

# 第六章

# 软件测试

## § 6.1 基本概念

软件开发过程必须伴有质量保证活动。

软件测试是软件质量保证的关键元素，代表了规约、设计和编码的最终检查。

# 有关测试的思考题

- 软件测试是一门非常重要的学科，主要研究内容是什么？
- 软件测试需要什么样的专业基础
- 软件质量到底是什么？
- 测试的目标是什么？
- 开发一个测试系统之前你是否明白：
  - 可以测试什么？
  - 应该测试什么？
  - 最终能够测试什么？

测试的目标是什么？

软件产品最大的成本是检测软件错误、修正软件错误的成本。

在整个软件开发中，测试工作量一般占30%~40%，甚至 $\geq 50\%$ 。

在人命关天的软件（如飞机控制、核反应堆等）测试所花费的时间往往是其它软件工程活动时间之和的三到五倍

# 软件测试背景

软件是人编的—所以不完美  
实例：

- **1999年12月3日, 美国航天局火星极地登陆飞船失踪**
- **1991年爱国者导弹防御系统系统时钟错误积累造成跟踪系统失去精确度**
- **千年虫, 世界各地解决2000年错误超过数亿美元**

# 软件测试的认识的发展

人们对软件测试认识的五个阶段：

阶段1 — 测试=调试

阶段2 — 测试是证明软件正确

阶段3 — 测试是发现软件中错误

阶段4 — 测试是减小软件不工作的风险

（是度量软件质量要素的过程）

阶段5 — 测试可产生低风险的软件的一种认识上的训练

# 软件测试的目标

(1) 预防错误：几乎不可实现

(2) 发现错误

## 6. 1. 1 测试的目的与地位

G. J. Myers在<软件测试技巧>中认为：

1. 测试是为了寻找错误而运行程序的过程。
2. 一个好的测试用例是指很可能找到迄今为止尚未发现的错误的测试。
3. 一个成功的测试是揭示了迄今为止尚未发现的错误的测试。

E. W. Dijkstra 指出：

“程序测试能证明错误的存在，但不能证明错误不存在。”

测试的目的是发现程序中的错误，是为了证明程序有错，而不是证明程序无错。



**“测试的目的是说明程序正确地执行它应有的功能” 这种说法正确吗？**

**例：程序Triangle，输入三个整数，表示一个三角形的三个边长，该程序产生一个结果，指出该三角形是等边三角形、等腰三角形还是不等边三角形。**

**为说明其能正确执行它的功能，可使用“测试用例” (3, 4, 5), (5, 5, 6), (6, 6, 6), 程序都能给出正确结果，是否就可认为程序是正确的？**

# 难以说清的软件缺陷

如果软件中的问题没有人发现，那么它算不算软件缺陷？”

古谚：“一片树叶飘落在森林中没有人听见，  
谁能说它发出了声音？”

