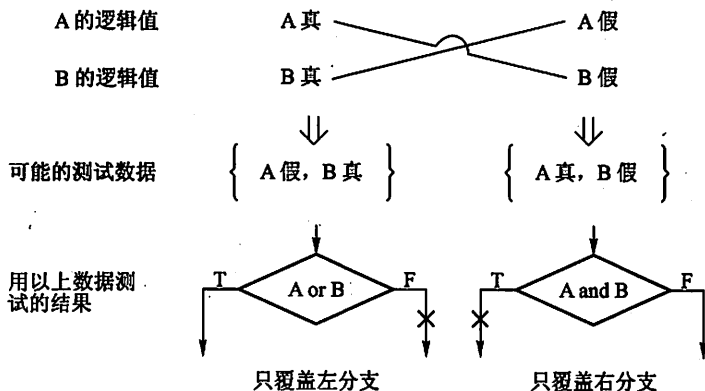


图 8.6 只能覆盖一个分支的条件覆盖一例