第三部分

白盒测试

白盒测试的思想

- 利用程序内部的逻辑结构及有关信息，设计或选择测试用例。
  - 对程序所有逻辑路径进行测试。
  - 通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。
- 又称： 结构性测试，逻辑驱动测试。

白盒测试的一般原则

- 保证模块中所有的
  - **独立路径**：至少被执行一次;
  - **条件语句的每一个分支**：至少被执行一次;
  - **循环语句**都在边界条件和一般条件下：至少被执行一次;
  - 第9章：
    - 逻辑覆盖→基本路径测试
    - 工具：控制流图
- 验证所有内部数据结构的有效性
  - 第10章

逻辑覆盖

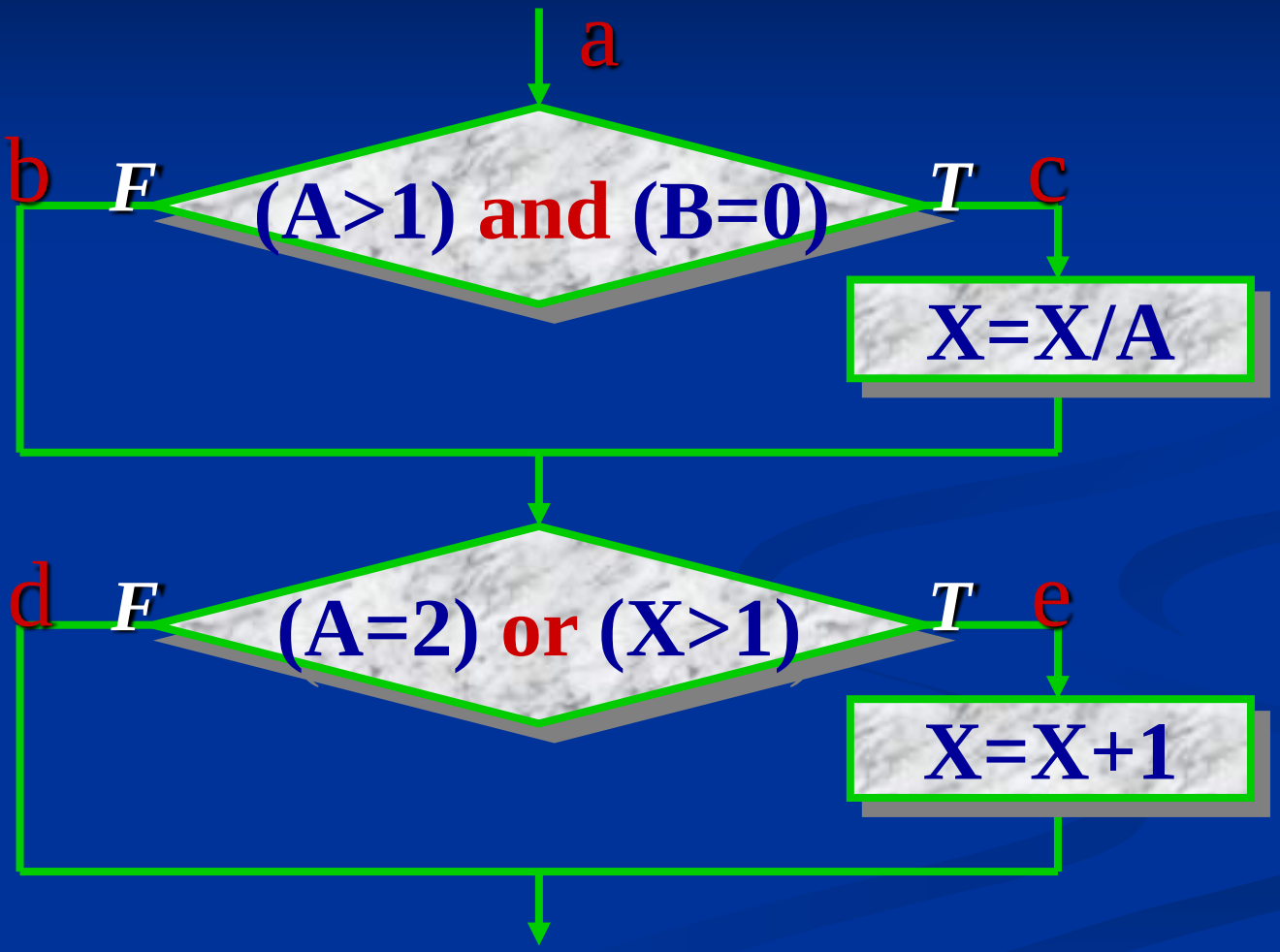
- 测试用例的生成
  - 考察程序内部的逻辑结构
  - 根据一定的逻辑覆盖的目标

- 逻辑覆盖可分为：
  - 语句覆盖SC
  - 判定覆盖DC
  - 条件覆盖CC
  - 条件/判定覆盖 C/DC
  - 修订的条件/判定覆盖 MC/DC
  - 条件组合（多条件）覆盖 MCC
  - 路径覆盖