由于不能报告没有看见的问题，因此，  
没有看见就不能说存在软件缺陷

只有看到了，才能断言软件缺陷，  
尚未发现的软件缺陷只能说是未知软件缺陷。

眼  
见  
为  
实

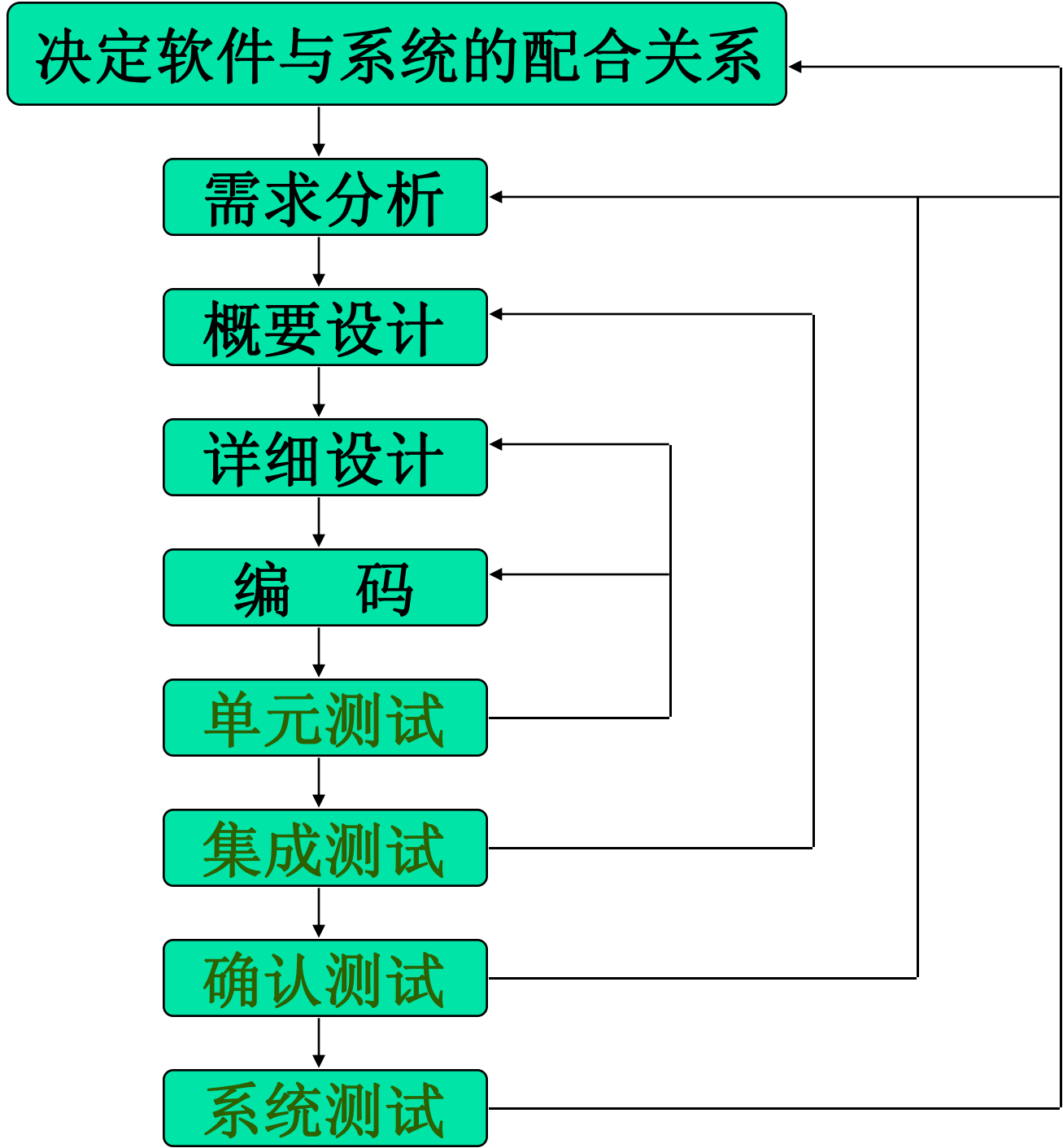
## 6. 1. 2 测试原则

(1) 所有的测试都应追溯到用户需求

最严重的错误(从用户角度)是那些导致软件无法满足需求的错误。

程序中的问题根源可能在开发前期的各阶段解决、纠正错误也必须追溯到前期工作。

测试与开发前期工作的关系



(2) 概要设计时应完成测试计划，详细的测试用例定义可在设计模型确定后开始，所有测试可在任何代码被产生之前进行计划和设计。

# 软件测试不等于程序测试

软件测试应贯穿于软件定义与开发的整个期间；

据美国一家公司统计，查出的软件错误中，属于需求分析和软件设计的错误约占 64%，属于程序编写的错误仅占 36%。程序编写的许多错误是“先天的”。

# 测试阶段工作步骤

- **单元测试：** 检验每个模块能否单独工作。
- **集成测试：** 检验概要设计中模块接口设计问题
- **确认测试：** 以需求规格说明书为检验尺度
- **系统测试：** 综合检验

(3) pareto原则：测试发现的错误中80%很可能起源于20%的模块中。应孤立这些疑点模块重点测试。

(4) 穷举测试是不可能的。

(5) 应由独立的底三方来构造测试。

（开发和测试队伍分别建立）



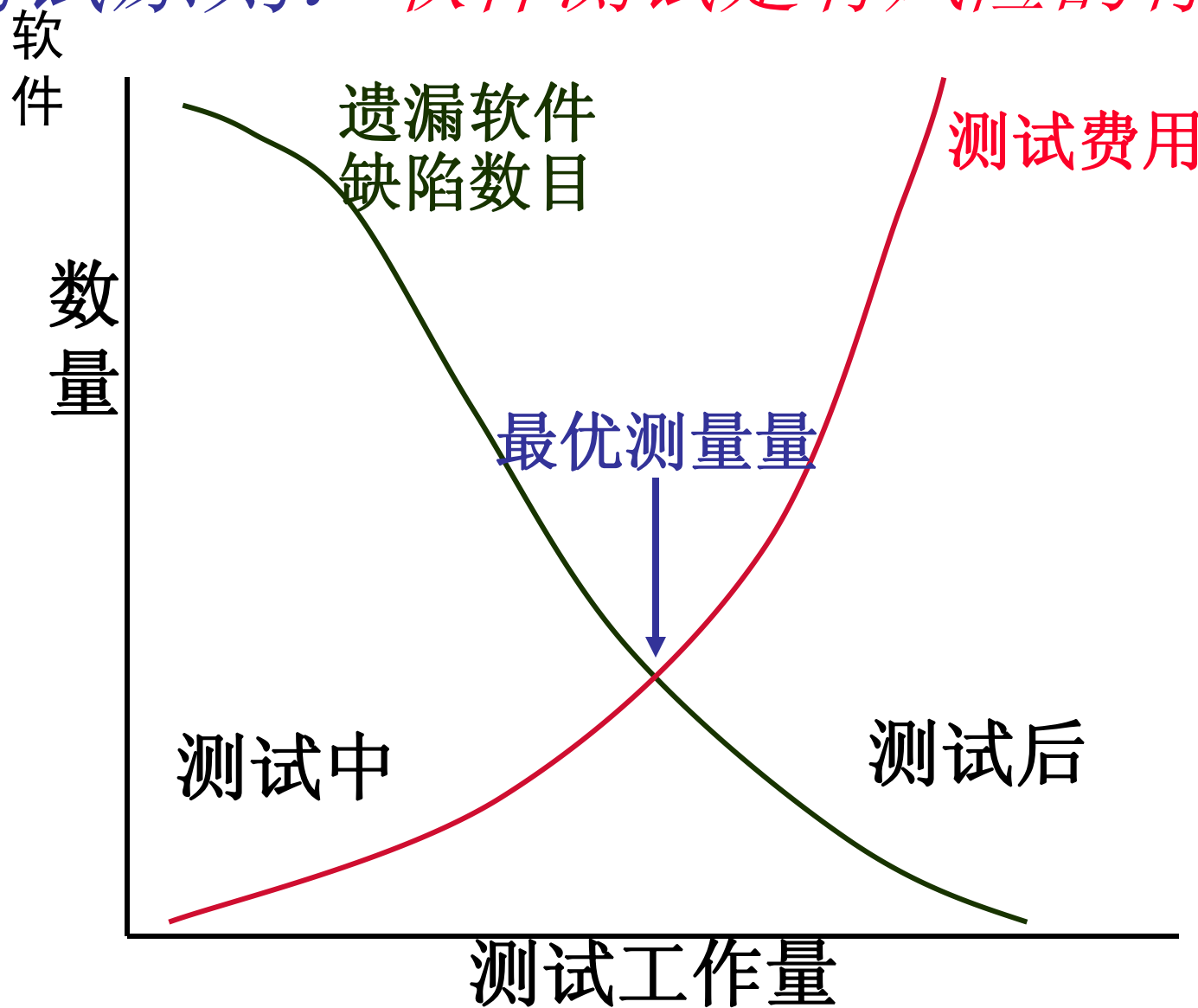
(6) 测试用例应由输入数据和预期的输出结果两部分组成.

