

软件测试

(Software Testing)



outline

- 软件测试基础
- 软件测试策略
- 软件测试技术

软件测试概述

- 软件工程的其它阶段

- 构造出系统

- 软件测试

- 证明程序中有错误

“建设性”

“破坏性”

软件测试概述

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using **artificial data**.
- Can reveal the **presence** of errors NOT their **absence**.
- Testing is part of a more general **verification and validation** process, which also includes static validation techniques.

软件测试的目标

- 测试是为了发现程序中的错误而执行程序的过程；
- 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案；
- 成功的测试是发现了至今为止尚未发现的错误的测试。

----G. Myers

软件测试准则

- 所有测试都应该能追溯到用户需求
 - 最严重的错误是导致程序不能满足用户需求的那些错误
- 应该远在测试开始之前就制定出测试计划
 - 完成了需求模型就可以制定测试计划
- 把Pareto原理应用到软件测试中
 - 测试发现的错误中80%很可能是由程序中20%的模块造成的

软件测试准则

- 应该从“小规模”测试开始，并逐步进行“大规模”测试
 - 单个程序模块 → 集成的模块簇 → 整个系统
- 穷举测试是不可能的
 - 测试只能证明程序中有错误，不能证明程序中没有错误
- 为了达到最佳的测试效果，应该由独立的第三方从事测试工作
 - 开发软件的软件工程师不是完成全部测试工作的最佳人选

测试方法

黑盒测试(功能测试):

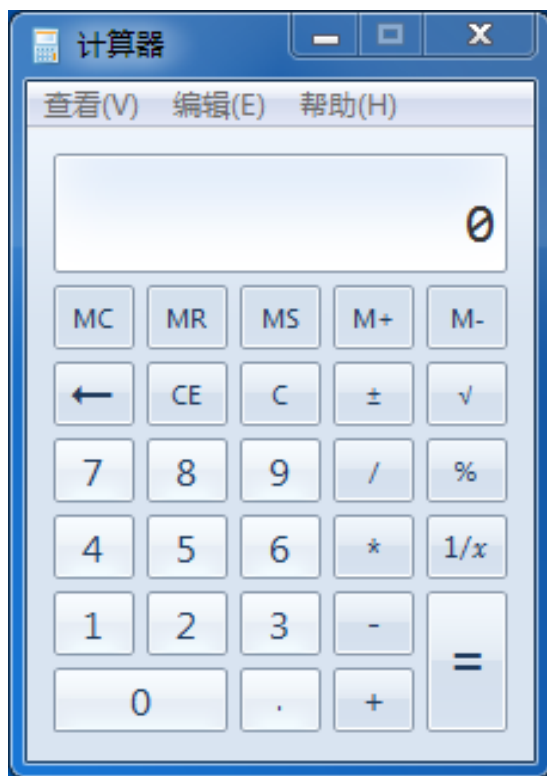
- 把程序看作一个黑盒子;
- 完全不考虑程序的内部结构和处理过程;
- 是在程序接口进行的测试。

白盒测试(结构测试):

- 把程序看成装在一个透明的盒子里;
- 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预定要求正确工作。

测试方法

黑盒测试不可能实现穷尽测试：

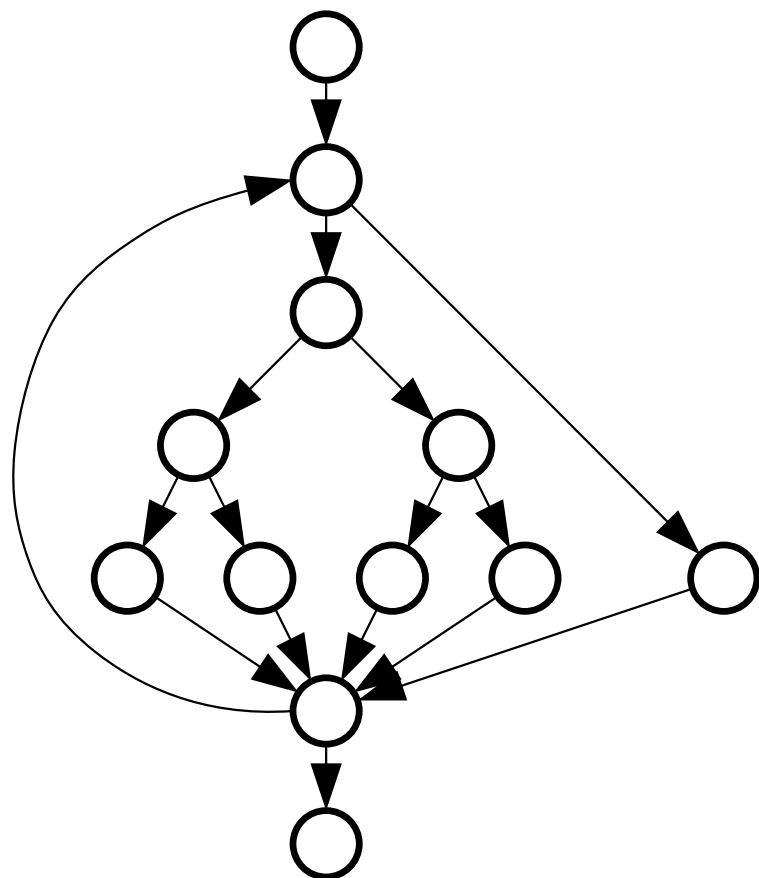


- $1+1$
- $1+2$
- ...
- $999999999999999999+999999999999999999$
- $0.1+0.1$
- ...
- 减法、乘法、除法、求平方根...
- ...

测试方法

白盒测试也不能实现穷尽测试：

- 图中所示的一个小程序的控制流程。曲线代表执行次数不超过20的循环，循环体中共有5条通路。
- 可能执行的路径有 5^{20} 条，近似为 10^{14} 条可能的路径。
- 如果完成一个路径的测试需要1毫秒，那么整个测试过程需要3170年。



outline

- 软件测试基础
- 软件测试策略
- 软件测试技术

问题

- 如何进行测试？
- 是否应该制定正式的测试计划？
- 应该将整个程序作为一个整体测试，还是应该只测试其中的一小部分？
- 当向一个大型系统加入新的构件时，对于已经做过的测试，是否还要重新测试？
- 什么时候需要客户参与测试工作？

制定测试策略（testing strategy）

测试策略

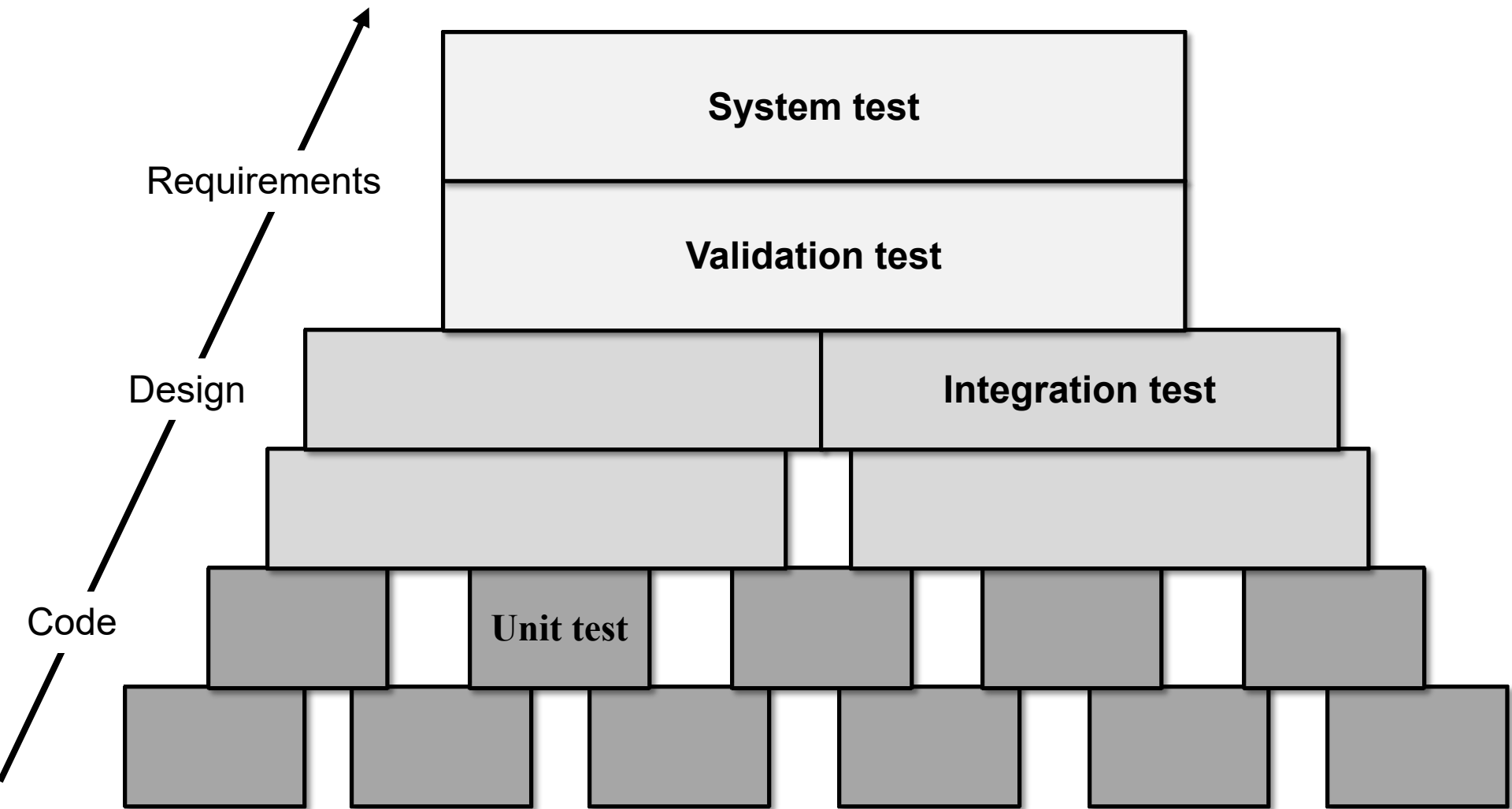
■ 指导测试活动的执行

- 测试步骤
- 何时计划测试、何时实施测试
- 需要多少成本、时间和资源

■ 软件测试的三种策略

- 在系统开发完成后对整个系统进行测试
- 采用增量的方式进行测试：先测各个模块，再测模块的集成，再测整个系统
- 每天系统完成部分功能后都进行测试

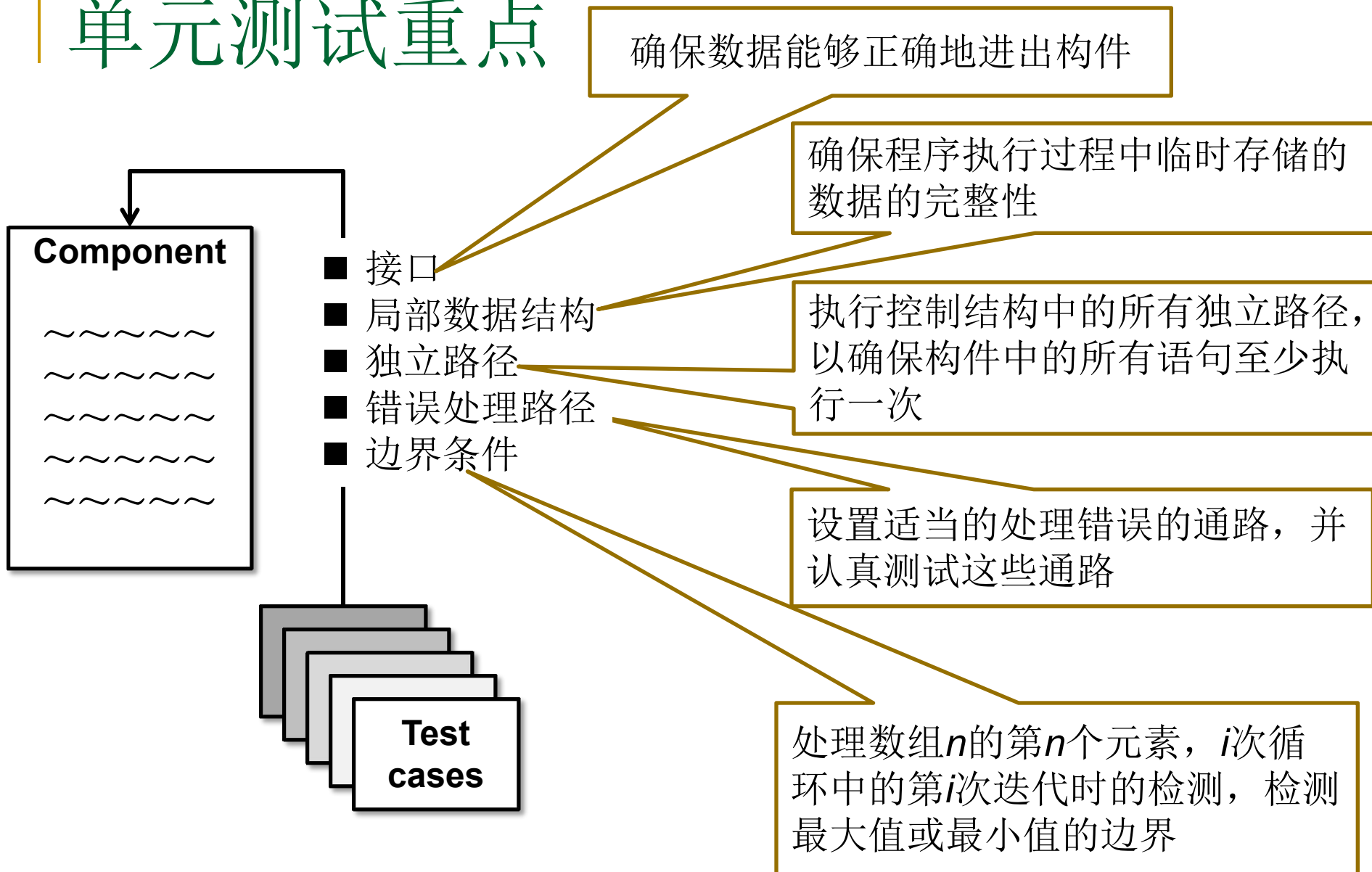
测试步骤



单元测试

- 单元测试关注单个构件或相关的一小组构件；
- 基于构件级设计，单元测试关注构件内部处理逻辑和数据结构。
- 单元测试和编码属于软件过程的同一个阶段；
- 可以应用人工测试和计算机测试这样两种不同类型的测试方法；
- 单元测试主要使用白盒测试技术，对多个模块的测试可以并行地进行。