逻辑覆盖—例子

```
Procedure example(A,B:real; Var X:real)
begin
    if (A>1) and (B=0) then X:=X/A;
    if (A=2) or (X>1) then X:=X+1;
end
```

逻辑覆盖—流程图



- 有上图可知，程序包含4条路径：
  - L_1：（a→c→e）
  - L_2：（a→b→d）
  - L_3：（a→b→e）
  - L_4：（a→c→d）

- L_1: （a→c→e）

= {(A>1) and (B=0)} and

{(A=2) or (X/A>1)}

= (A>1) and (B=0) and (A=2) or

(A>1) and (B=0) and (X/A>1)

= (A=2) and (B=0) or

(A>1) and (B=0) and (X/A>1)

- L_2: （ a→ b → d ）

= not{(A>1) and (B=0)} and
not{(A=2) or (X/A>1)}

= { not (A>1) or not (B=0) } and
{ not (A=2) and not (X/A>1) }

= not (A>1) and not (A=2) and not (X/A>1)
or
not (B=0) and not (A=2) and not (X/A>1)

- L_3: （ a→ b→ e ）

= not {(A>1) and (B=0)} and
{(A=2) or (X/A>1)}

= { not (A>1) or not (B=0)} and
{(A=2) or (X/A>1)}

= not (A>1) and (A=2) or
not (A>1) and (X/A>1) or
not (B=0) and (A=2) or
not (B=0) and (X/A>1)

- L_4: （ a→ c → d ）

= {(A>1) and (B=0)} and

not {(A=2) or (X/A>1)}

= (A>1) and (B=0) and not (A=2) and not (X/A>1)

语句覆盖SC

- 语句覆盖: 是指在测试的过程中，软件测试人员应选择足够多的测试用例，保证被测试程序中的每一个语句至少执行一次。

- 例：为上例设计的满足语句覆盖的测试用例为:

  【(2, 0, 4)，(2, 0, 3)】　　覆盖 ace【L1】

  - 测试用例的格式为：

    【输入的(A, B, X)，预期输出的(A, B, X)】

  - 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。

- 语句覆盖是最弱的覆盖准则。

- 不能发现如下错误：
  - 对于判断语句：只能覆盖其中的一个分支，不能发现另一个分支中的错误。
    - 本例：只有条件为“真”的分支被测试到了，而条件为“假”的分支被忽视了，如果这个分支出现错误，则不能被这种测试方法发现；
  - 对于复合条件语句：
    - **(A>1) and (B=0)** 被误写成 **(A>1) or (B=0)**
      - 本测试用例 **(2, 0, 4)** 无法发现此类错误。

判定覆盖DC

- 判定覆盖，又称分支覆盖
  - 保证程序中每一个判断的取真分支至少经历一次；取假分支至少经历一次。

- 图例：选择路径L1和L2，就可满足判定覆盖:

  【(2, 0, 4)，(2, 0, 3)】覆盖 ace【L1】

  【(1, 1, 1)，(1, 1, 1)】覆盖 abd【L2】

- 测试用例的取法并不唯一;

  选择路径L3和L4，也可满足判定覆盖:

  [a,b,X] : {[2,1,1] , [2,1,2]} 覆盖L3

  [a,b,X] : {[3,0,3] , [3,1,1]} 覆盖L4

- 判定覆盖比语句覆盖强
- 但是，判定覆盖仍然不能确保一定查出在判定条件中存在的错误。
  - 对于复合条件，有可能查不出某个子条件中的错误。
  - 上例： 第二个条件

    (A=2) or (X>1)　误写成　(A=2) or (X<1)
    - 本测试用例【 (2, 0, 4)，(2, 0, 3) 】和【 (1, 1, 1)，(1, 1, 1) 】无法发现此类错误。

条件覆盖CC

- 条件覆盖：保证程序中每个判断的每一个子条件的可能取值至少执行一次。
- 在图例中，我们事先可对所有条件取值加以标记。例如：
- 对于第一个判断：
  - 子条件 A＞1 取真为 T_1，取假为 $\overline{T_1}$

    子条件 B＝0 取真为 T_2，取假为 $\overline{T_2}$
- 对于第二个判断：
  - 子条件A＝2 取真为 T_3，取假为 $\overline{T_3}$

    子条件X＞1 取真为 $\perp^{\dashv}$，取假为 $\overline{T_4}$

测试用例	通过路径	条件取值	覆盖分支
[(1,0,3),(1,0,4)]	abe(L_3)	$\overline{T}_1 T_2 \overline{T}_3 \overline{T}_4$	b、e
[(2,1,1),(2,1,2)]	abe(L_3)	$T_1 \overline{T}_2 T_3 T_4$	b、e