(7) 兼顾合理的输入和不合理的输入数据

(8) 程序修改后要回归测试

(9) 应长期保留测试用例，直至系统废弃。

# 测试原则：软件测试是有风险的行为



每一个软件项目都有一个最优的测量量

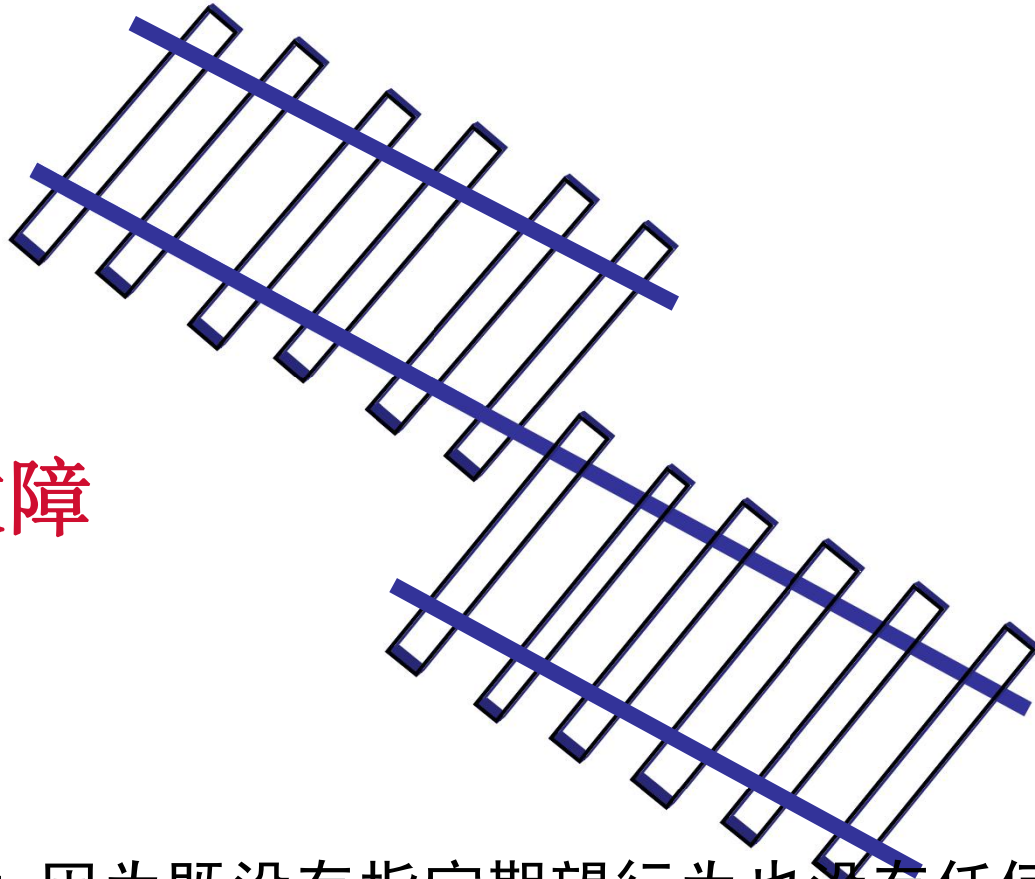
## 6. 1. 3 可测试性

# “好”测试的属性：

- 发现错误的可能性高
- 不冗余
- 在目的相似的测试中，应使用最可能找到错误的测试
- 每一个测试应独立执行

- **错误**，也称缺陷或不足，是可能引起组件不正常行为的设计或编码错误。
- **误差**是系统执行过程中错误的表现。
- **故障**是组件的规格说明与其行为之间的偏差，故障是由一个或多个误差引起的。
- **测试用例**是一组输入和期待的结果，它根据引起故障和检查的目的来使用组件。
- **改正**是对组件的变化。改正的目的在于修正错误。改正可能会产生新的错误。