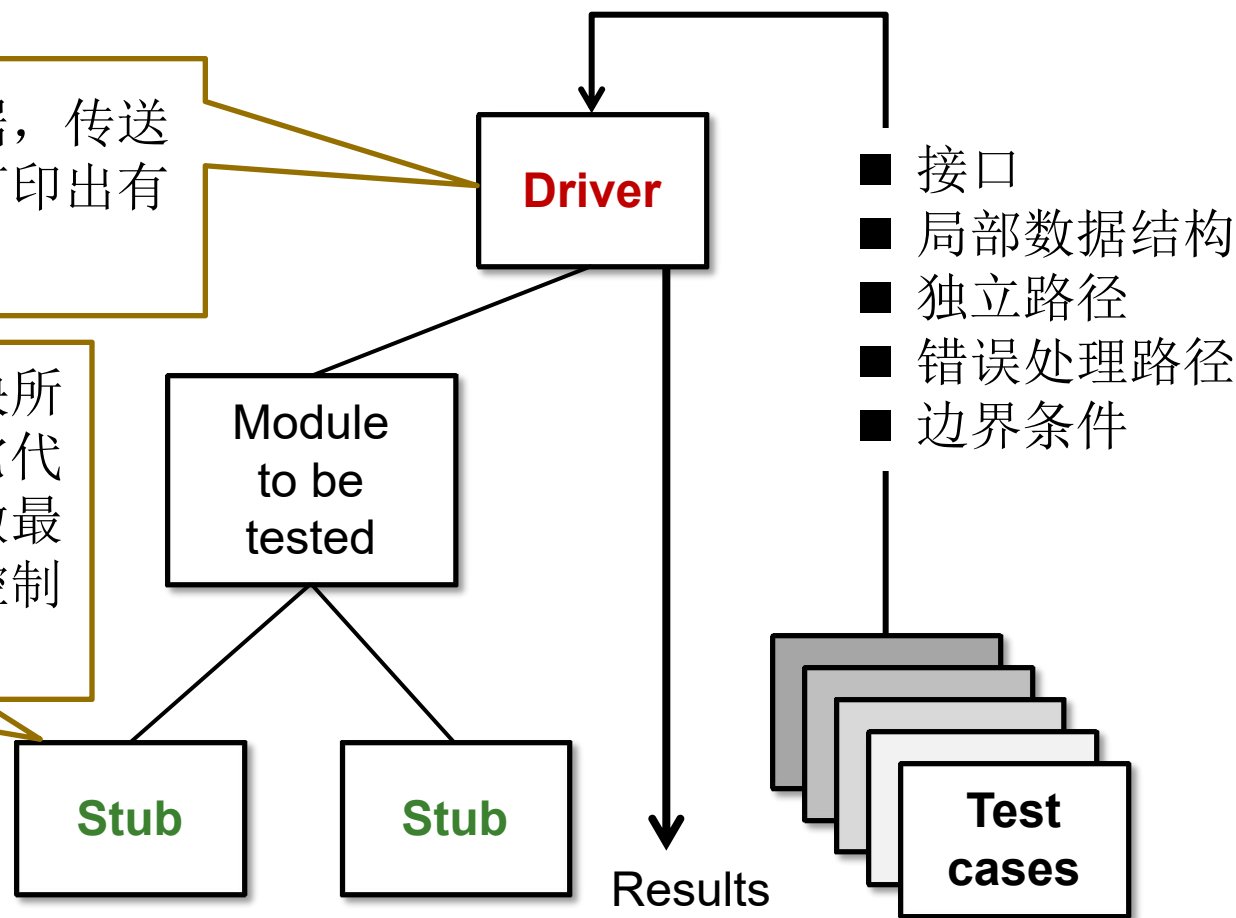
单元测试重点



单元测试过程

驱动程序：接收测试数据，传送给被测试的模块，并且打印出有关的结果。

存根程序：代替被测模块所调用的模块，它使用被它代替的模块的接口，可能做少量的数据操作，并把控制归还给调用它的模块。



注：驱动程序和存根程序代表测试开销，通常并不把它们作为软件产品的一部分交给用户。

自动化测试

- 自动化测试提高测试效率
- 利用自动化测试框架(JUnit)
 - 1997 年由 Erich Gamma 和 Kent Beck 共同开发完成
 - “在软件开发领域，从来就没有如此少的代码起到了如此重要的作用。” -----Martin Fowler
 - 使用JUnit进行TDD
 - 先测试后编码

梦想与现实



100% 自动化是软件测试的一个梦想！

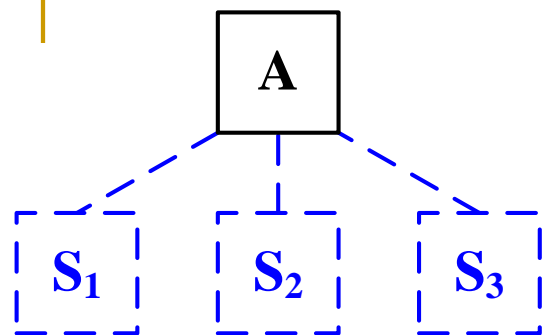
Software Testing Research: Achievements, Challenges, Dreams, A. Bertolino, In
Future of Software Engineering @ ICSE 2007

集成测试

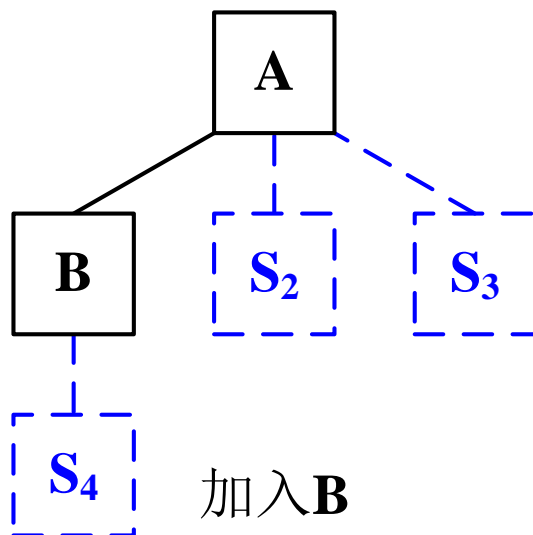
- 集成测试是测试和组装软件的系统化技术，主要目标是发现与接口有关的问题。
- 两种集成方法
 - 非渐增式集成
 - 把所有模块组合在一起，并把庞大的程序作为一个整体来测试。
 - 在庞大的程序中想要诊断定位一个错误是非常困难的，改正错误更是极端困难。
 - 渐增式集成
 - 把下一个要测试的模块同已经测试好的模块结合起来进行测试
 - 同时完成单元测试和集成测试
 - 把程序划分成小段来构造和测试，比较容易定位和改正错误
 - 两种集成策略：**自顶向下**和**自底向上**

自顶向下集成的步骤

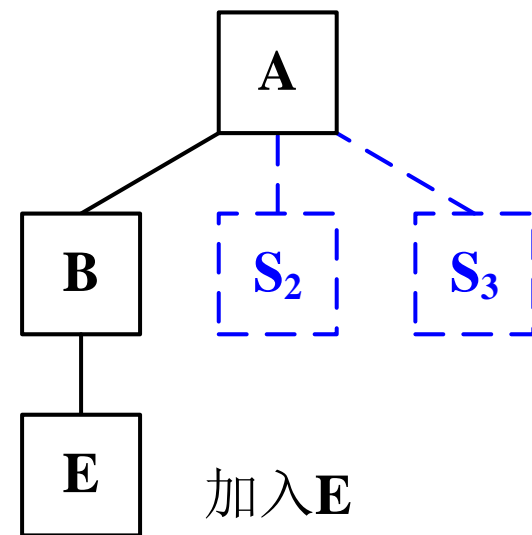
- ① 对主控模块进行测试，测试时用存根程序代替所有直接附属于主控制模块的模块；
- ② 根据所选择的集成方法，每次用实际模块替换一个存根程序；
- ③ 集成每个模块后都进行测试；
- ④ 为了保证加入模块没有引进新的错误，可能需要进行回归测试(全部或部分地重复以前做过的测试)。