- “条件覆盖”一定比“判定覆盖”强？
  - 不一定。本例：
    - 满足条件覆盖，
    - 不能满足判定覆盖：对于第一个判断，只覆盖了取假分支

条件/判定覆盖 （C/D C）

- 保证每个判断的所有分支至少执行一次，判断中每一个子条件的所有可能取值至少执行一次，。
- 图例:

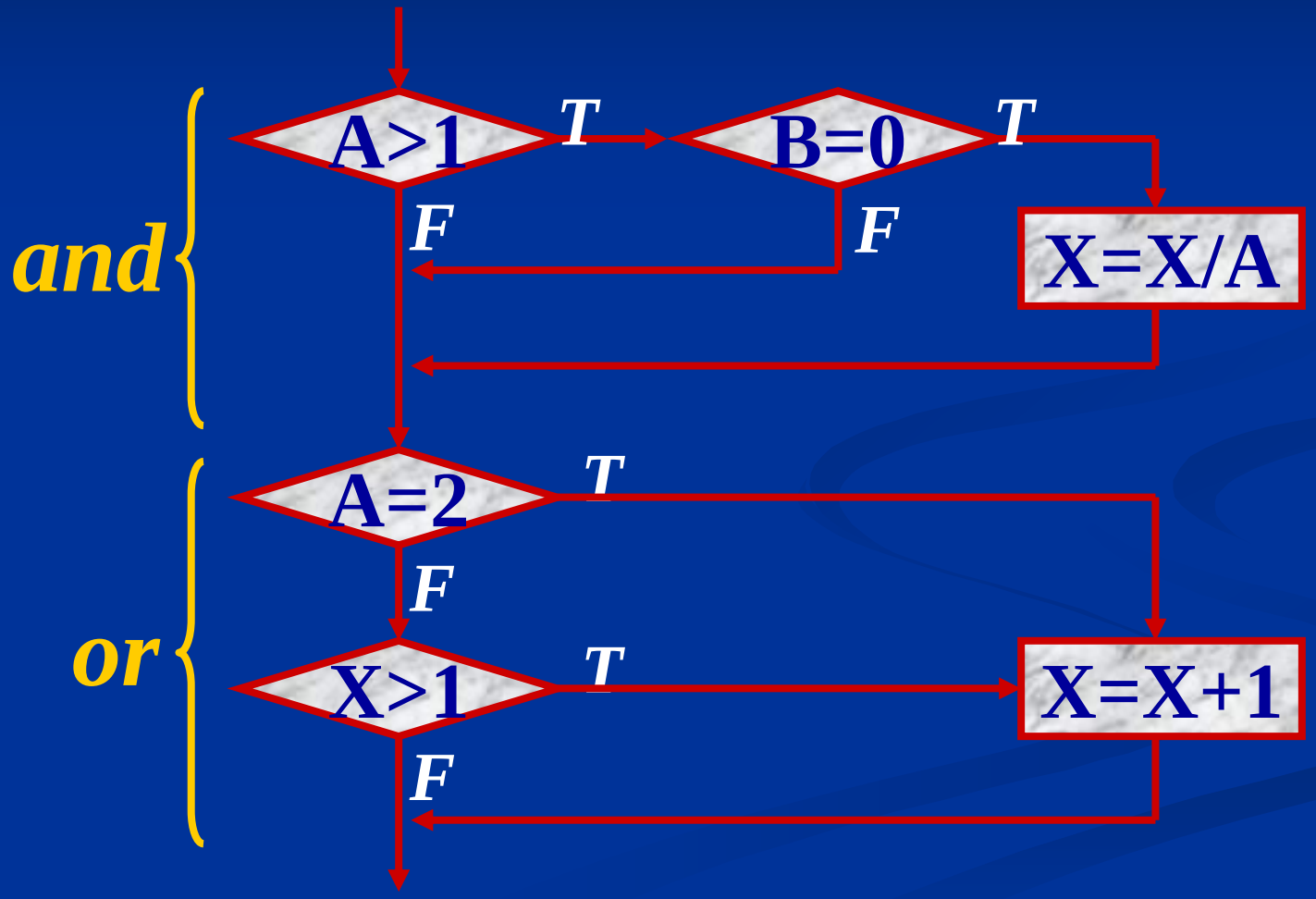
测试用例	通过路径	条件取值	覆盖分支
[(2,0,3),(2,0,4)]	ace(L_1)	$T_1T_2T_3T_4$	c、e
[(1,1,1),(1,1,1)]	abd(L_2)	$\overline{T}_1\overline{T}_2\overline{T}_3\overline{T}_4$	b、d

- “条件/判定覆盖”和判定覆盖一样存在缺陷，不一定能查出在判定条件中存在的错误。
  - 对于复合条件，有可能查不出某个子条件中的错误。
  - 上例： (A=2) or (X>1) 误写成(A=2) or (X<1)
    - 本测试用例【(2, 0, 3)，(2, 0, 4)】和
      【(1, 1, 1)，(1, 1, 1)】