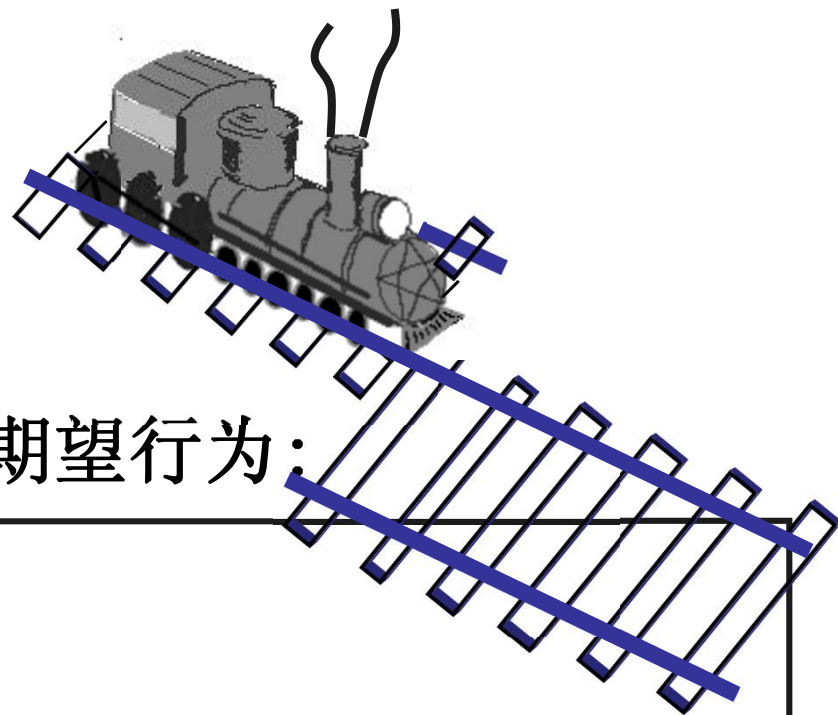
# 错误 (***fault***)、误差 (***error***) 和故障 (***failure***)



出轨(出故障)  
)?

图中不表示故障, 因为既没有指定期望行为也没有任何观察的行为, 它也不表示误差, 因为这不意味着系统正处在进一步处理将导致故障的状态.







# 错误 (*fault*)、误差 (*error*) 和故障 (*failure*)



用例DriveTrain指定了列车的期望行为:

用例名称	DriveTrain
参与执行者	火车司机
入口条件	司机按下控制面板上的StrartTrain
事件流	1. 列车开始在轨道1上运行 2. 列车平移到轨道2
退出条件	列车运行在轨道2上
特殊条件	没有

# 软件错误分类

-  功能错 (需求分析错误)
-  软件结构错
-  数据错
-  编码错
-  软件集成错
-  测试定义与测试执行错误



## 6. 1. 4 测试用例设计

选择测试用例是软件测试员最重要的一项工作。

测试用例的属性：

属性	描述
name	测试用例的名称
location	可执行的完全路径名
input	输入数据或命令
oracle	与测试输入相比较的期待测试结果
log	测试生产的输出

# 程序测试举例

例：程序 Triangle， 输入三个整数， 表示一个三角形的三个边长， 该程序产生一个结果， 指出该三角形是等边三角形、等腰三角形还是不等边三角形。

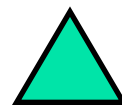
# 判断三角型的测试用例设计:

输入数据

预期结果

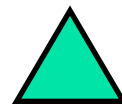
(1) 6;6;6

等边



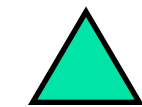
(2) 8;8;4

等腰



(3) 4;5;6

一般



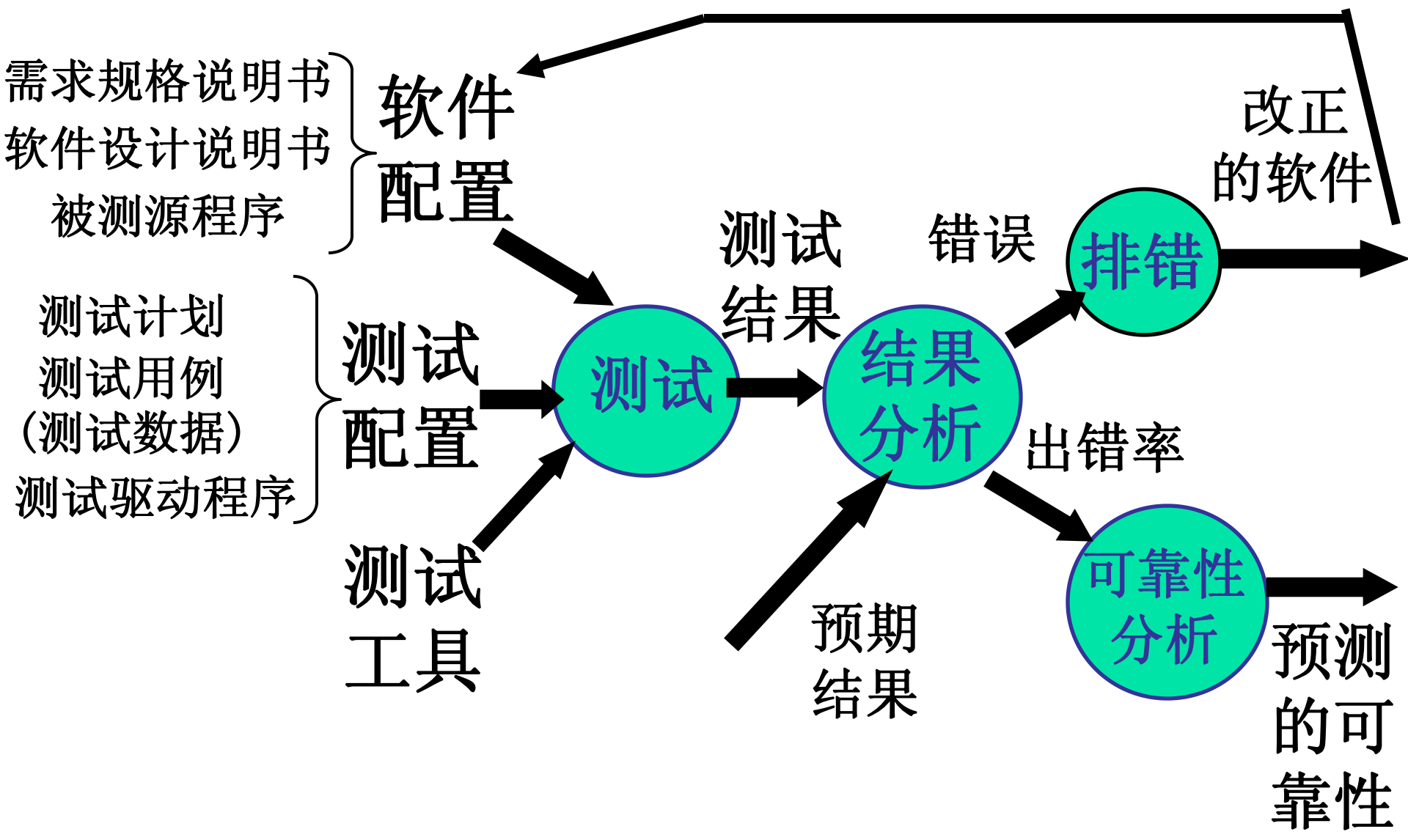
还应输入非法数据:

0; 7; 9

-7;3; 5

a; 2; 7 等

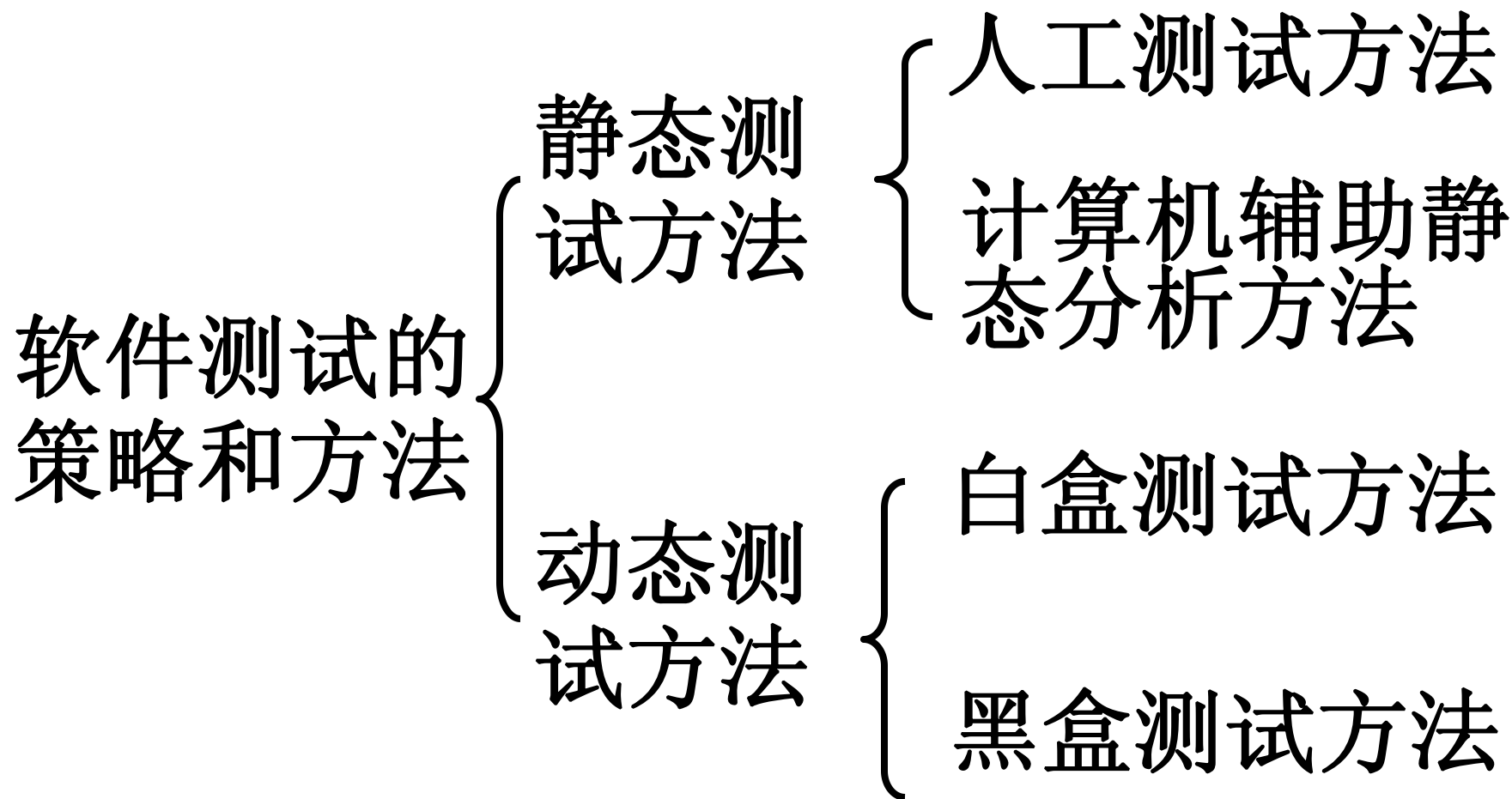
# 6.1.5 软件测试信息流



# 测试设计中需要考虑的**22**种测试类型


- 黑盒测试
- 白盒测试
- 单元测试
- 累计综合测试
- 集成测试
- 功能测试
- 系统测试
- 端到端测试
- 健全测试
- 衰竭测试
- 接受测试
- 负载测试
- 强迫测试
- 性能测试
- 可用性测试
- 安装/卸载测试
- 恢复测试
- 兼容测试
- 安全测试
- 比较测试
- Alpha测试
- Beta测试