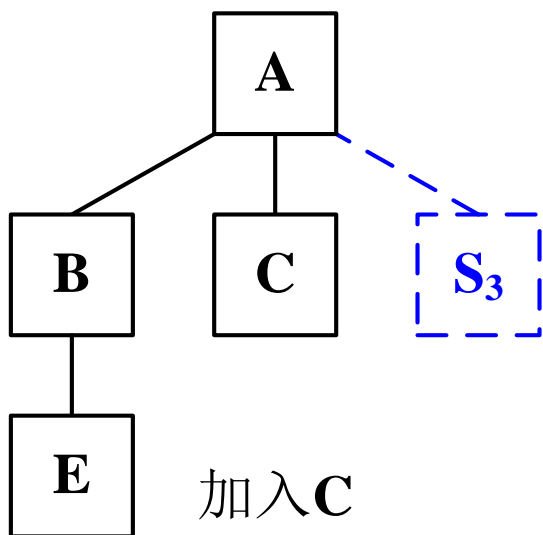
测试A



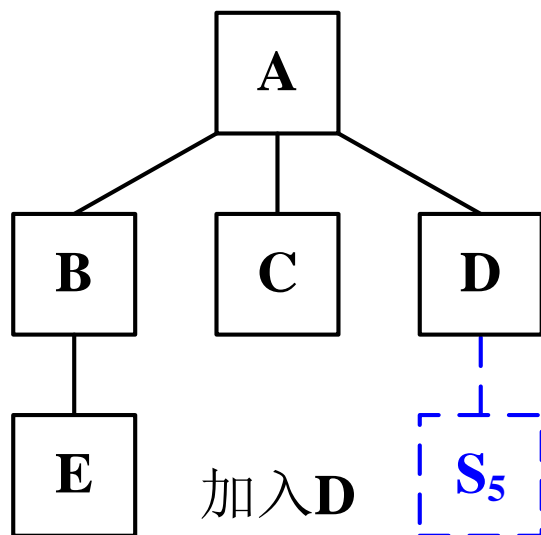
加入B



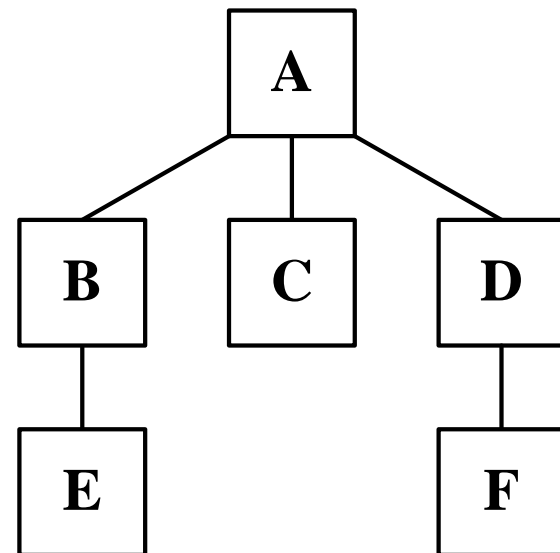
加入E



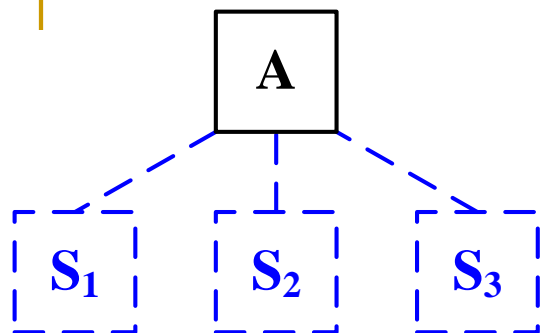
加入C



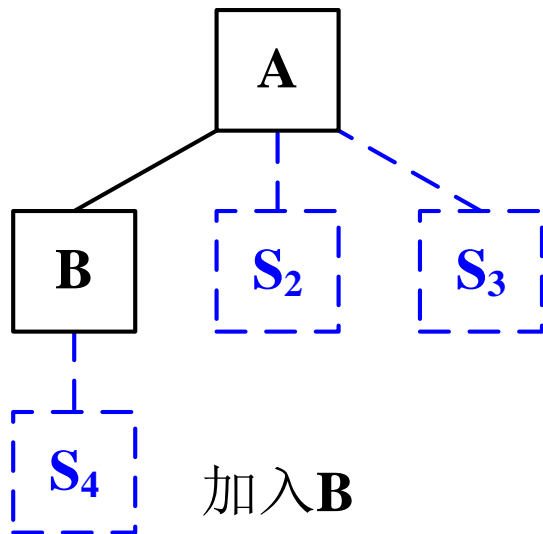
加入D



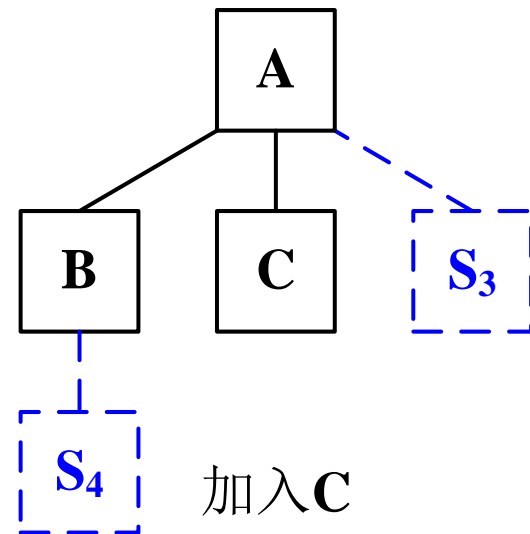
按深度优先策略集成



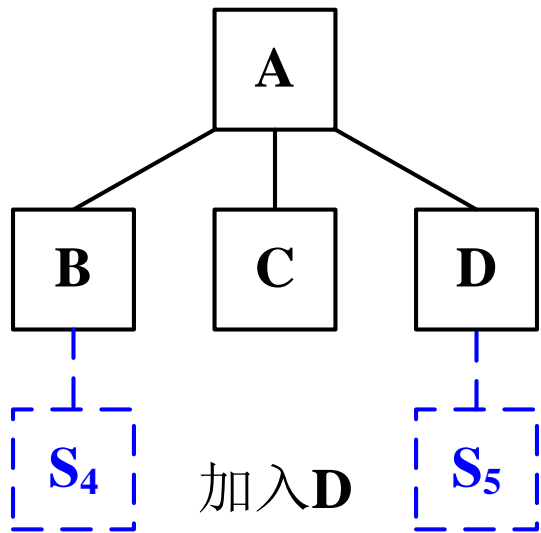
测试A



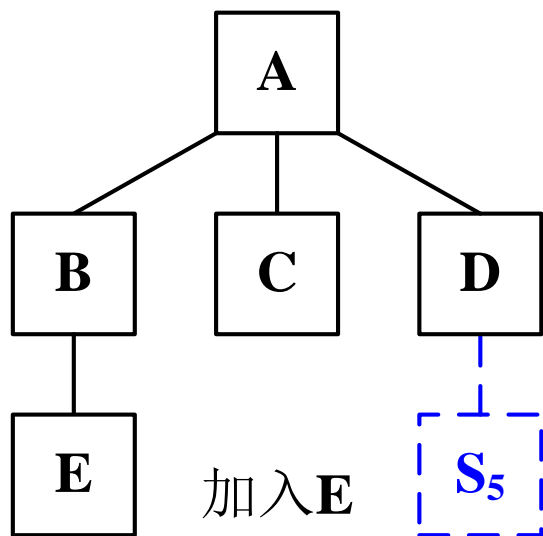
加入B



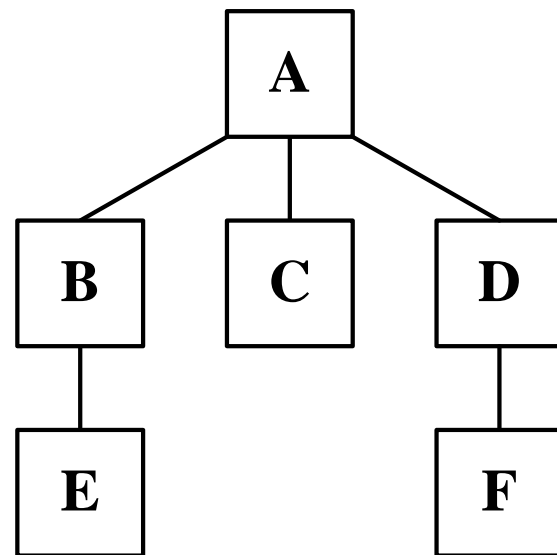
加入C



加入D




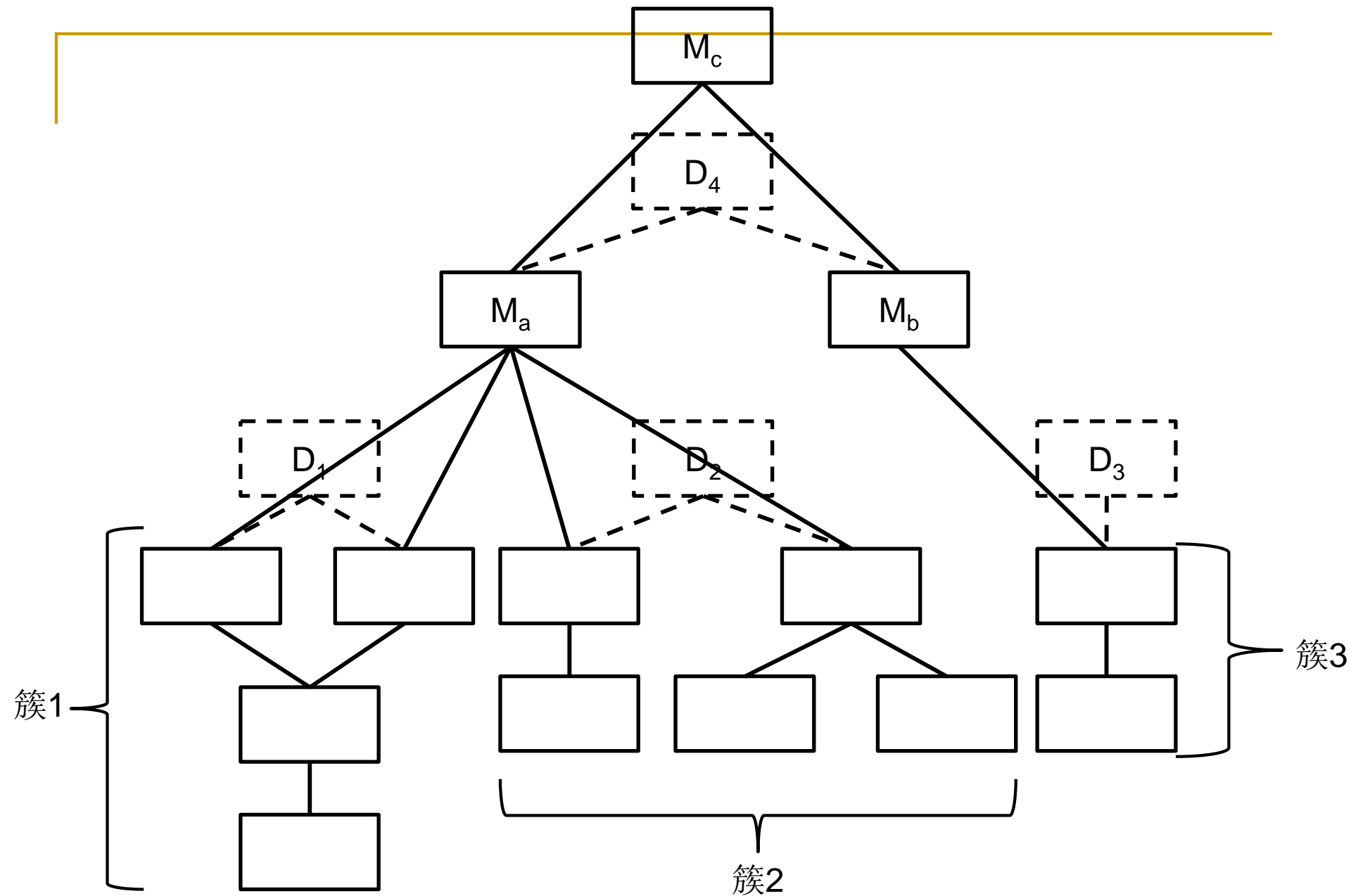
加入E



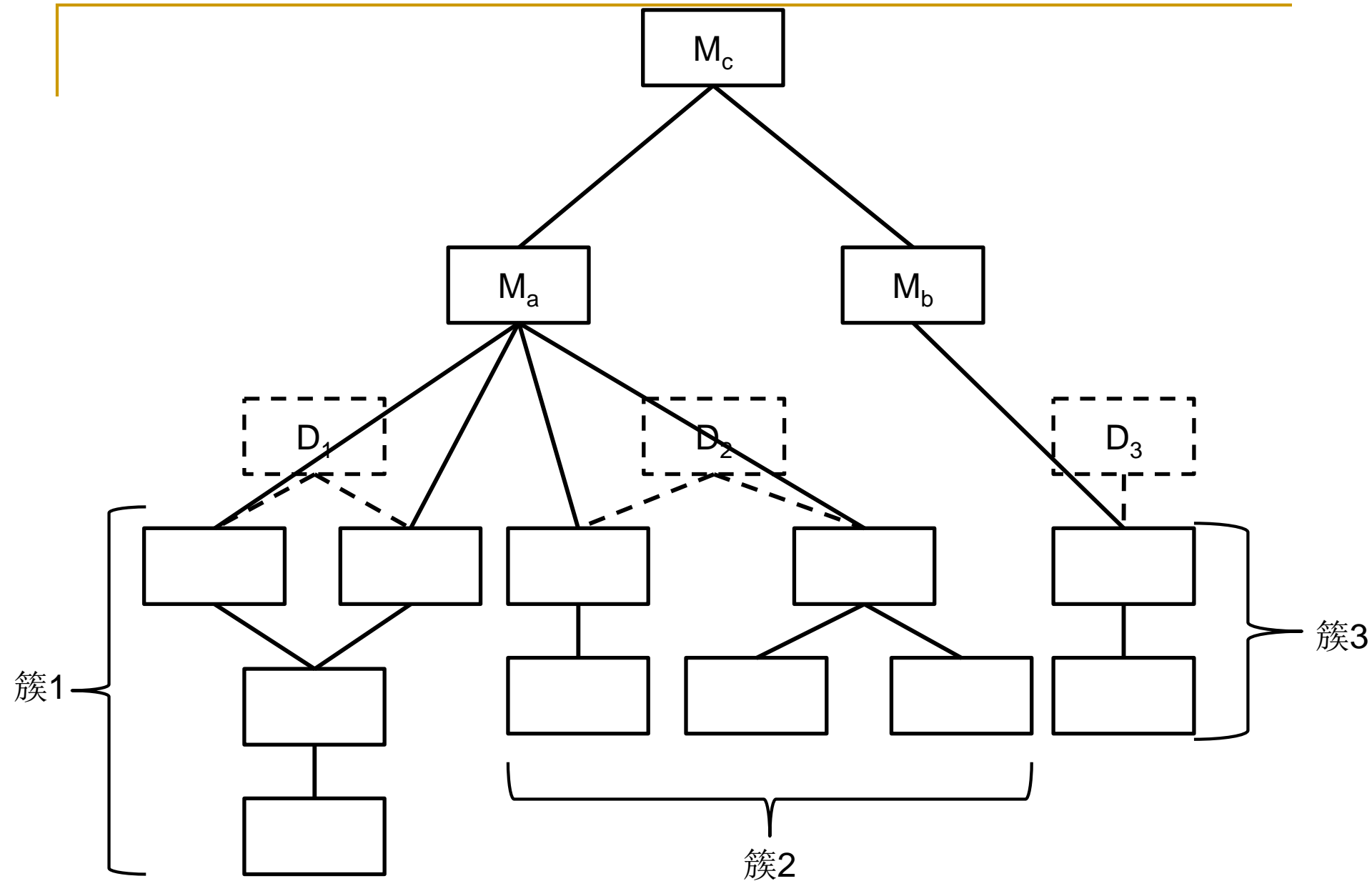
按宽度优先策略集成

自底向上集成的步骤

- ① 连接底层构件以构成完成特定子功能的簇；
 - ② 编写驱动模块以协调测试用例的输入和输出；
 - ③ 对由模块组成的子功能簇进行测试；
 - ④ 去掉驱动程序，沿着程序结构向上逐步连接簇。
- 



自底向上集成



混合两种集成策略

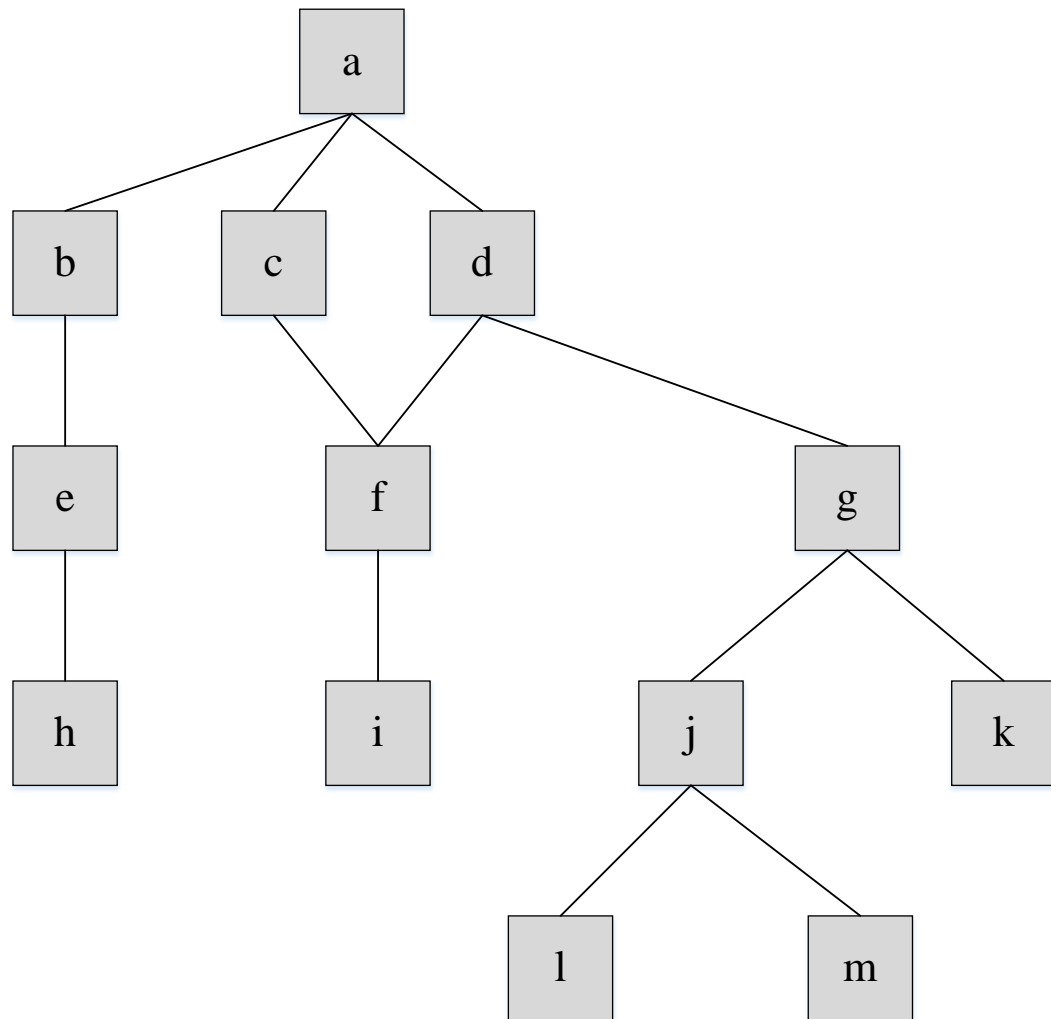
讨论

- 13个构件，使用自底向上的集成策略进行集成测试时的集成顺序？

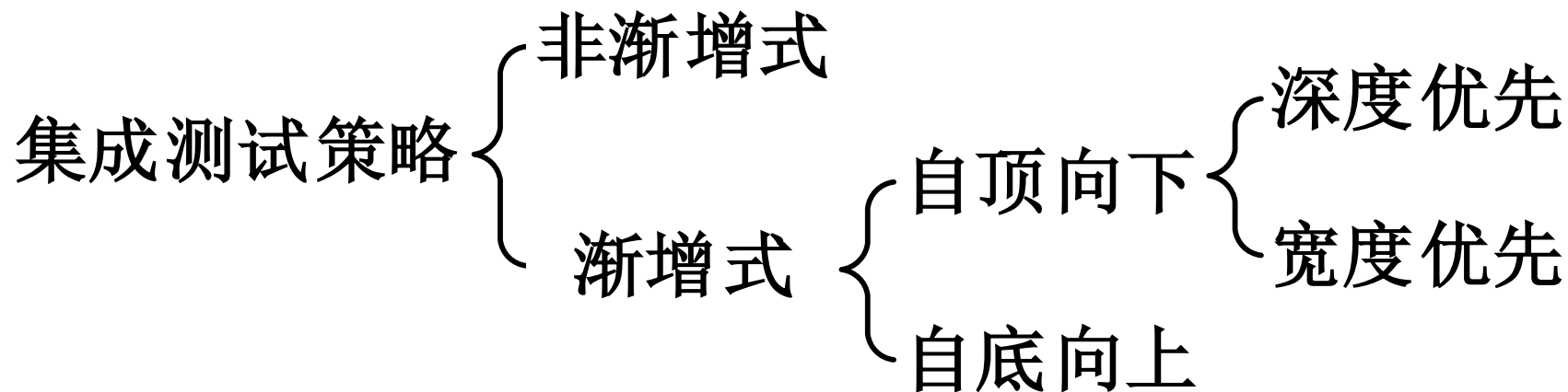
- l,m,h,i,j,k,e,f,g,b,c,d,a

- 如果由一个3人组成的测试小组进行集成测试，如何进行分工？

- 1: h, e, b
 - 2: i, f, c
 - 3: l, m, j, k, g, d
 - 1 or 2 or 3: a



集成测试策略小结



回归测试

- 回归测试是指重新执行已测试过的某些子集，以确保变更没有传播不期望的副作用。
- 回归测试用于保证由于测试或其他原因引起的变更，不会导致非预期的软件行为或额外错误。
- 手工
 - 重新执行所有测试用例的子集
- 自动
 - 捕捉/回放工具：捕捉在测试过程中的测试用例，以后重复执行这个过程。

确认测试

- 确认测试必须有用户积极参与，或者以用户为主进行。
- 软件配置复查（Configuration Review）
 - 保证软件配置的所有成分都齐全
 - 质量符合要求
 - 文档与程序完全一致
 - 具有完成软件维护所必须的细节
 - 编好目录

确认测试

■ Alpha测试

- 由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试
- 在受控的环境中进行

■ Beta测试

- 验收测试（acceptance testing）
- 由软件的最终用户们在一个或多个客户场所进行
- 开发者通常不在现场
- 用户记录测试过程中遇到的问题，并报告给开发人员

outline

- 软件测试基础
- 软件测试策略
- 软件测试技术

测试方案

- 所谓**测试方案**包括具体的测试目的（预定要测试的具体功能），应该输入的测试数据和预期的结果。
- 把测试数据和预期的输出结果称为**测试用例**。



测试用例设计技术

■ 白盒测试

- 逻辑覆盖
- 基本路径测试
-

■ 黑盒测试

- 等价划分
- 边界值分析
- 组合测试
-

逻辑覆盖

- 有选择地执行程序中某些最有代表性的通路是对穷尽测试的惟一可行的替代办法。
- 从覆盖源程序语句的详尽程度分析，大致有以下一些不同的覆盖标准：
 - 语句覆盖
 - 判定覆盖
 - 条件覆盖
 - 判定/条件覆盖
 - 条件组合覆盖

