软件质量保证与软件测试
-路径测试补充内容

结构性测试主要检查内容

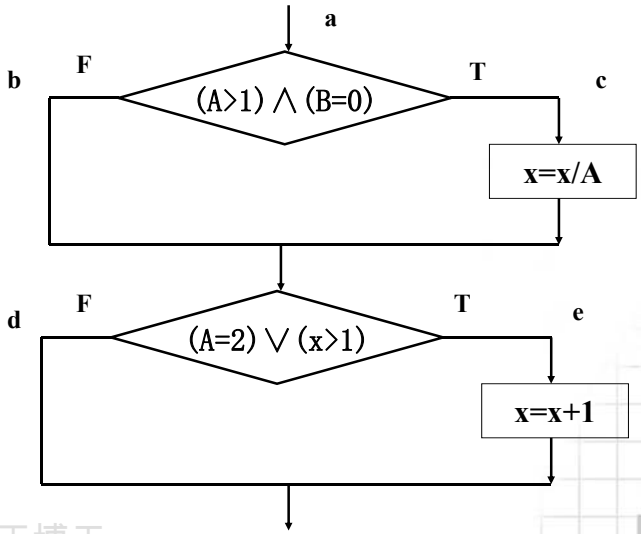
1. 对程序模块的所有独立的执行路径至少测试一次
2. 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次
3. 在循环的边界和运行界限内执行循环体
4. 等等

结构性测试的主要方法

逻辑覆盖，是以程序内部的逻辑结构为基础的设计测试用例的技术，它属于白盒测试

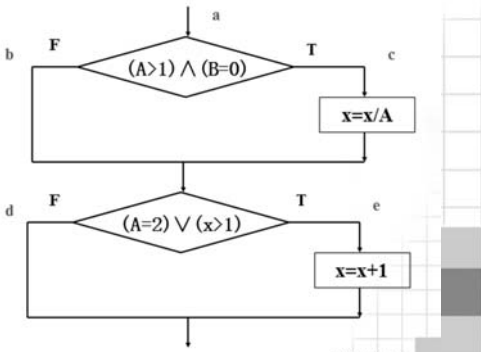
- 语句覆盖
- 判定（判断）覆盖
- 条件覆盖
- 判定-条件覆盖
- 条件组合覆盖
- 路径覆盖

示例



路径

- 该程序有4条执行路径：
  - L1（ a → c → e ）
  - L2（ a → b → d ）
  - L3（ a → b → e ）
  - L4（ a → c → d ）



语句覆盖

- 语句覆盖就是设计若干个测试用例，运行被测程序，使得每一条可执行语句至少执行一次
- 在图例中，正好所有的可执行语句都在路径L1上，所以选择路径 L1设计测试用例，就可以覆盖所有的可执行语句

L1:(A=2) and (B=0) and (x/A>1)

【(2, 0, 4), (2, 0, 3)】覆盖 ace【L1】

- 语句覆盖可能发现不了判定中逻辑运算中出现的错误
- 如果将例子中的∧和∨互换，使用该用例仍可覆盖所有4个可执行语句

语句覆盖

- 优点：
  - 可以很直观地从源代码得到测试用例，无须细分每条判定表达式
- 缺点：
  - 由于这种测试方法仅仅针对程序逻辑中显式存在的语句，但对于隐藏的条件和可能到达的隐式逻辑分支，是无法测试的
  - 在if结构中若源代码没有给出else后面的执行分支，那么语句覆盖测试就不会考虑这种情况
  - 在Do-While结构中，语句覆盖执行其中某一个条件分支
  - 语句覆盖对于多分支的逻辑运算是无法全面反映的，它只在乎运行一次，而不考虑其他情况

判定覆盖（分支覆盖）

- 判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次

L1: (A=2) and (B=0) and (x/A>1)
L2: (A<=1) and (x<=1) 或者
(B<>0) and (A<>2) and (x<=1)

【(2, 0, 4), (2, 0, 3)】覆盖 ace【L1】
【(1, 1, 1), (1, 1, 1)】覆盖 abd【L2】

- 判定覆盖不能保证一定能查出在判断条件中存在的错误
- 如果将X>1错写成X<1，利用上面的测试用例仍可得到同样的结果

判定覆盖（分支覆盖）

- 优点：
  - 判定覆盖比语句覆盖要多几乎一倍的测试路径，当然也就具有比语句覆盖更强的测试能力
  - 同样判定覆盖也具有和语句覆盖一样的简单性，无须细分每个判定就可以得到测试用例

- 缺点：
  - 往往大部分的判定语句是由多个逻辑条件组合而成（如，判定语句中包含AND、OR），若仅仅判断其整个最终结果，而忽略每个条件的取值情况，必然会遗漏部分测试路径

条件覆盖

- 条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次

- 对于第一个判断：
  条件 A＞1 取真为 T1，取假为 !T1
  条件 B＝0 取真为 T2，取假为 !T2

- 对于第二个判断：
  条件A＝2 取真为 T3，取假为 !T3
  条件X＞1 取真为 T4，取假为 !T4

条件覆盖测试用例

测试用例	通过路径	条件取值	覆盖分支
【(2, 0, 4)，(2, 0, 3)】	ace(L1)	T1 T2 T3 T4	c，e
【(1, 0, 1)，(1, 0, 1)】	abd(L2)	!T1 T2 !T3 !T4	b，d
【(2, 1, 1)，(2, 1, 2)】	abe(L3)	T1 !T2 T3 !T4	b，e

或者

测试用例	通过路径	条件取值	覆盖分支
【(1, 0, 3)，(1, 0, 4)】	abe(L3)	!T1 T2 !T3 T4	b，e
【(2, 1, 1)，(2, 1, 2)】	abe(L3)	T1 !T2 T3 !T4	b，e