## 6.1.6 测试的方法与技术



# 静态和动态测试

## 汽车的检查过程：

- 踩油门
  - 看车漆
  - 打开前盖检查
- 
- 静态测试

- 发动汽车
  - 听听发动机声音
  - 上路行使
- 
- 动态测试

**静态测试:** 基本特征是在对软件进行分析、检查和审阅，不实际运行被测试的软件。

静态测试约可找出30~70%的逻辑设计错误。

对需求规格说明书、软件设计说明书、源程序做检查和审阅，包括：

- 是否符合标准和规范；
- 通过结构分析、流图分析、符号执行指出软件缺陷；



**动态测试：** 通过运行软件来检验软件的动态行为和运行结果的正确性

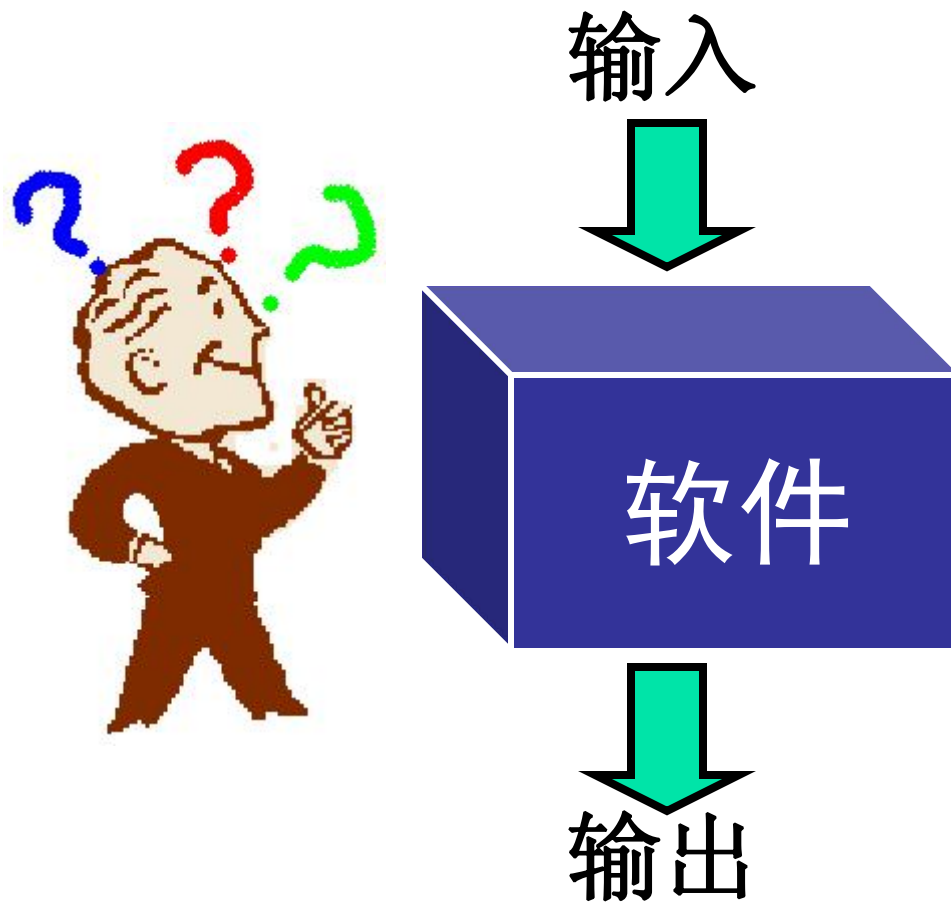
**动态测试的两个基本要素：**

- ◆ 被测试程序
- ◆ 测试数据（测试用例）

# 动态测试方法

- (1) 选取定义域有效值, 或定义域外无效值.
- (2) 对已选取值决定 **预期的结果**
- (3) 用选取值执行程序
- (4) **执行结果** 与 (2) 结果相比,  
**不吻合和程序有错.**