逻辑覆盖

1. 语句覆盖

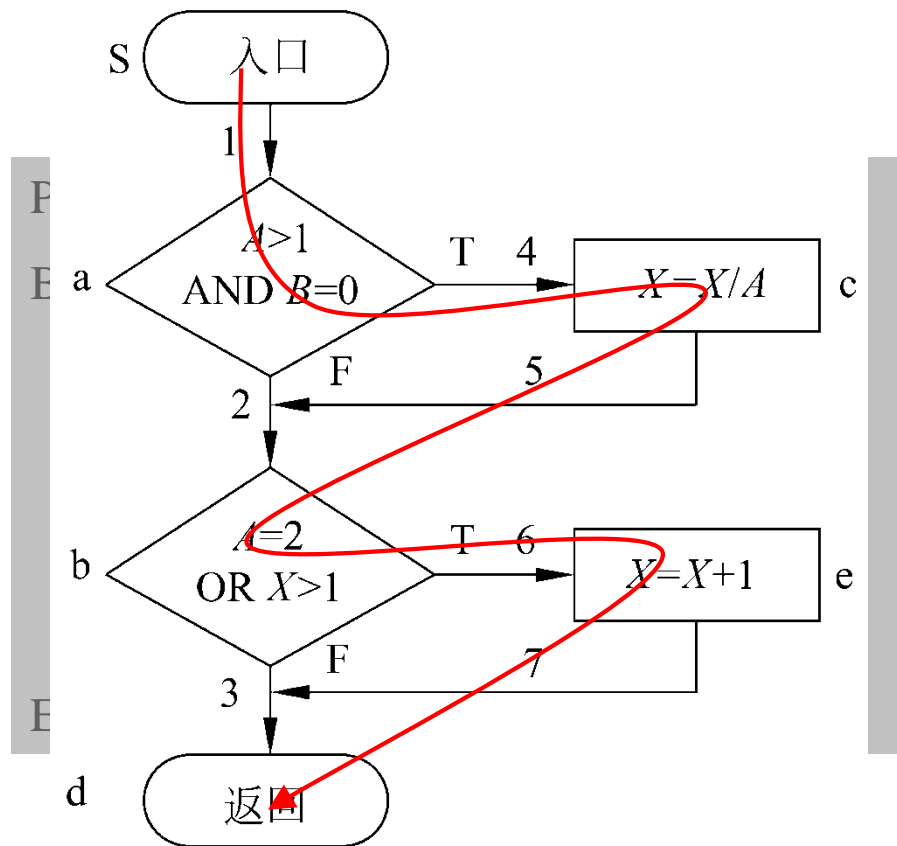
- 含义：选择足够多的测试数据，使被测程序中每个语句至少执行一次。

- 分析：
执行路径sacbed

- 测试用例：

A=2, B=0, X=4

覆盖sacbed



逻辑覆盖

2. 判定覆盖（分支覆盖）

- 含义：不仅每个语句必须至少执行一次，而且**每个判定的每种可能的结果都应该至少执行一次**。

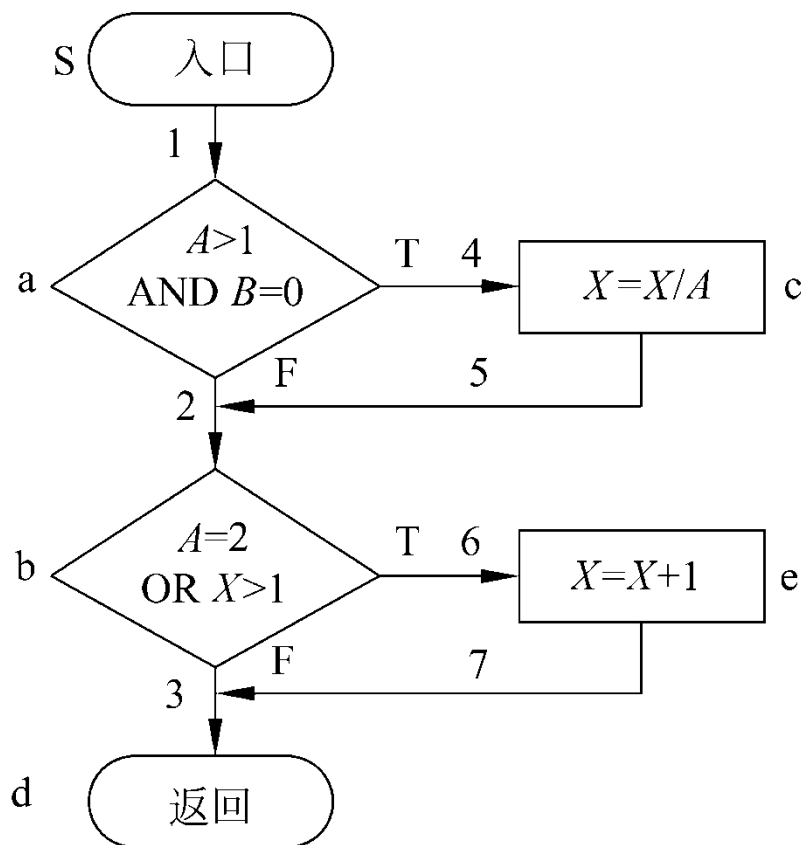
- 所有判定分支：

(1) a点判定为T

(2) a点判定为F

(3) b点判定为T

(4) b点判定为F



■ 所有判定分支:

(1) a点判定为T

(2) a点判定为F

(3) b点判定为T

(4) b点判定为F

■ 测试用例:

I. 满足(1)(4)

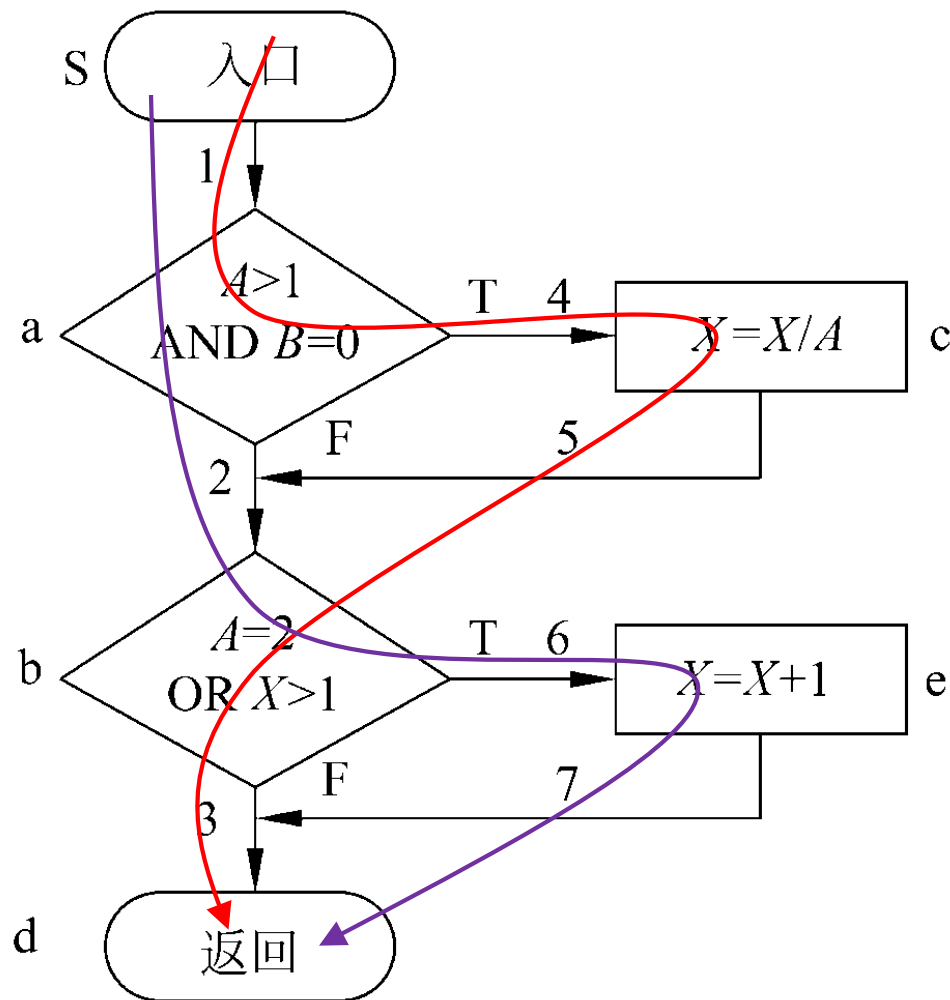
A=3, B=0, X=-3

覆盖sacbd

II. 满足(2)(3)

A=2, B=1, X=1

覆盖sabed



■ 所有判定分支:

(1) a点判定为T

(2) a点判定为F

(3) b点判定为T

(4) b点判定为F

■ 或者

I. 满足(1)(3)

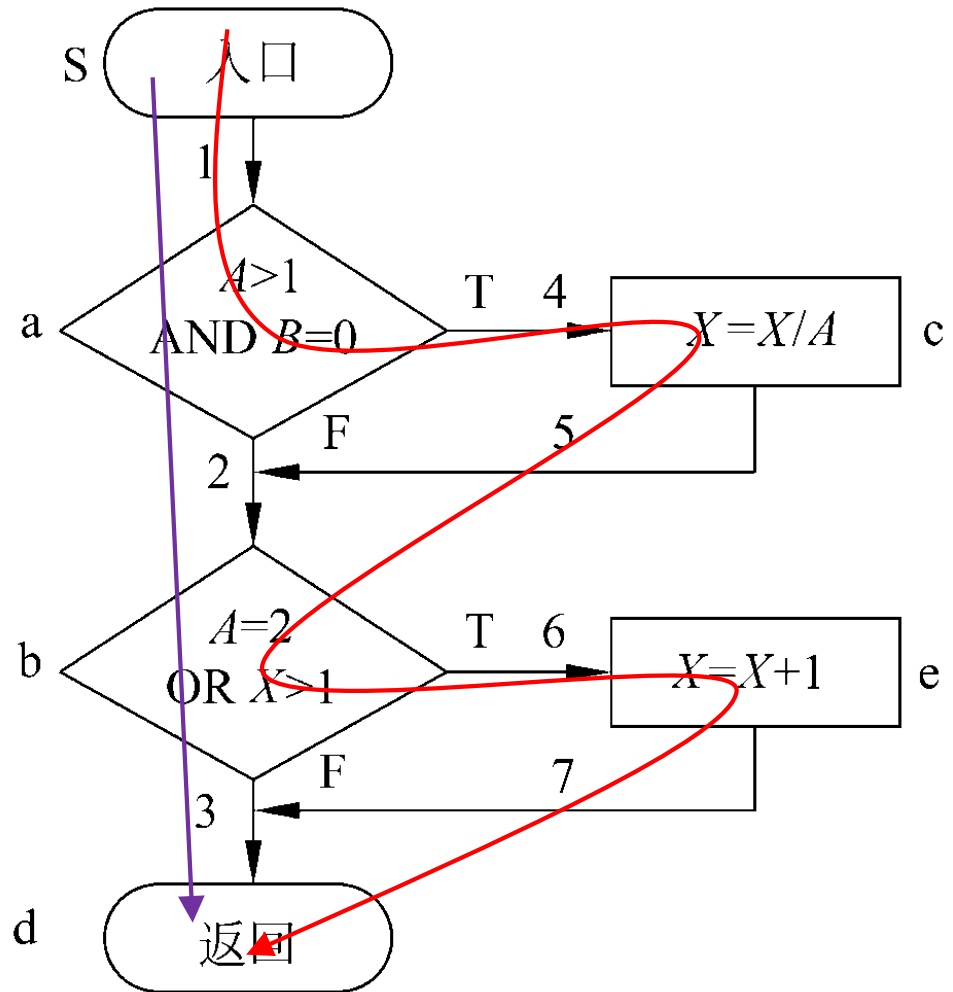
$A=2, B=0, X=3$

覆盖sacbed

II. 满足(2)(4)

$A=3, B=1, X=1$

覆盖sabd



逻辑覆盖

3. 条件覆盖

- 含义：不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。

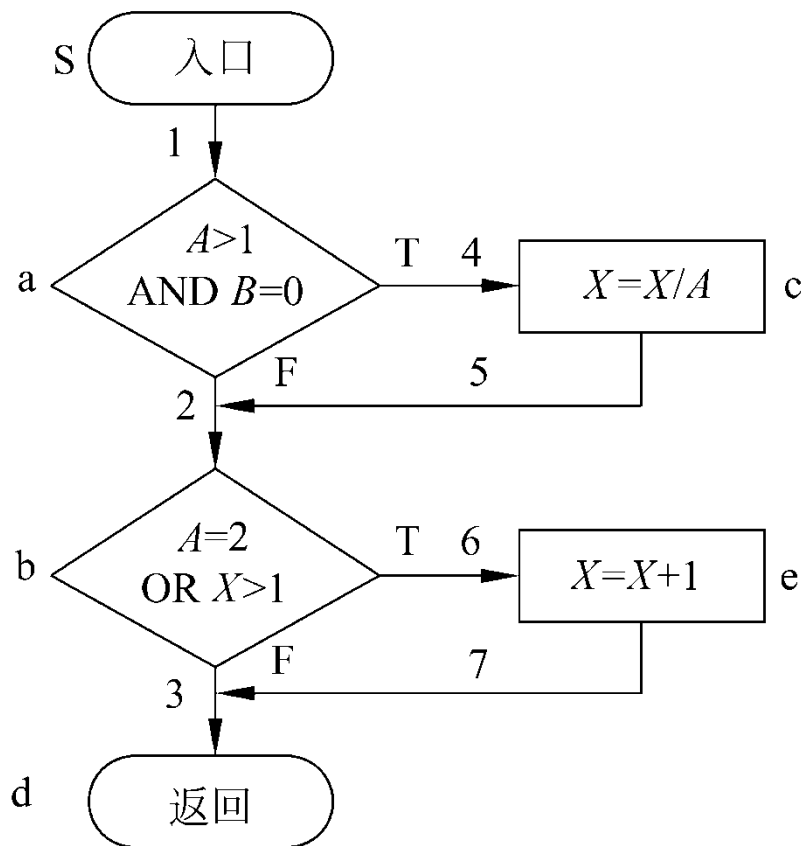
- 所有条件：

(1) $A > 1$ (2) $A \leq 1$

(3) $B = 0$ (4) $B \neq 0$

(5) $A = 2$ (6) $A \neq 2$

(7) $X > 1$ (8) $X \leq 1$



■ 所有条件:

(1) $A > 1$ (2) $A \leq 1$

(3) $B = 0$ (4) $B \neq 0$

(5) $A = 2$ (6) $A \neq 2$

(7) $X > 1$ (8) $X \leq 1$

■ 测试用例:

I. 满足(1)(3)(5)(7)

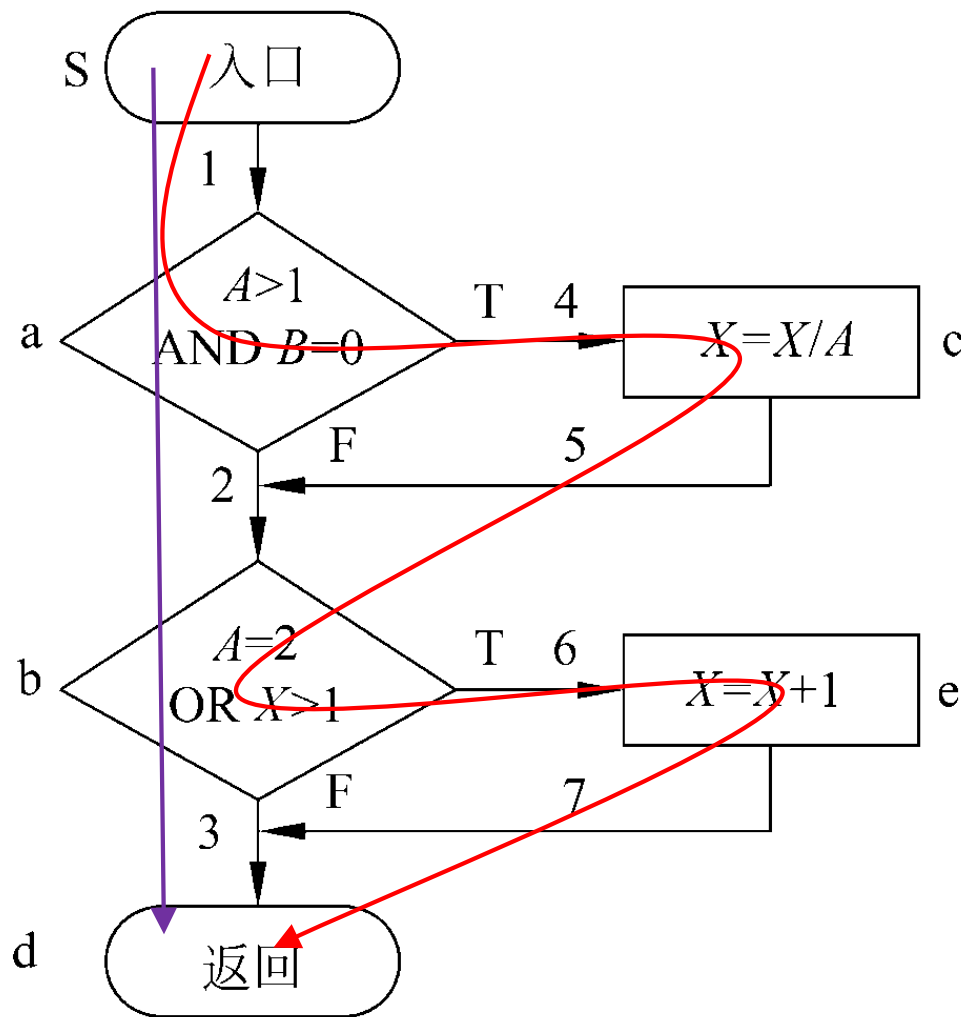
$A=2, B=0, X=4$

覆盖sacbed

II. 满足(2)(4)(6)(8)

$A=1, B=1, X=1$

覆盖sabd



■ 所有条件:

(1) $A > 1$ (2) $A \leq 1$

(3) $B = 0$ (4) $B \neq 0$

(5) $A = 2$ (6) $A \neq 2$

(7) $X > 1$ (8) $X \leq 1$

■ 或者

I. 满足(1)(3)(5)(8)

$A = 2, B = 0, X = 1$

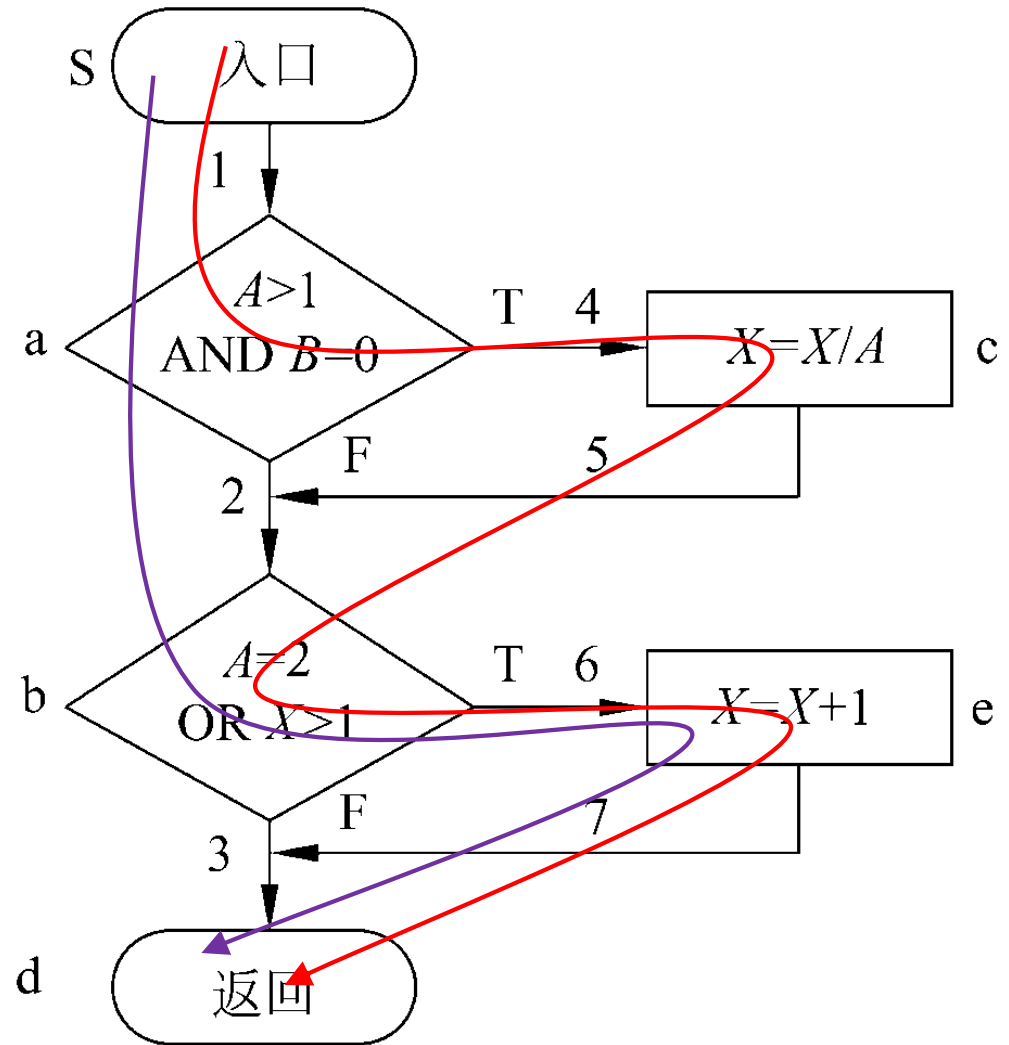
覆盖sacbed

II. 满足(2)(4)(6)(7)

$A = 1, B = 1, X = 2$

覆盖sabcd

■ 或者.....



4. 判定/条件覆盖

- 含义：使得判定表达式中的每个条件都取到各种可能的值，每个判定表达式也都取到各种可能的结果。
- 所有判定分支：
 - (1) a点判定为T
 - (2) a点判定为F
 - (3) b点判定为T
 - (4) b点判定为F
- 所有条件：
 - (1) $A > 1$
 - (2) $A \leq 1$
 - (3) $B = 0$
 - (4) $B \neq 0$
 - (5) $A = 2$
 - (6) $A \neq 2$
 - (7) $X > 1$
 - (8) $X \leq 1$

■ 所有判定分支:

- (1) a点判定为T (2) a点判定为F
(3) b点判定为T (4) b点判定为F

■ 所有条件:

- (1) $A > 1$ (2) $A \leq 1$
(3) $B = 0$ (4) $B \neq 0$
(5) $A = 2$ (6) $A \neq 2$
(7) $X > 1$ (8) $X \leq 1$

■ 测试用例:

I. 满足条件(1)(3)(5)(7)和判定(1)(3)

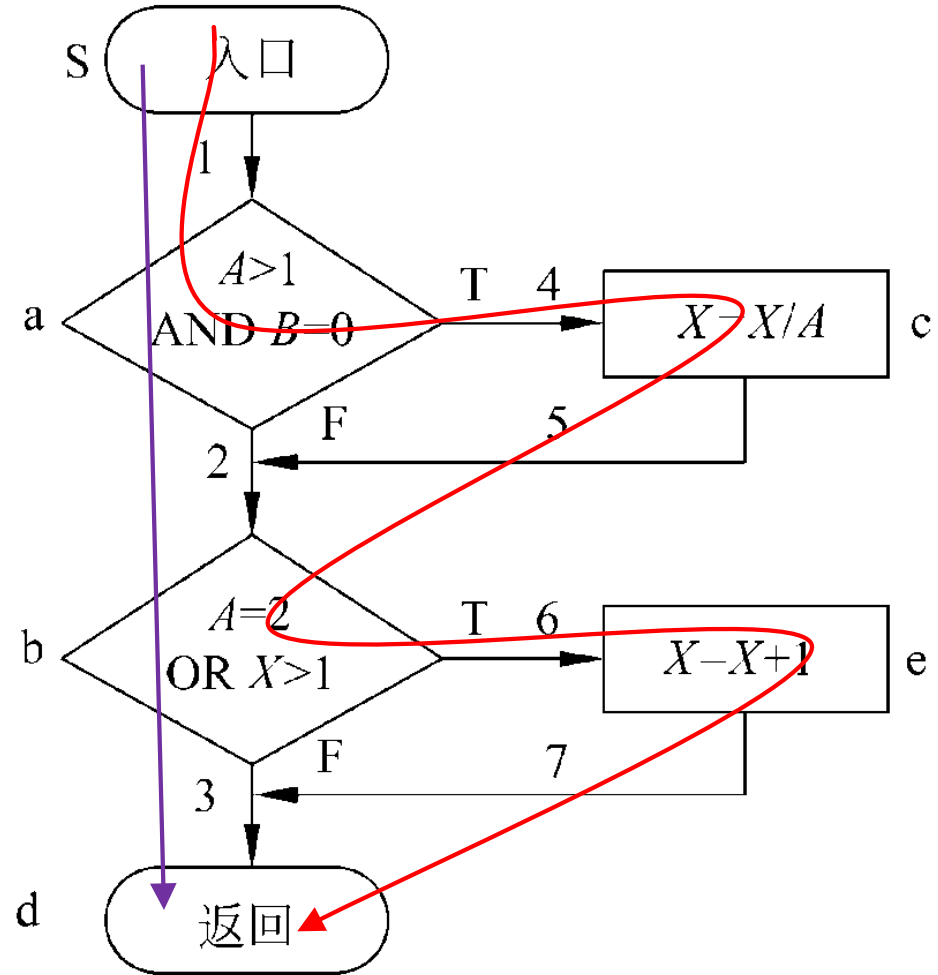
$A=2, B=0, X=4$

覆盖sacbed

II. 满足条件(2)(4)(6)(8)和判定(2)(4)

$A=1, B=1, X=1$

覆盖sabd



5. 条件组合覆盖

- 含义：要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。

- 条件组合：

(1) $A > 1$, $B = 0$

(2) $A > 1$, $B \neq 0$

(3) $A \leq 1$, $B = 0$

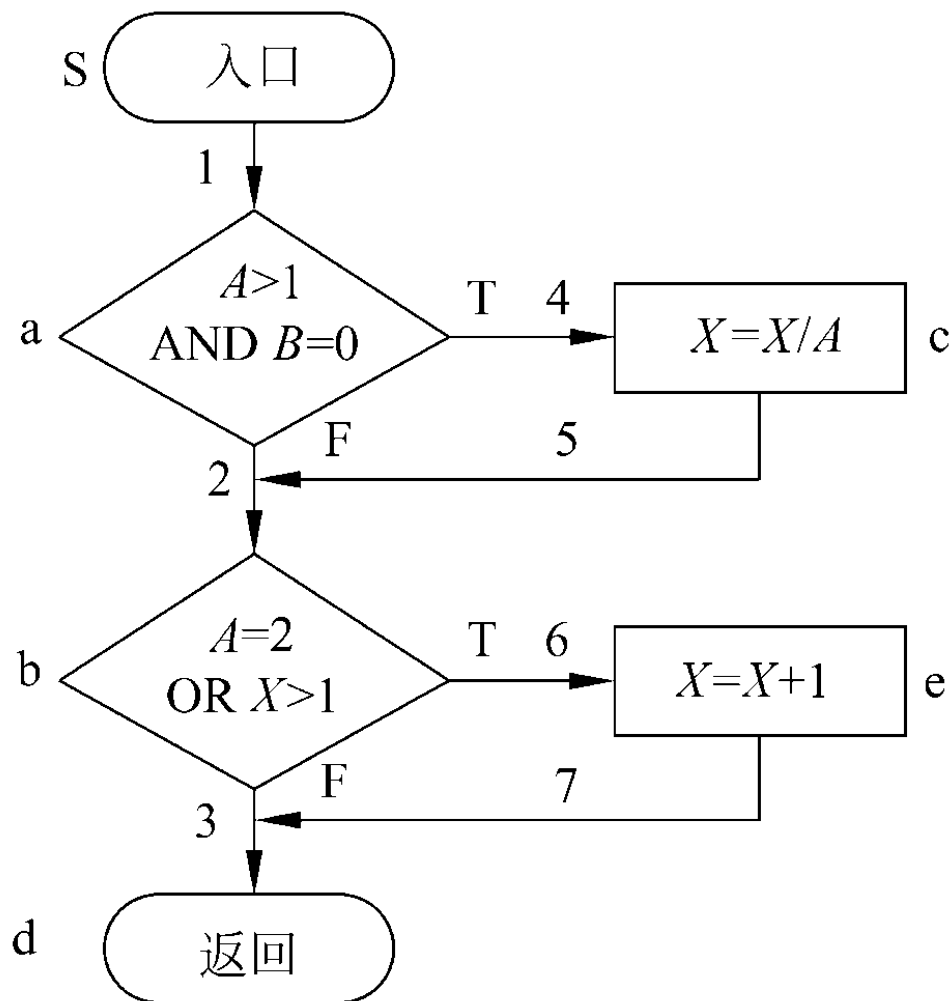
(4) $A \leq 1$, $B \neq 0$

(5) $A = 2$, $X > 1$

(6) $A = 2$, $X \leq 1$

(7) $A \neq 2$, $X > 1$

(8) $A \neq 2$, $X \leq 1$



条件组合:

(1) $A > 1, B = 0$ (2) $A > 1, B \neq 0$

(3) $A \leq 1, B = 0$ (4) $A \leq 1, B \neq 0$

(5) $A = 2, X > 1$ (6) $A = 2, X \leq 1$

(7) $A \neq 2, X > 1$ (8) $A \neq 2, X \leq 1$

测试用例:

I. 满足(1)(5)

$A = 2, B = 0, X = 4$

覆盖sacbed

II. 满足(2)(6)

$A = 2, B = 1, X = 1$

覆盖sabcd

III. 满足(3)(7)

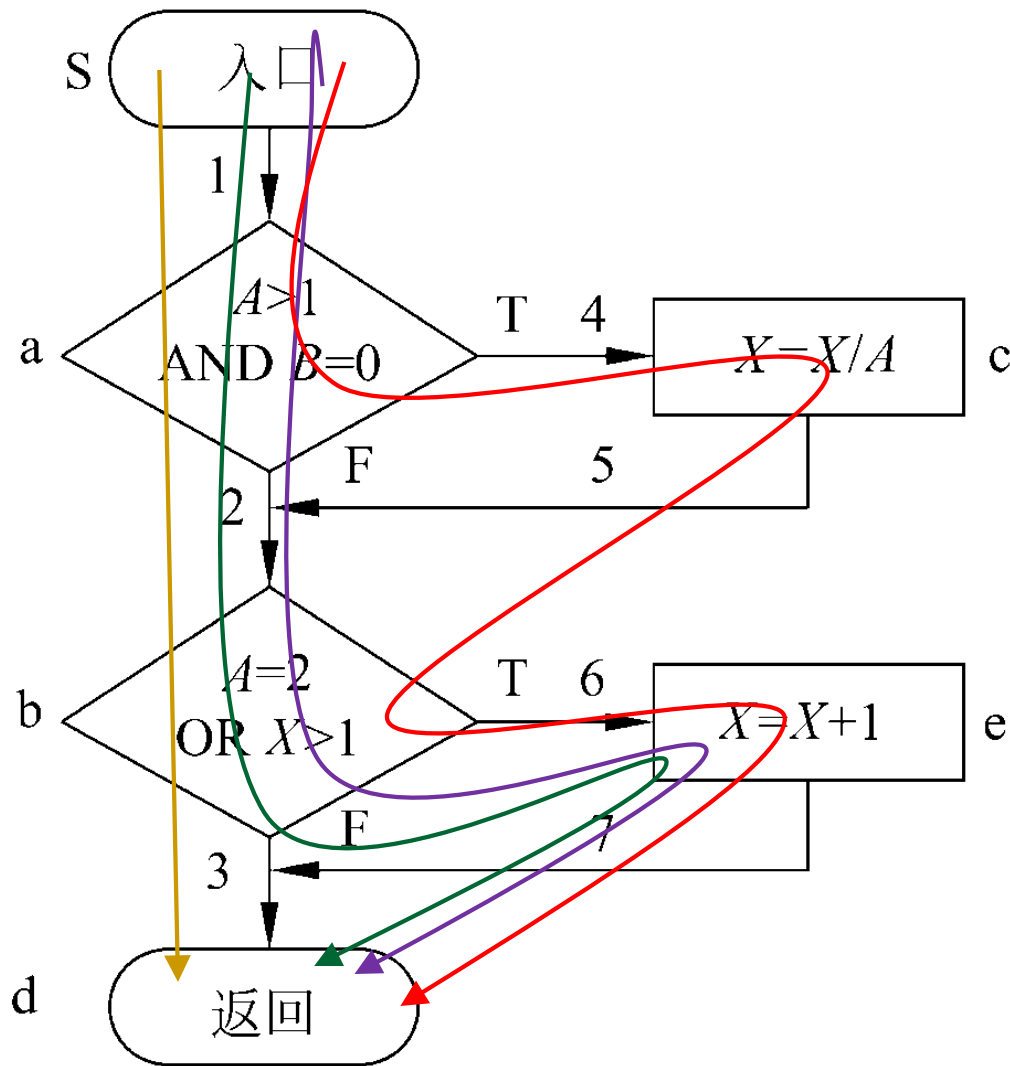
$A = 1, B = 0, X = 2$

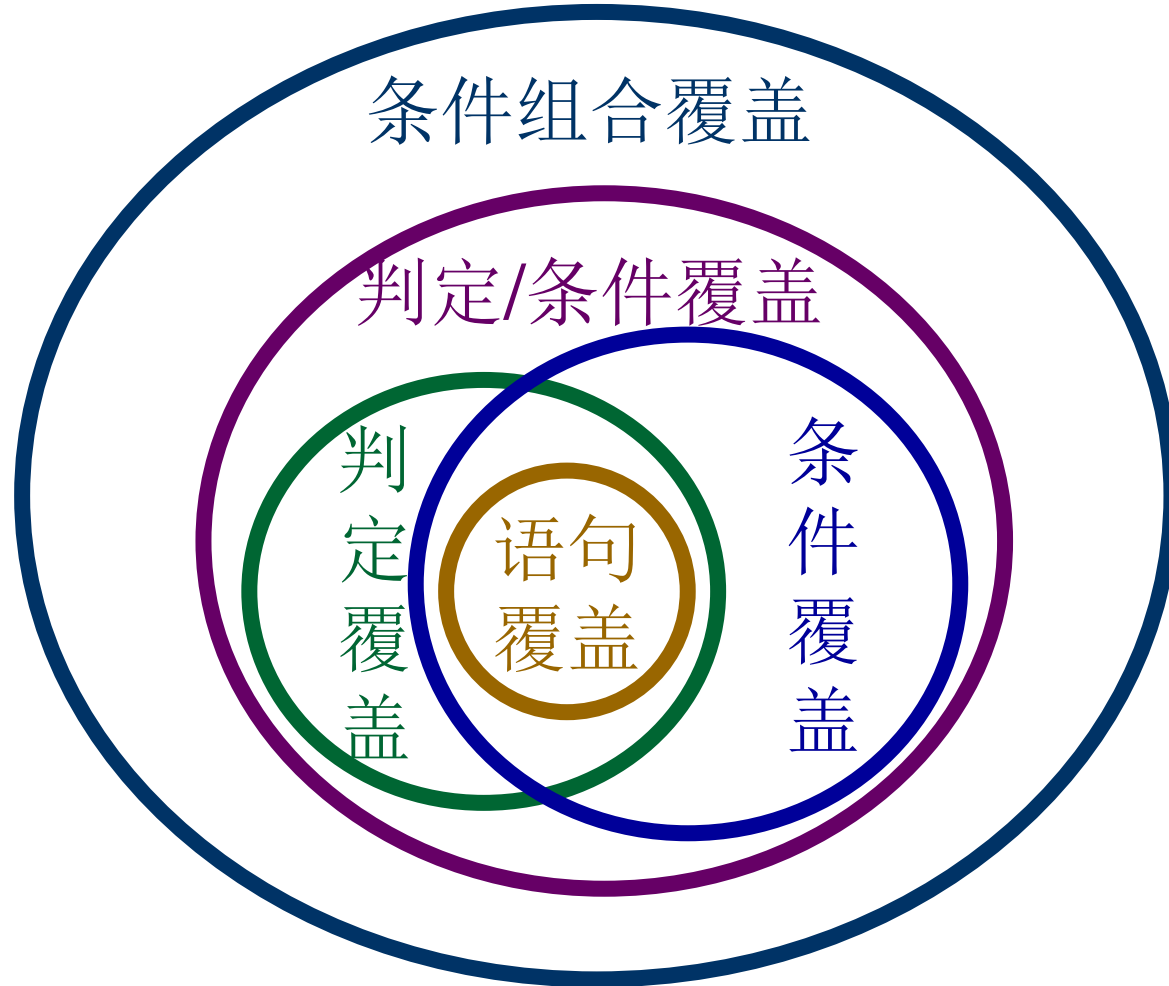
覆盖sabcd

IV. 满足(4)(8)

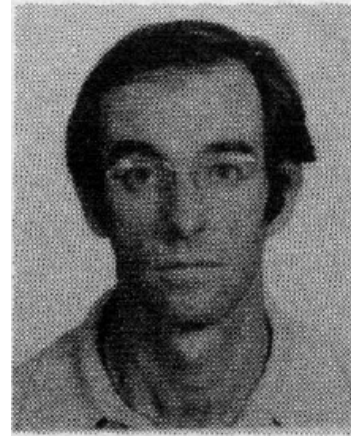
$A = 1, B = 1, X = 1$

覆盖sabd



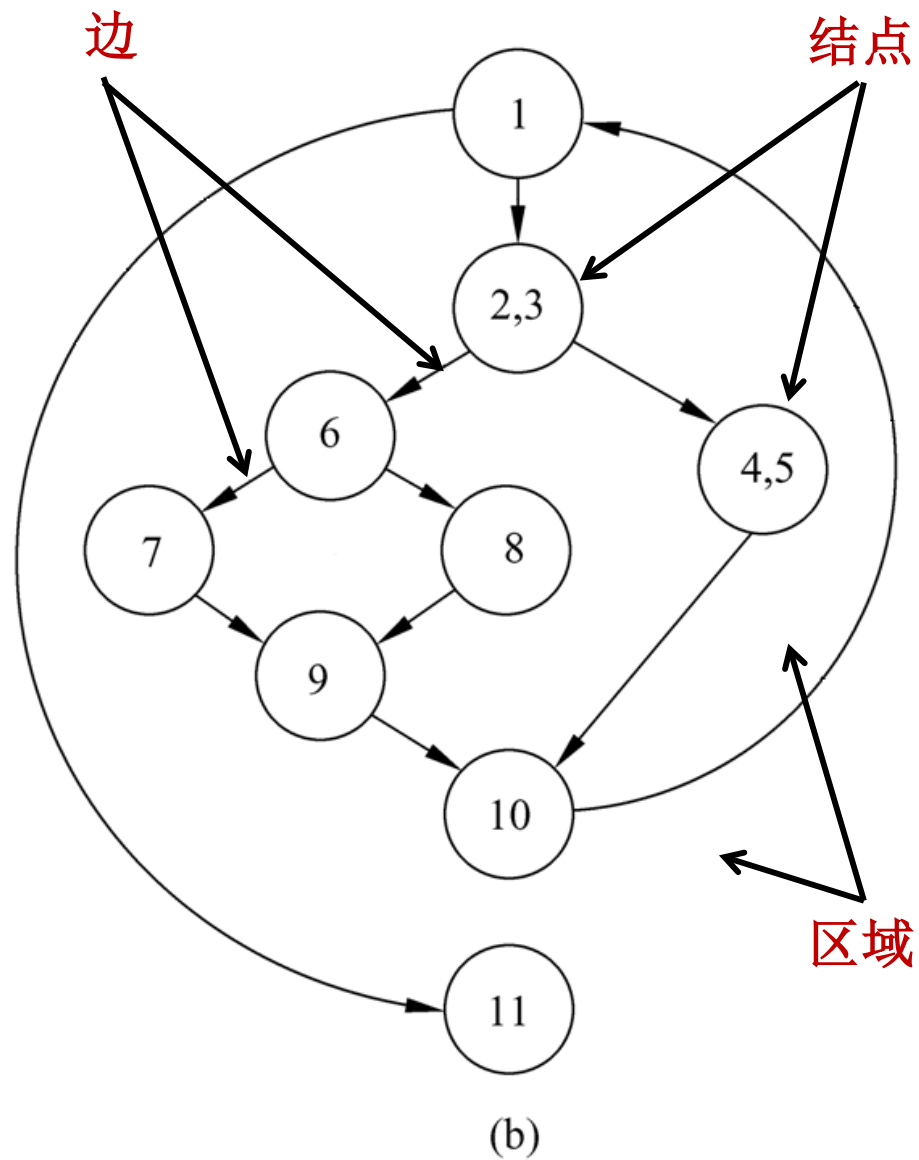
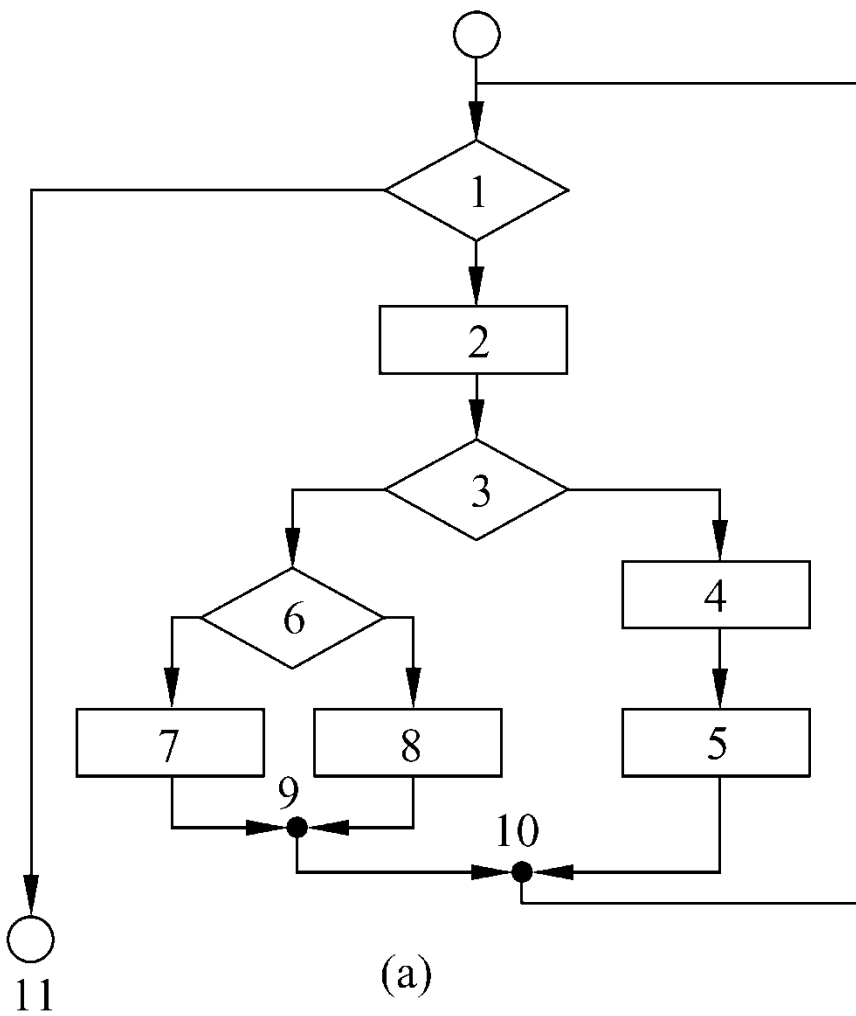


基本路径测试（Basic Path Testing）



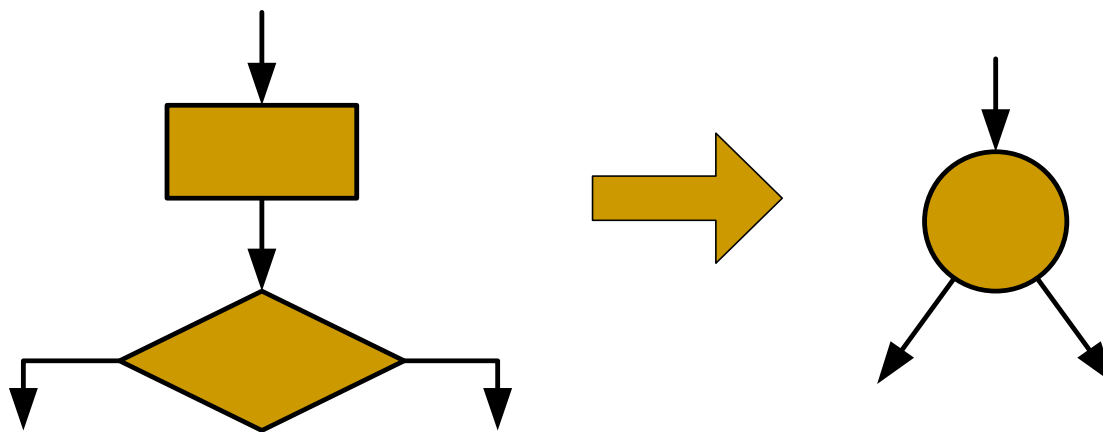
- Tom McCabe提出的一种白盒测试技术
 - 画出流图（Flow graph）
 - 计算环形复杂度
 - 以该复杂度为指南确定独立路径的基本集合
 - 从该基本集合导出测试用例

流图

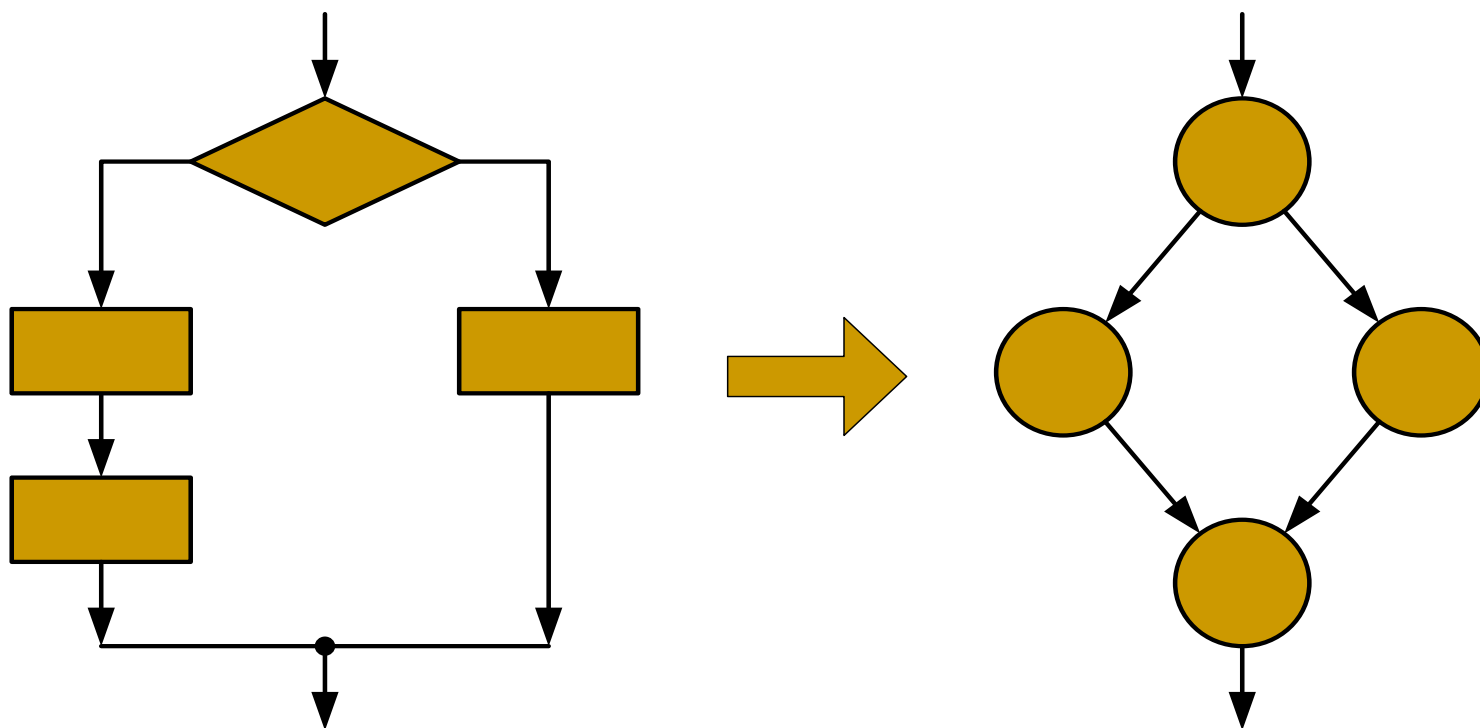


映射方法:

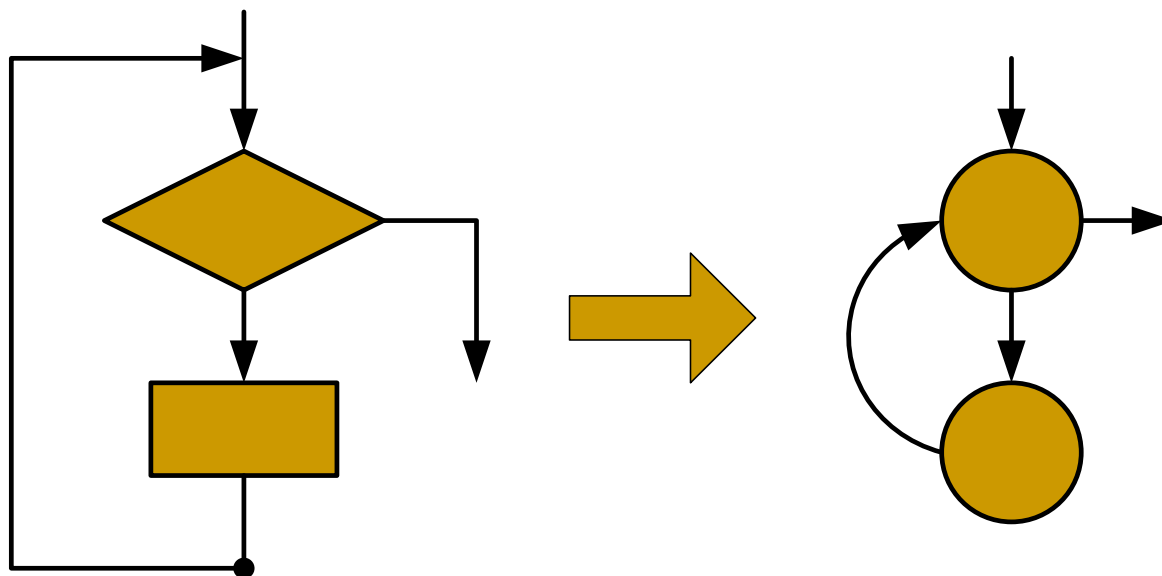
- 任何方法表示的过程设计结果，都可以翻译成流图。
- 对于顺序结构，一个顺序处理序列和下一个选择或循环的开始语句，可以映射成流图中的一个结点。