条件覆盖

- 优点：
  - 显然条件覆盖比判定覆盖，增加了对符合判定情况的测试，增加了测试路径

- 缺点：
  - 要达到条件覆盖，需要足够多的测试用例
  - 但条件覆盖并不能保证判定覆盖,条件覆盖只能保证每个条件至少有一次为真，而不考虑所有的判定结果

判定--条件覆盖

- 判定－条件覆盖就是设计足够的测试用例，使得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

测试用例	通过路径	条件取值	覆盖分支
【(2, 0, 4), (2, 0, 3)】	ace(L1)	T1 T2 T3 T4	c，e
【(1, 1, 1), (1, 1, 1)】	abd(L2)	!T1 !T2 !T3 !T4	b，d

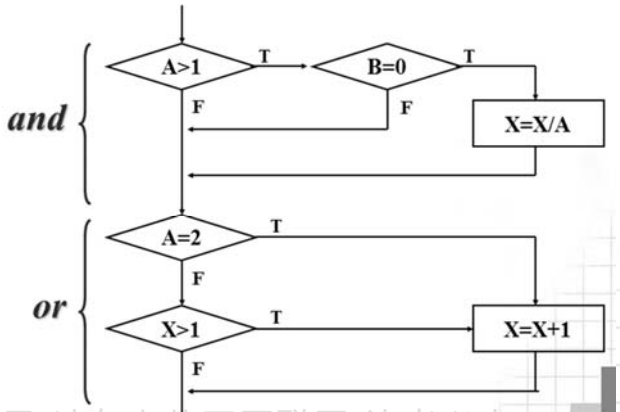
- 采用判定-条件覆盖，逻辑表达式中的错误不一定能够查出来

判定--条件覆盖

- 优点：
  - 判定/条件覆盖满足判定覆盖准则和条件覆盖准则，弥补了二者的不足
- 缺点：
  - 判定/条件覆盖准则的缺点是未考虑条件的组合情况

分解例子

- 因为判断是复合条件，包含多个条件，因此，将条件分解，可以进一步细化设计测试用例



条件组合覆盖

- 条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次

记：① A＞1，B＝0　作 T1T2
② A＞1，B≠0　作 T1!T2
③ A≯1，B＝0　作 !T1T2
④ A≯1，B≠0　作 !T1!T2

⑤ A＝2，X＞1　作 T3T4
⑥ A＝2，X≯1　作 T3!T4
⑦ A≠2，X＞1　作 !T3T4
⑧ A≠2，X≯1　作 !T3!T4

条件组合覆盖测试用例

测试用例	通过路径	覆盖条件	覆盖组合号
【(2, 0, 4), (2, 0, 3)】	ace(L1)	T1 T2 T3 T4	① ⑤
【(2, 1, 1), (2, 1, 2)】	abe(L3)	T1 !T2 T3 !T4	② ⑥
【(1, 0, 3), (1, 0, 4)】	abe(L3)	!T1 T2 !T3 T4	③ ⑦
【(1, 1, 1), (1, 1, 1)】	abd(L2)	!T1 !T2 !T3 !T4	④ ⑧

条件组合覆盖

- 优点：
  - 多重条件覆盖准则满足判定覆盖、条件覆盖和判定-条件覆盖准则
  - 作为改进的判定-条件测试，要求设计足够的测试用例，使得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；并且每个条件都显示能单独影响判定结果
- 缺点：
  - 线性地增加了测试用例的数量

路径测试

- 路径测试就是设计足够的测试用例，覆盖程序中所有可能的路径

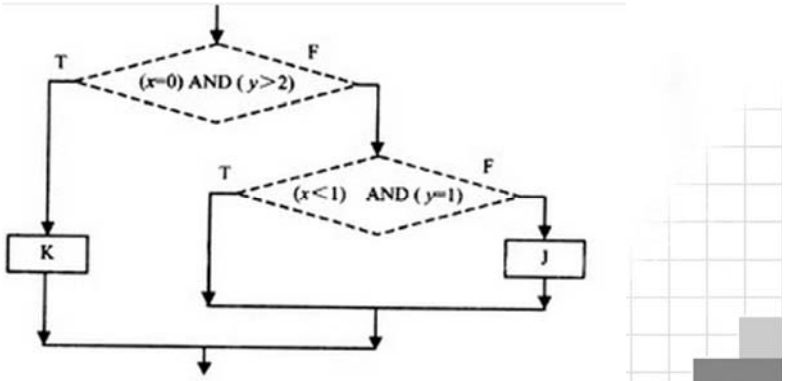
测试用例	通过路径	覆盖条件
【(2, 0, 4), (2, 0, 3)】	ace(L1)	T1 T2 T3 T4
【(1, 1, 1), (1, 1, 1)】	abd(L2)	!T1 !T2 !T3 !T4
【(1, 1, 2), (1, 1, 3)】	abe(L3)	!T1 !T2 !T3 T4
【(3, 0, 3), (3, 0, 1)】	acd(L4)	T1 T2 !T3 !T4

路径测试

- 优点：
  - 可以对程序进行彻底的测试，比前面五种的覆盖面都广
- 缺点：
  - 由于路径覆盖需要对所有可能的路径进行测试（包括循环、条件组合、分支选择等），那么需要设计大量、复杂的测试用例，使得工作量呈指数级增长
  - 在有些情况下，一些路径是不可能被执行的
  - 这样不仅降低了测试效率，而且大量的测试结果的累积，也为排错带来麻烦

练习1

- 为该图所示的程序，采用该部分所讲方法，设计测试用例



练习2

- 对三角形问题程序（这里为两种实现），采用该部分所讲方法，设计测试用例