# 动态黑盒测试 — 闭着眼睛测试软件

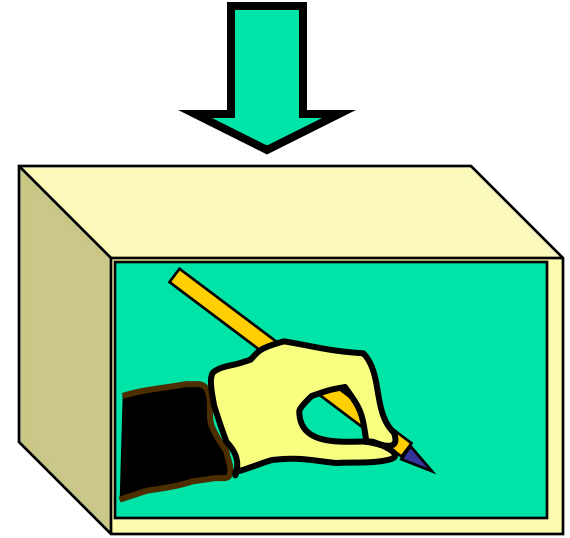
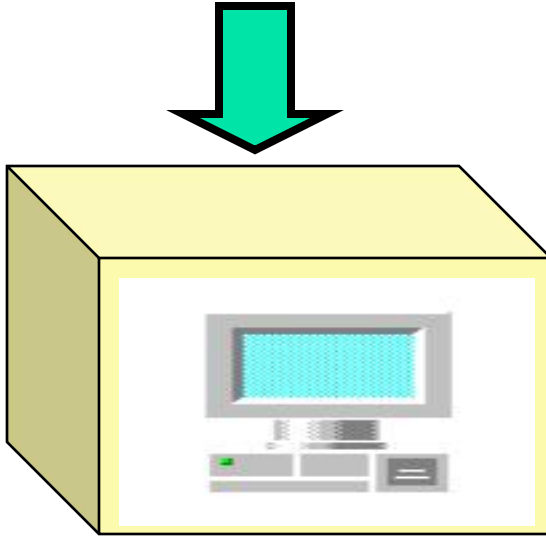


不深入代码细节的测试方法称为动态黑盒测试。  
软件测试员充当客户来使用它。

# 动态白盒测试 — 带上X光眼镜测试软件

$250 * (1 + 0.015) * ((1 + 0.015)^{360} - 1) / 0.015$

$250 * (1 + 0.015) * ((1 + 0.015)^{360} - 1) / 0.015$



3581322.293419985680302829734315

????????????????

假如知道一个盒子包含一台计算机, 而另一个盒子是人用纸笔计算, 就会选择不同的测试用例

了解软件的运作方式会影响测试手段

# § 6.2 两种类型的测试

## 6.2.1 黑盒测试

又称：功能测试

数据驱动测试

基于规格说明书的测试

## 6.2.2 白盒测试

又称：开盒测试

结构测试

玻璃盒测试

基于覆盖的测试.

根据被测程序的逻辑结构设计  
测试用例；

力求提高测试覆盖率；

# 黑盒测试与白盒测试比较

黑盒测试是从用户观点，按规格说明书要求的输入数据与输出数据的对应关系设计测试用例，是根据程序外部特征进行测试。

白盒测试是根据程序内部逻辑结构进行测试。

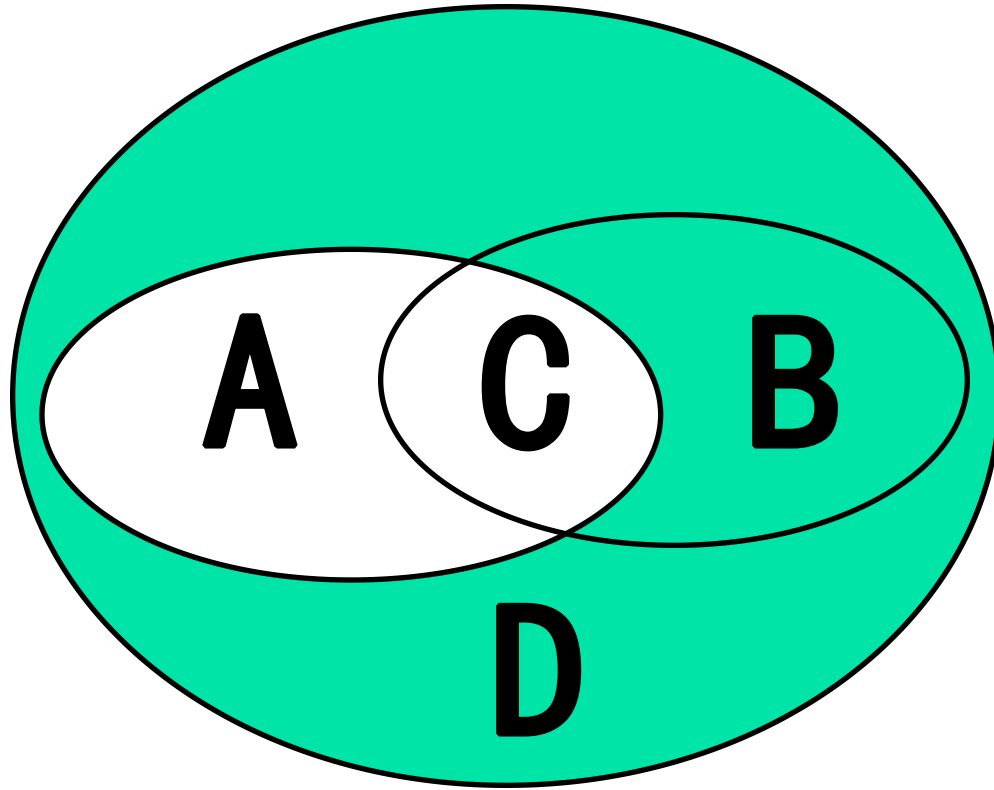
# 黑盒测试与白盒测试优缺点比较

	黑盒测试	白盒测试
优点	<ul style="list-style-type: none"><li>①适用于各阶段测试</li><li>②从产品功能角度测试</li><li>③容易入手生成测试数据</li></ul>	<ul style="list-style-type: none"><li>①可构成测试数据使特定程序部分得到测试</li><li>②有一定的充分性度量手段</li><li>③可或较多工具支持</li></ul>
缺点	<ul style="list-style-type: none"><li>①某些代码得不到测试</li><li>②如果规格说明有误，则无法发现</li><li>③不易进行充分性测试</li></ul>	<ul style="list-style-type: none"><li>①不易生成测试数据(通常)</li><li>②无法对未实现规格说明的部分进行测试</li><li>③工作量大，通常只用于单元测试，有应用局限</li></ul>
性质	是一种 <b>确认</b> 技术，回答“我们在构造一个正确的系统吗？”	是一种 <b>验证</b> 技术，回答“我们在正确地构造一个系统吗？”