      无法发现此类错误。

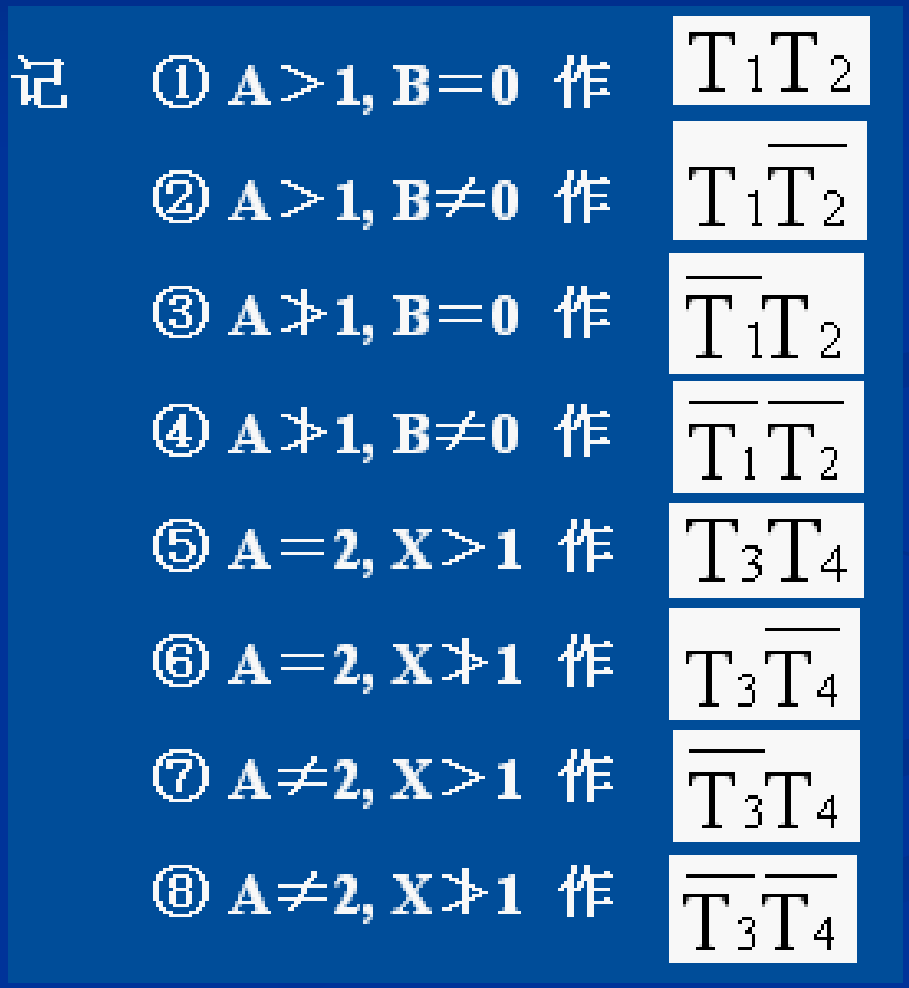
- 理解【(2, 0, 3)，(2, 0, 4)】：需要编译原理的一些知识。
  - 对于表达式(A=2) or (X>1)来说,如果(A=2)的结果为假,则还要测试(X>1)才能决定表达式的值;
  - 而如果(A=2)的结果为真,表达式的值立刻为真，条件(X>1)将不能被检测到。
- 理解【(1, 1, 1)，(1, 1, 1)】:
  - 因为X值为1， 所以子条件 为 X<1 和 为 X>1 都一样是“假”

- 解决：在流程图中，将单个的“复合条件”判定分解为多个基本的子条件判断。



条件组合（多条件）覆盖MCC

- 使得每一个判断的所有可能的子条件取值组合至少执行一次。
- 图例：



条件组合覆盖

测试用例	通过路径	条件取值	覆盖分支
[(2,0,4),(2,0,3)]	ace(L_1)	$T_1T_2T_3T_4$	1、2
[(1,0,3),(1,0,4)]	abe(L_3)	$T_1\overline{T}_2T_3\overline{T}_4$	3、7
[(2,1,1),(2,1,2)]	ace(L_3)	$\overline{T}_1T_2\overline{T}_3T_4$	2、6
[(1,1,1),(1,1,1)]	abd(L_2)	$\overline{T}_1\overline{T}_2\overline{T}_3\overline{T}_4$	4、8