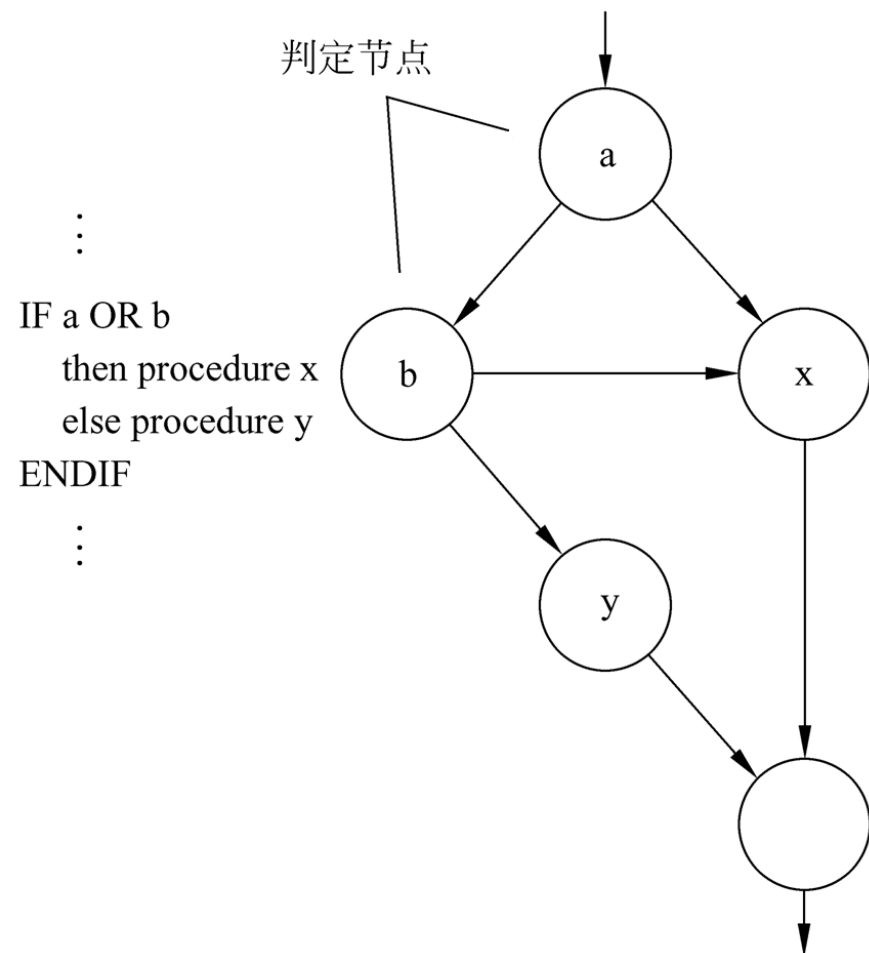
- 对于选择结构，开始语句映射成一个结点；两条分支至少各映射成一个结点；结束映射成一个结点。



- 对于循环结构，开始和结束语句各映射成一个结点。



- 当过程设计中包含复合条件时，把复合条件分解为若干个简单条件，每个简单条件对应流图中一个结点。



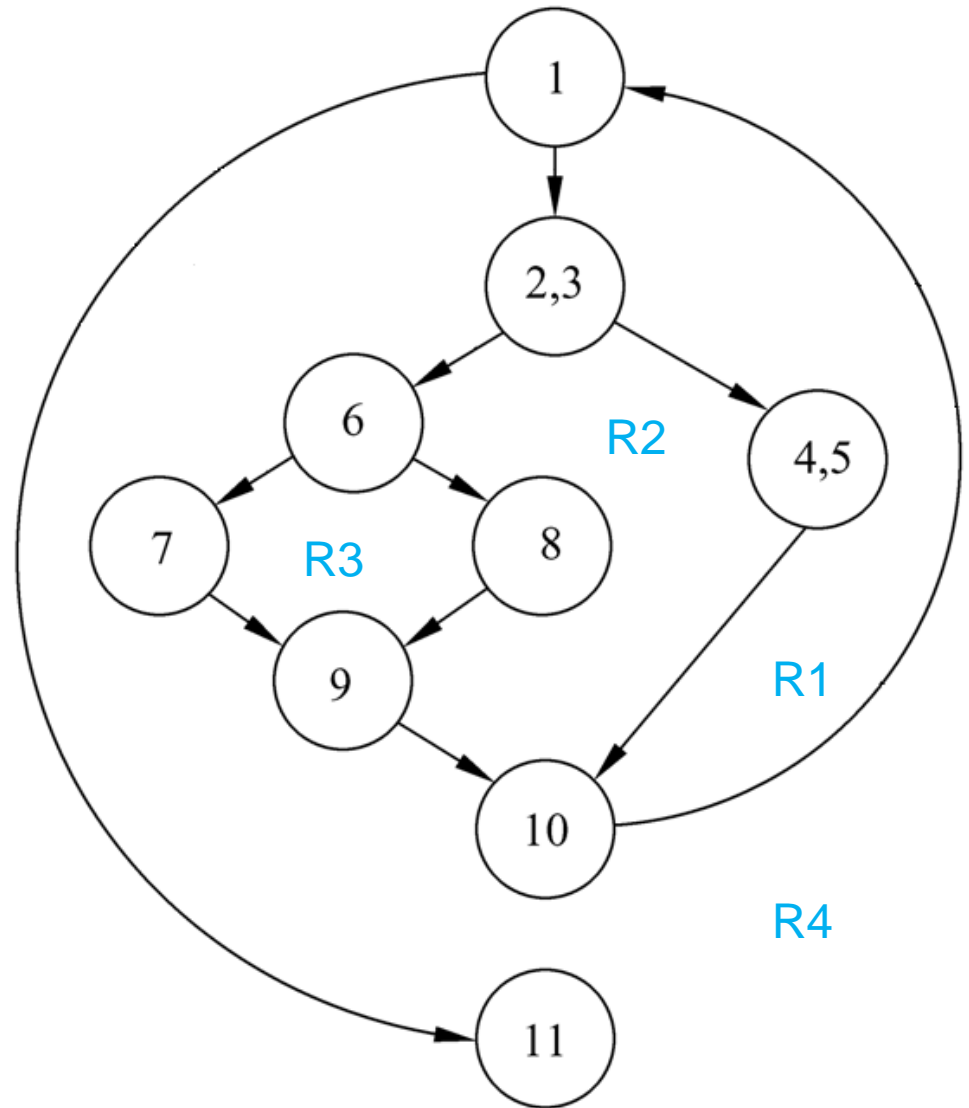
基本路径测试步骤

- 画出流图（Flow graph）
- 计算环形复杂度
- 以该复杂度为指南确定独立路径的基本集合
- 从该基本集合导出测试用例

环形复杂度

- 定量度量程序的逻辑复杂度
- 环形复杂度 $V(G)$ 计算方法
 - $V(G)$ =流图中的区域数
 - $V(G)=E-N+2$ ，其中 E 是流图中的边数， N 是结点数
 - $V(G)=P+1$ ，其中 P 是流图中判定结点的数目

- $V(G) = \text{区域数}$
 $= 4$
- $V(G) = E - N + 2$
 $= 11 - 9 + 2 = 4$
- $V(G) = P + 1$
 $= 3 + 1 = 4$



基本路径测试步骤

- 画出流图（Flow graph）
- 计算环形复杂度
- 以该复杂度为指南确定独立路径的基本集合
- 从该基本集合导出测试用例

独立路径

- 至少包含一条在定义该路径之前不曾用过的边
- 环形复杂度用于定义程序基本路径集合中独立路径的数量

基本路径集合:

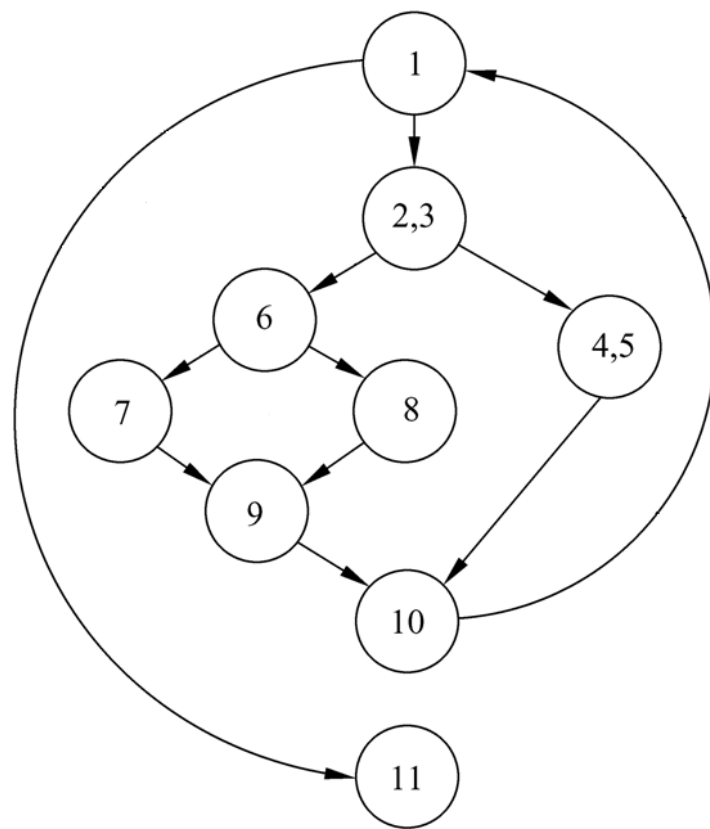
路径1: 1—11

路径2: 1—2—3—4—5 —10—1—11

路径3: 1—2—3—6—8 —9 —10—1—11

路径4: 1—2—3—6—7 —9 —10—1—11

1—2—3—4—5 —10—1—2—3—6—8 —9 —10—1—11



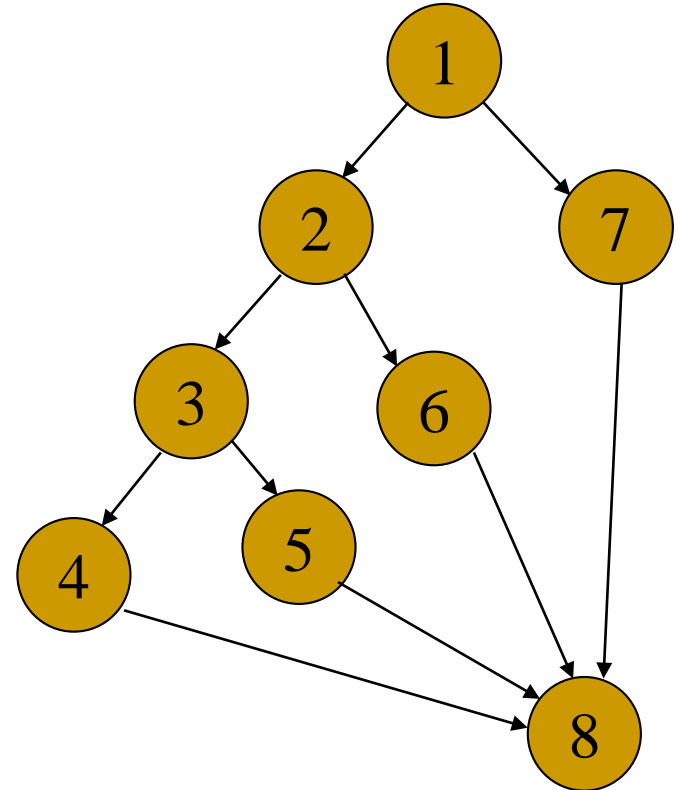
基本路径测试步骤

- 画出流图（Flow graph）
- 计算环形复杂度
- 以该复杂度为指南确定独立路径的基本集合
- 从该基本集合导出测试用例
 - 强制执行基本集合中的每条路径

例：判断闰年

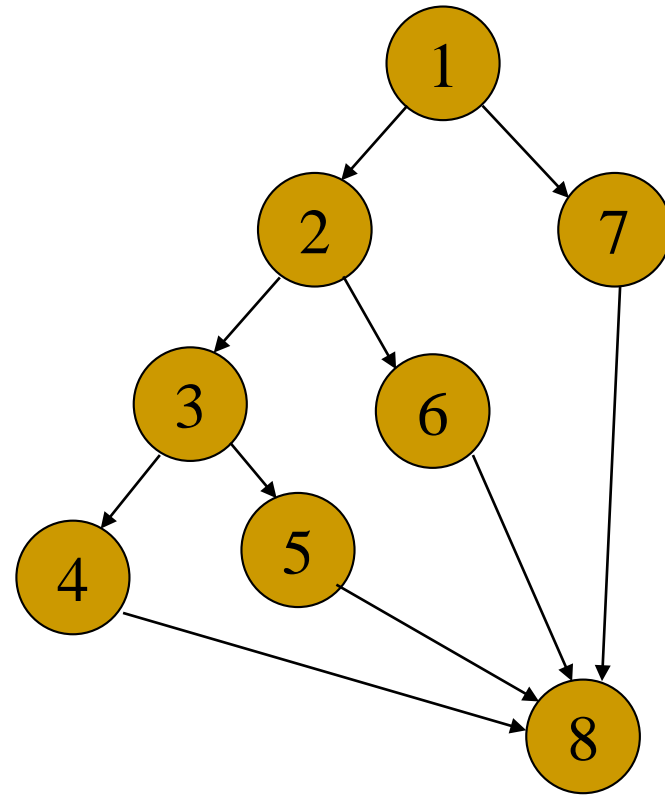
(1) 根据过程设计结果画出相应的流图

```
def IsLeap(int year):  
1:   if (year % 4 == 0):  
2:       if (year % 100 == 0):  
3:           if (year % 400 == 0):  
4:               leap = 1  
5:           else: leap = 0  
6:       else: leap = 1  
7:   else: leap = 0  
8:   return leap
```



(2) 计算流图的环形复杂度。

$$V(G)=10-8+2=4$$



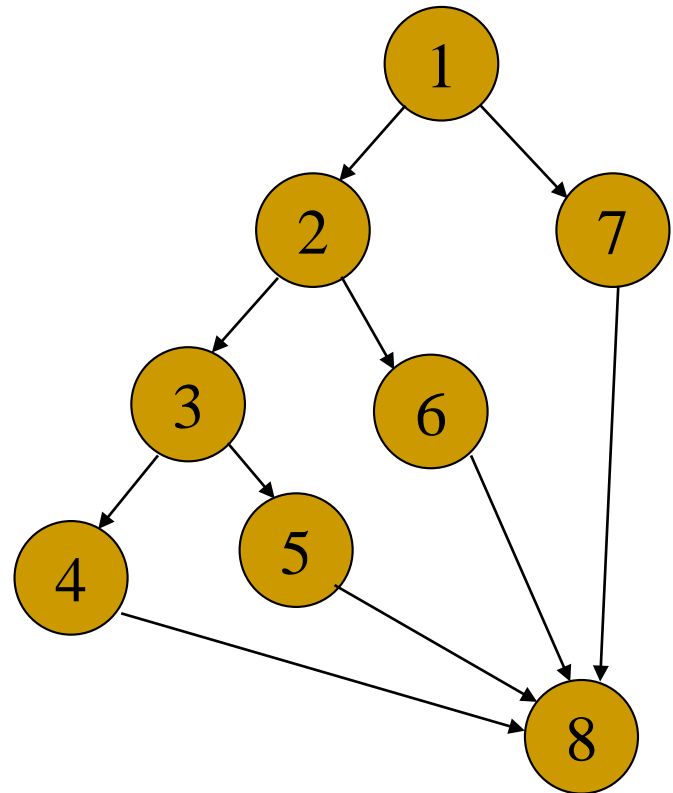
(3) 确定独立路径的基本集合

路径1: 1—7—8

路径2: 1—2—6—8

路径3: 1—2—3—5—8

路径4: 1—2—3—4—8



(4) 设计测试用例

		输入数据	预期输出
测试用例1	路径1	1999	leap=0
测试用例2	路径2	1996	leap=1
测试用例3	路径3	1800	leap=0
测试用例4	路径4	1600	leap=1

黑盒测试的原理



- 把软件当成一个黑盒，看不到、也不关心其内部的实现细节。
- 根据软件**功能**设计**测试用例**。

如何设计测试用例

例如：课程注册系统中，要求“软件工程”课程最少10名最多不超过100名学生注册。

编号	测试数据	预期结果
1	12人	应正常处理
2	13人	应正常处理
3	14人	应正常处理
4	15人	应正常处理

■ 还要继续测试有9人、10、100、101人注册的情况吗？

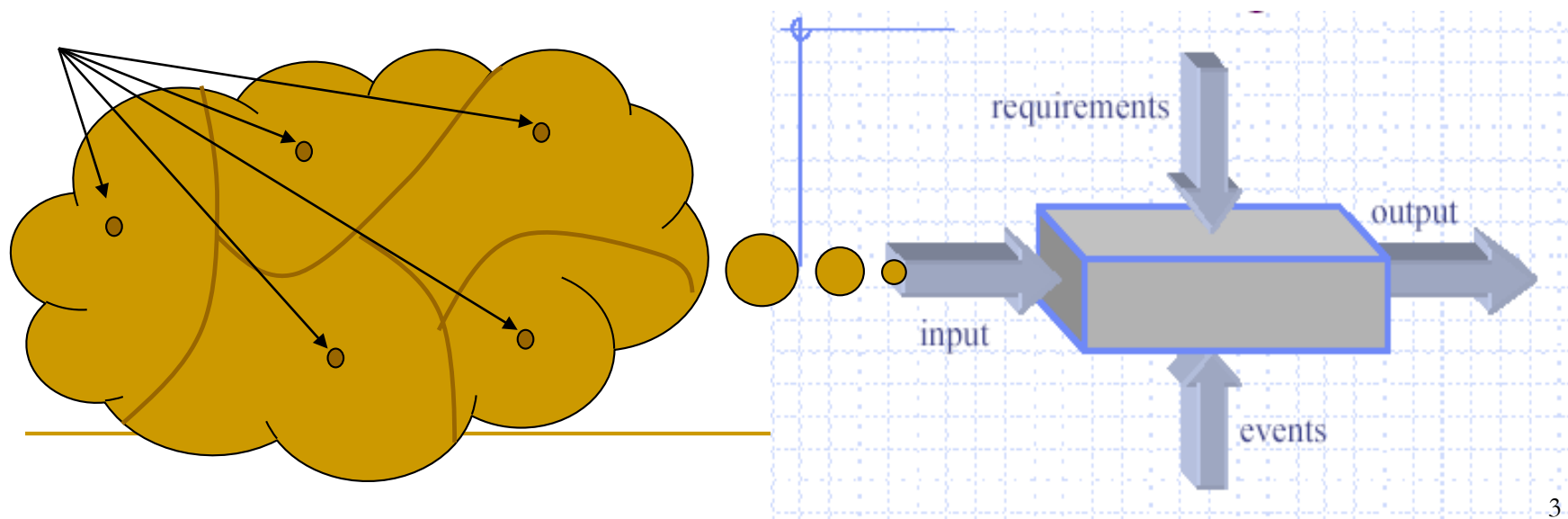
经验表明：输入域的边界值很容易出现错误。
为什么？

划分等价类

边界值分析

等价类划分(Equivalence Partitioning)

- 主要思想：把软件的输入数据集合按输入条件划分为若干个等价类，从中生成测试用例
 - 一个等价类中每个输入数据的作用是等效的
 - 为每个等价类设计一个（或几个）测试用例
- 减少测试用例的数量，并且不丧失发现错误的机会



如何划分等价类

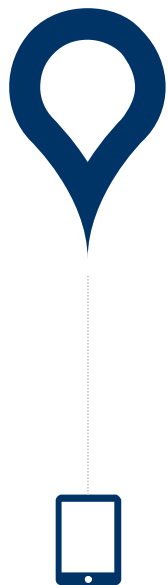
- 某个数据的输入条件为一个数值范围 $[a, b]$
 - 一个有效等价类 $[a, b]$
 - 两个无效等价类 $x < a$ 和 $x > b$
- 某个数据的输入条件为一个集合 $A = \{a_1, a_2, \dots\}$
 - 一个有效等价类 A
 - 一个无效等价类： A 的补集

如果输入条件是一个布尔表达式，如何划分等价类？

划分等价类实例

- 程序输入条件为小于100大于10的整数 x
 - 一个有效等价类： $10 < x < 100$
 - 两个无效等价类： $x \leq 10$ 和 $x \geq 100$
- 某个数据的输入条件为日期类型
 - 一个有效等价类：日期类型的数据
 - 一个无效等价类：非日期类型的数据
- 某个数据的输入条件要求密码非空
 - 一个有效等价类：非空密码
 - 一个无效等价类：密码为空

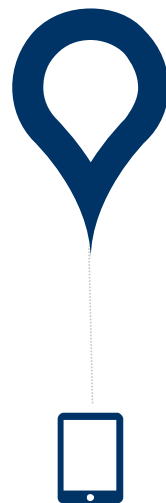
设计测试用例步骤



划分等价类，画出等价类表，并为每个等价类编号



设计一个测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到覆盖所有的有效等价类



为每一个无效等价类设计一个测试用例

案例1：NextDate函数问题

- nextDate函数包含三个变量：month、day和year，函数的输出为输入日期的后一天的日期。
- 函数要求输入变量均为整数，并且满足下列条件：
 - $1 \leq \text{month} \leq 12$
 - $1 \leq \text{day} \leq 31$
 - $1812 \leq \text{year} \leq 2050$
- 要求：给出等价类划分表并设计测试用例

等价类划分表

输入	有效等价类	等价类编号	无效等价类	等价类编号
日期的类型	数字字符	1	非数字字符	8
年	1812~2050	2	年小于1812	9
			年大于2050	10
月	1~12	3	月小于1	11
			月大于12	12
非闰年的2月	1~28	4	日小于1	13
			日大于28	14
闰年的2月	1~29	5	日小于1	15
			日大于29	16
月份为1月、3月、5月、7月、8月、10月、12月	1~31	6	日小于1	17
			日大于31	18
月份为4月、6月、9月、11月	1~30	7	日小于1	19
			日大于30	20

为有效等价类设计测试用例

序号	输入数据			预期输出			覆盖范围 (等价类编号)
	年	月	日	年	月	日	
1	2003	3	15	2003	3	16	1、2、3、6
2	2004	2	13	2004	2	14	1、2、3、5
3	1999	2	3	1999	2	4	1、2、3、4
4	1970	9	29	1970	9	30	1、2、3、7

为无效等价类设计测试用例

序号	输入数据			预期结果	覆盖范围 (等价类编号)
	年	月	日		
1	xy	5	9	输入无效	8
2	1700	4	8	输入无效	9
3	2300	11	1	输入无效	10
4	2005	0	11	输入无效	11
5	2009	14	25	输入无效	12
6	1989	2	-1	输入无效	13
7	1977	2	30	输入无效	14
8	2000	2	-2	输入无效	15
9	2008	2	34	输入无效	16
10	1956	10	0	输入无效	17
11	1974	8	78	输入无效	18
12	2007	9	-3	输入无效	19
13	1866	4	35	输入无效	20

案例2：课程注册功能的测试用例

- 课程注册系统中，要求“软件工程”课程最少10名最多不超过100名学生注册。

等价类：

- (1) $10 \leq \text{StudentCount} \leq 100$
- (2) $\text{StudentCount} < 10$
- (3) $\text{StudentCount} > 100$

三个测试用例，输入值为：

- (1) $\text{StudentCount} = 46$
- (2) $\text{StudentCount} = 8$
- (3) $\text{StudentCount} = 106$



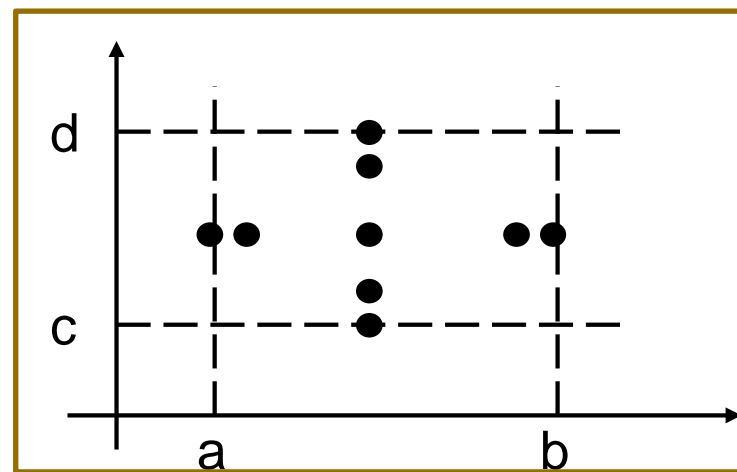
101

边界值分析(Boundary Value Analysis)

- 主要思想：边界条件处理容易出现错误，测试时需要特别考虑等价类的边界值。
 - 联合使用等价划分和边界值分析
- 如何设计测试用例
 - 某个数据的输入条件为一个数值范围[a, b]
 - 选择a、b和紧邻a+、b-的值
 - 选择a-、b+的值
 - 某个数据的输入条件为一个数值集合
 - 选择其中最大值、最小值以及紧邻的值

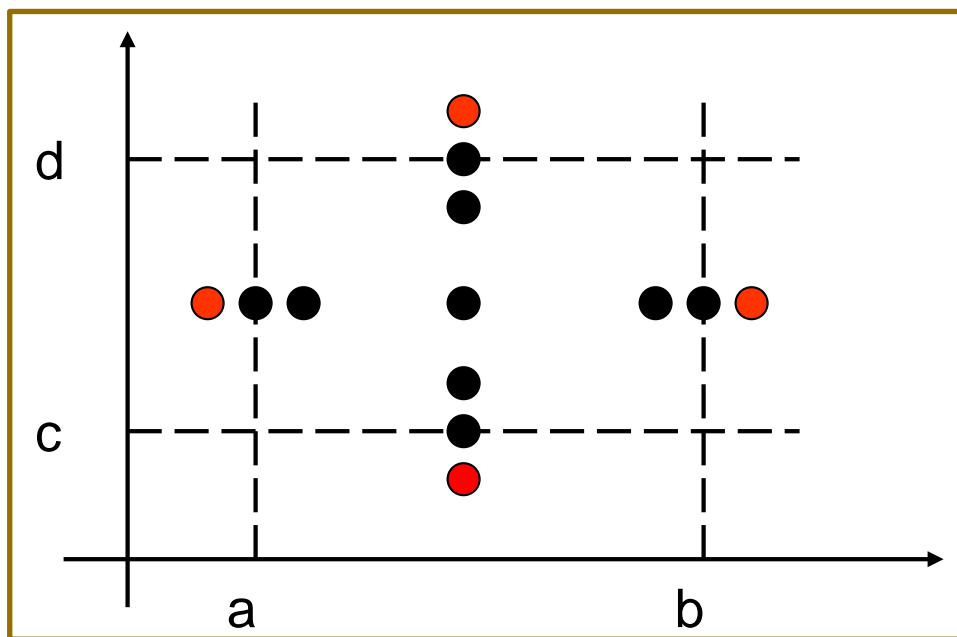
边界值分析

- 两个变量 X_1 ($a \leq X_1 \leq b$) and X_2 ($c \leq X_2 \leq d$)
 - X_1 的取值: $X_1\min$, $X_1\min+$, $X_1\text{nom}$, $X_1\text{max-}$, $X_1\text{max}$
 - X_2 的取值: $X_2\min$, $X_2\min+$, $X_2\text{nom}$, $X_2\text{max-}$, $X_2\text{max}$
 - 两个变量函数的边界值分析测试用例如下:
 $\{ \langle X_1\text{nom}, X_2\min \rangle, \langle X_1\text{nom}, X_2\min+ \rangle, \langle X_1\text{nom}, X_2\text{nom} \rangle, \langle X_1\text{nom}, X_2\text{max-} \rangle, \langle X_1\text{nom}, X_2\text{max} \rangle, \langle X_1\min, X_2\text{nom} \rangle, \langle X_1\min+, X_2\text{nom} \rangle, \langle X_1\text{max-}, X_2\text{nom} \rangle, \langle X_1\text{max}, X_2\text{nom} \rangle \}$
 - 保留其中一个变量, 让其余变量依次取min, min+, nom, max-和max
 - 一个含有 n 个变量的程序, 边界值分析会产生 $4n+1$ 个测试用例



健壮性边界值分析

- 两个变量 X_1 ($a \leq X_1 \leq b$) and X_2 ($c \leq X_2 \leq d$)
 - X_1 的取值: $X_1\text{min-}$, $X_1\text{min}$, $X_1\text{min+}$, $X_1\text{nom}$, $X_1\text{max-}$, $X_1\text{max}$, $X_1\text{max+}$
 - X_2 的取值: $X_2\text{min-}$, $X_2\text{min}$, $X_2\text{min+}$, $X_2\text{nom}$, $X_2\text{max-}$, $X_2\text{max}$, $X_2\text{max+}$
 - 一个含有 n 个变量的程序, 健壮性边界值分析会产生 $6n+1$ 个测试用例



边界值分析实例

TriTyp(a,b,c) :

$$1 \leq a \leq 100$$

$$1 \leq b \leq 100$$

$$1 \leq c \leq 100$$

a = {0, 1, 2, 50, 99, 100, 101}

b = {0, 1, 2, 50, 99, 100, 101}

c = {0, 1, 2, 50, 99, 100, 101}

案例1：课程注册功能的测试用例

- 课程注册系统中，要求“软件工程”课程最少10名最多不超过100名学生注册。

等价类：

- (1) $10 \leq \text{StudentCount} \leq 100$
- (2) $\text{StudentCount} < 10$
- (3) $\text{StudentCount} > 100$

已有测试输入：

$\text{StudentCount} = 46, 8, 101$

通过边界值分析新增加的测试输入：

$\text{StudentCount} = 10, 11, 99, 100$

案例2: NextDate函数问题

- $1 \leq \text{month} \leq 12$
- $1 \leq \text{day} \leq 31$
- $1812 \leq \text{year} \leq 2050$

Month = {0, 1, 2, 6, 11, 12, 13}

Day = {0, 1, 2, 15, 30, 31, 32}

Year = {1811, 1812, 1813, 2001, 2049, 2050, 2051}

案例2：NextDate函数问题

测试用例	day	month	year	预期输出
Test1	15	6	1811	变量year超出范围
Test2	15	6	1812	1812年6月16日
Test3	15	6	1813	1813年6月16日
Test4	15	6	2001	2001年6月16日
Test5	15	6	2049	2049年6月16日
Test6	15	6	2050	2050年6月16日
Test7	15	6	2051	变量year超出范围

案例2：NextDate函数问题

测试用例	day	month	year	预期输出
Test8	0	6	2001	变量day超出范围
Test9	1	6	2001	2001年6月2日
Test10	2	6	2001	2001年6月3日
Test11	30	6	2001	2001年7月1日
Test12	31	6	2001	输入日期不正确
Test13	32	6	2001	变量day超出范围

案例2：NextDate函数问题

测试用例	day	month	year	预期输出
Test14	15	0	2001	变量month超出范围
Test15	15	1	2001	2001年1月16日
Test16	15	2	2001	2001年2月16日
Test17	15	11	2001	2001年11月16日
Test18	15	12	2001	2001年12月16日
Test19	15	13	2001	变量month超出范围