仍然有遗漏的路径

因为没有考虑 多个判定之间的组合。

- （第一个分支/第二个分支） TT；FT；TF；FF
- 所以少了TF

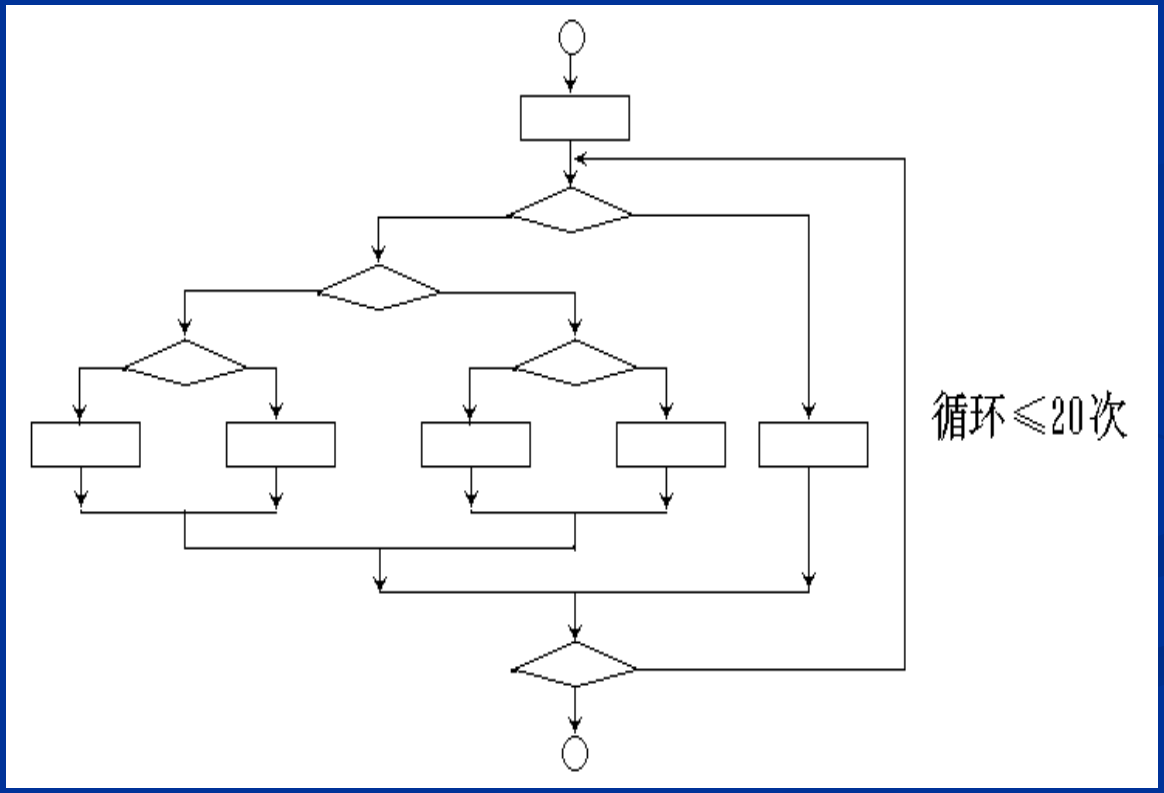
路径测试

- 覆盖程序中所有可能的路径.

测试用例	通过路径	条件取值
[(2,0,4),(2,0,3)]	ace(L_1)	$T_1T_2T_3T_4$
[(1,1,1),(1,1,1)]	abd(L_2)	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{T_4}$
[(1,1,2),(1,1,3)]	abe(L_3)	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}T_4$
[(3,0,3),(3,0,1)]	acd(L_4)	$T_1T_2\overline{T_3}\,\overline{T_4}$

路径测试－穷举路径是不可能的

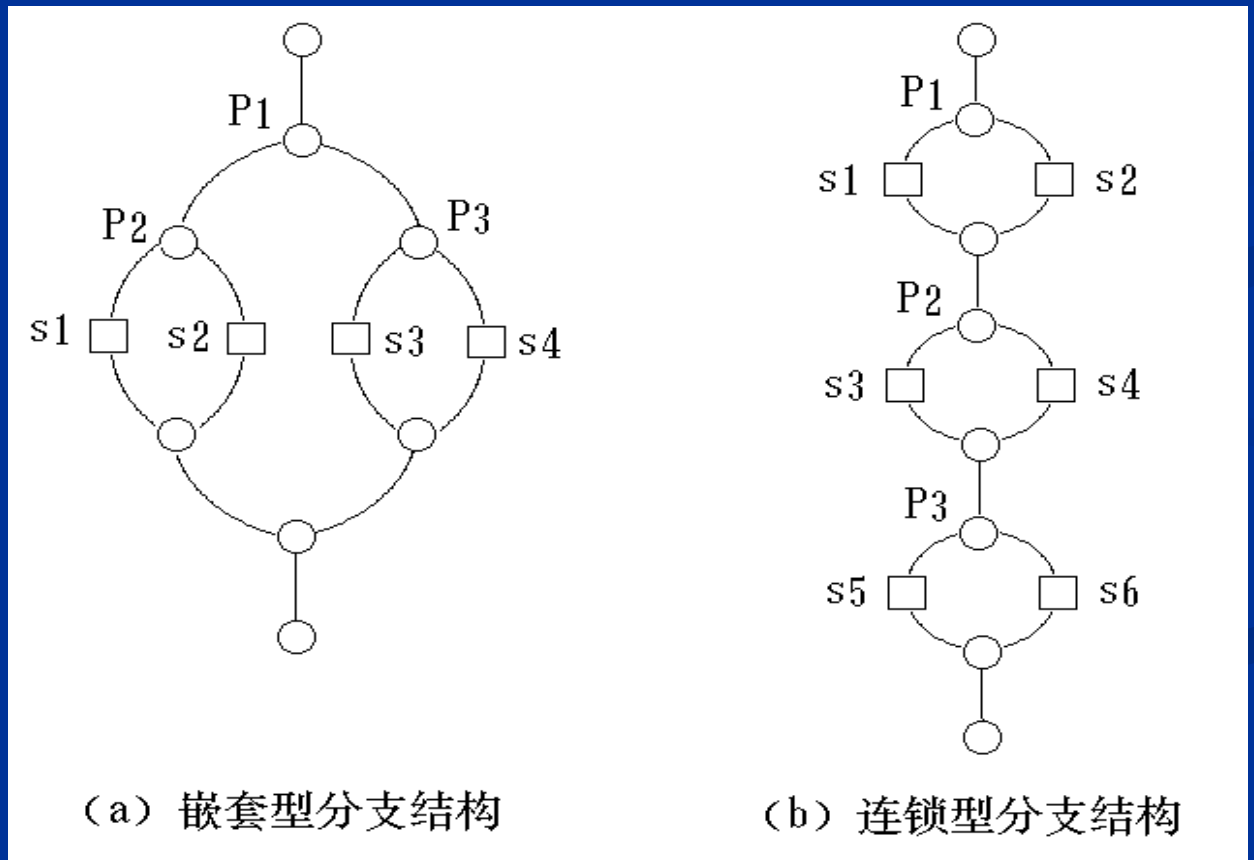
- 若程序具有多重判定和嵌套循环，路径数目可能是天文数字
- 例：假定对每一条路径进行测试需要1毫秒，一年工作365 × 24小时，要想把所有路径测试完，需多少年？ 大约6万年



- 路径测试中的路径选择：
  - 多重条件判断的路径选择
  - 循环的路径选择

多重条件判断的路径选择

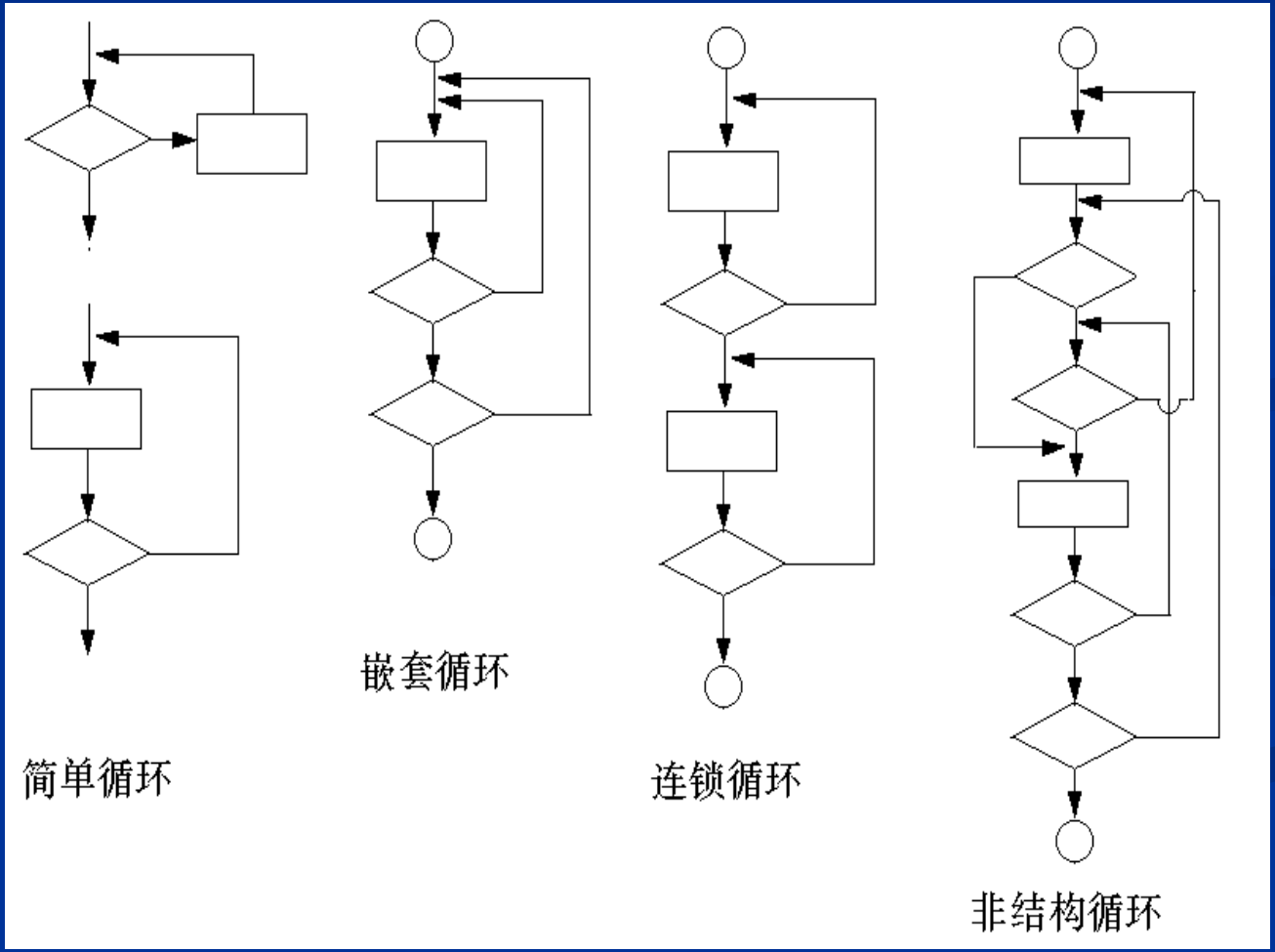
- 当程序中的判定条件多于一个时，分支结构可以分为：
  - 嵌套型分支结构：若有n个判定语句，则有n+1条路径；
  - 连锁型分支结构：若有n个判定语句，则有2^n条路径。



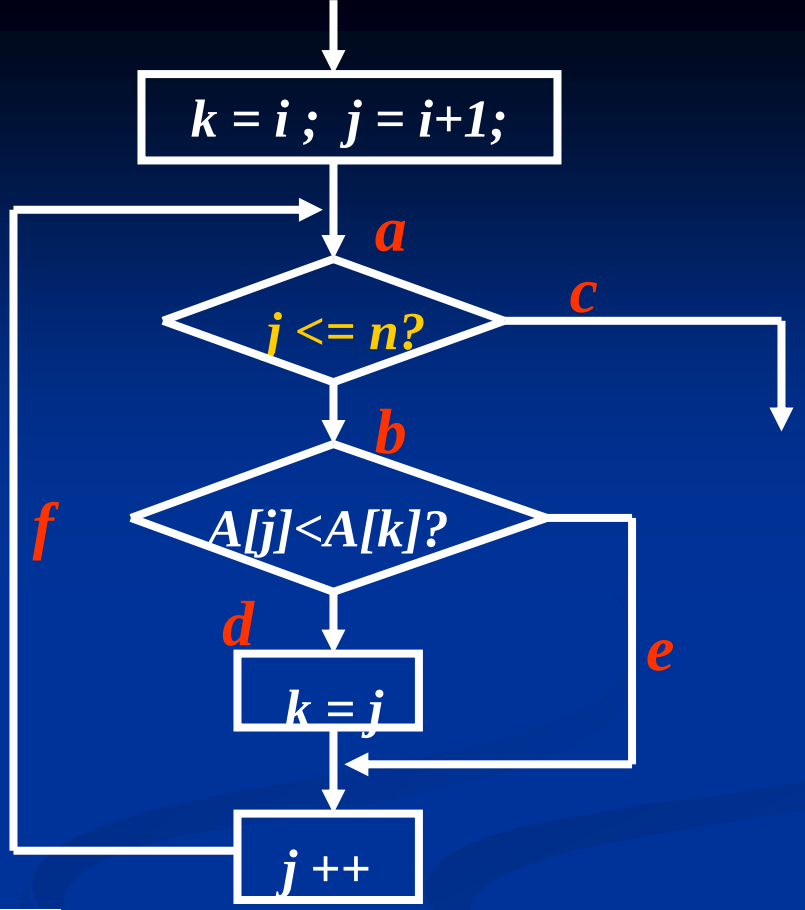
（a）嵌套型分支结构　　（b）连锁型分支结构

循环的路径选择

- 循环的类型：
  - 简单循环
  - 复杂循环：嵌套循环、连锁循环、非结构循环。



- **简单循环的路径选择：**
  （设循环的最大次数是n）
  - 跳过整个循环；
  - 只循环一次；
  - 循环执行二次；
  - 使循环变量的取值为m，(m<n)，执行一次
  - 使循环变量的取值分别为n-1、n、n+1， 执行三次
- 例：



循环	i	n	A[i]	A[i+1]	A[i+2]	k	路 径
0	1	1				i	a c
1	1	2	1	2		i	a b e f c
			2	1		i+1	a b d f c
2	1	3	1	2	3	i	a b e f e f c
			2	3	1	i+2	a b e f d f c
			3	2	1	i+2	a b d f d f c
			3	1	2	i+1	a b d f e f c

复杂循环的路径选择

- 嵌套循环
  - 对最内层循环做简单循环的全部测试，所有其它层的循环变量取最小循环次数
  - 逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环取最小循环次数，所有其它嵌套内层循环的循环变量取“典型”值；
  - 反复进行，直到所有各层循环测试完毕。
  - 最后，对全部各层的循环变量同时取最小循环次数，或者同时取最大循环次数。
- 连锁循环
  - 如果串接循环是独立的，采用简单循环的测试原则；
  - 如果第一循环的结果是第二循环的初值，则两个循环不独立，采用嵌套循环的测试策略。
- 非结构化循环
  - 先进行结构化改造。

- 软件测试的风险性
  - 穷举是最安全、最全面的方法，但是不可能。
  - 使用任何一种测试方法都意味着选择一种风险。
  - 但是，我们却不得不使用这些测试方法。

- 逻辑覆盖可分为：
  - 语句覆盖SC
  - 判定覆盖DC
  - 条件覆盖CC
  - 条件/判定覆盖 C/DC
  - 修订的条件/判定覆盖 MC/DC
  - 条件组合（多条件）覆盖 MCC
  - 路径覆盖
    - 基本路径覆盖
    - 主路径覆盖

实践中常用的逻辑覆盖法

- 语句覆盖：100%
- 修订的条件/判定覆盖：逻辑复杂、安全攸关系统
- 路径覆盖：基本路径覆盖；主路径覆盖

修订的条件/判定覆盖 MC/DC

- 视频
- 是欧美民用航空器的强制标准。
  - 属于DO-178B Level A认证标准
- MC/DC定义：
  - 判定: 包含条件、布尔值 或布尔操作符 的一个布尔表达式
    - 条件: 不含布尔操作符的布尔表达式
  - 修正的条件判定覆盖:
    - 满足C/DC （条件/判定覆盖）
      - 每个判定的真假被覆盖
      - 判定中每个条件的真假被覆盖
    - 一个判定中的每个条件能够独立影响这个判定的结果，
      - 即在其他条件不变的前提下仅改变这个条件的值，而使判定结果改变

修订的条件/判定覆盖 MC/DC

- MC/DC：
  - 首先要满足C/DC（条件/判定覆盖），
  - 在此基础上，对于每一个条件C，要求存在两次计算以满足如下条件：
    当“条件C取值相反，其余条件取值不变”时，判定的结果值也相反。
- 理解：
  - 核心：每个条件都要独立影响判定结果。

修订的条件/判定覆盖 MC/DC

- 示例代码：

```
int func(BOOL A, BOOL B, BOOL C)
{
    if(A && (B || C))
        return 1;
    return 0;
}
```

Name	Case 1	case 2	case 3
A	1	0	1
B	1	1	0
C	0	0	0
ret	1	0	0

对于条件A，用例1和用例2，A取值相反，B和C相同，判定结果分别为1和0；
对于条件B，用例1和用例3，B取值相反，A和C相同，判定结果分别为1和0；
对于条件C，？

修订的条件/判定覆盖 MC/DC

- 示例代码：

```
int func(BOOL A, BOOL B, BOOL C)
{
    if(A && (B || C))
        return 1;
    return 0;
}
```

Name	Case 1	case 2	case 3	case 4
A	1	0	1	1
B	1	1	0	0
C	0	0	0	1
ret	1	0	0	1

对于条件A，用例1和用例2，A取值相反，B和C相同，判定结果分别为1和0；
对于条件B，用例1和用例3，B取值相反，A和C相同，判定结果分别为1和0；
对于条件C，用例3和用例4，C取值相反，A和B相同，判定结果分别为0和1。

修订的条件/判定覆盖 MC/DC

- MC/DC：
  - 首先要满足C/DC（条件/判定覆盖），
  - 在此基础上，对于每一个条件C，要求存在两次计算以满足如下条件：
    当“条件C取值相反，其余条件取值不变”时，判定的结果值也相反。

- 理解：
  - 核心：每个条件都要独立影响判定结果。

  - 为什么是“两次计算”，而不是“两个用例”？
    - 当循环中有判定时，一个用例下同一判定可能被计算多次：
      若其中有两次计算满足上述条件，
      则这一个用例就能满足此判定的MC/DC。

修订的条件/判定覆盖 MC/DC

- MC/DC 包含（蕴含，Subsume）C/DC
- 条件组合覆盖 包含 MC/DC。
  - 条件组合覆盖：要求覆盖判定中所有条件取值的所有可能组合，需要大量的测试用例，实用性较差

- MC/DC具有条件组合覆盖的优势，同时大幅减少用例数。
  - 满足MC/DC的用例数下界为条件数+1，上界为条件数的两倍，
  - 例如，判定中有3个条件：
    - 条件组合覆盖需要8个用例，
    - MC/DC需要的用例数为4至6个。
  - 如果判定中条件很多，用例数的差别将非常大
  - 例如，判定中有10个条件：
    - 条件组合覆盖需要1024个用例
    - 而MC/DC只需要11至20个用例。

基本路径测试

- 下一节