不论黑盒还是白盒测试都**不能**  
**进行穷尽测试**，所以软件测试不可  
能发现程序中存在的**所有错误**，因  
此需精心设计测试方案，**力争尽可**  
**能少的次数**，测出**尽可能多的错误**。

# 黑盒测试与白盒测试能发现的错误



**A** - 只能用黑盒测试发现的错误

**B** - 只能用白盒测试发现的错误

**C** - 两种方法都能发现的错误

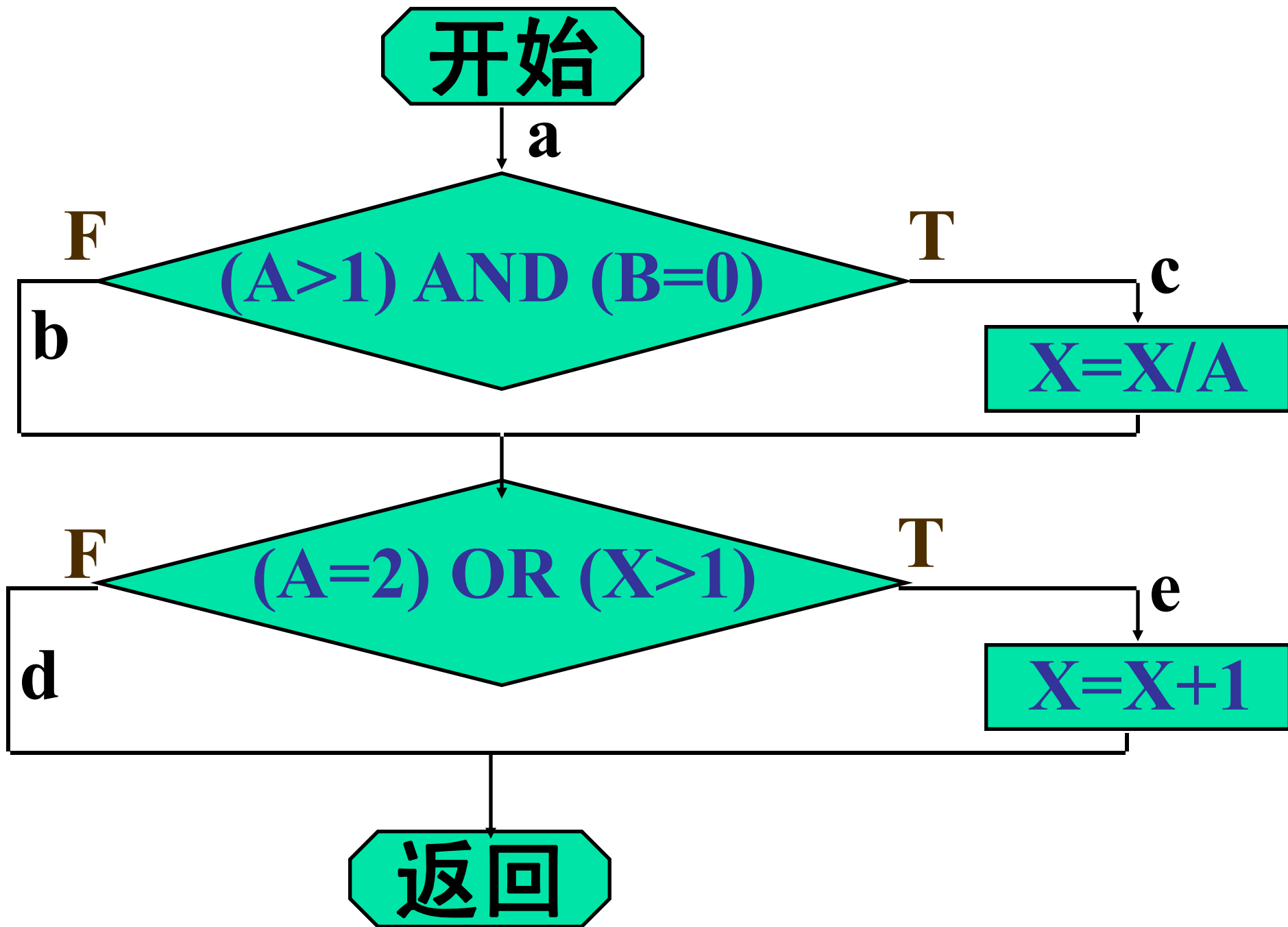
**D** - 两种方法都不能发现的错误

# § 6.3 白盒测试的测试用例设计

## 6.3.1 逻辑覆盖法

- (1) 语句覆盖
- (2) 判定覆盖
- (3) 条件覆盖
- (4) 判定/条件覆盖
- (5) 条件组合覆盖
- (6) 路径覆盖
- (7) 点覆盖
- (8) 边覆盖

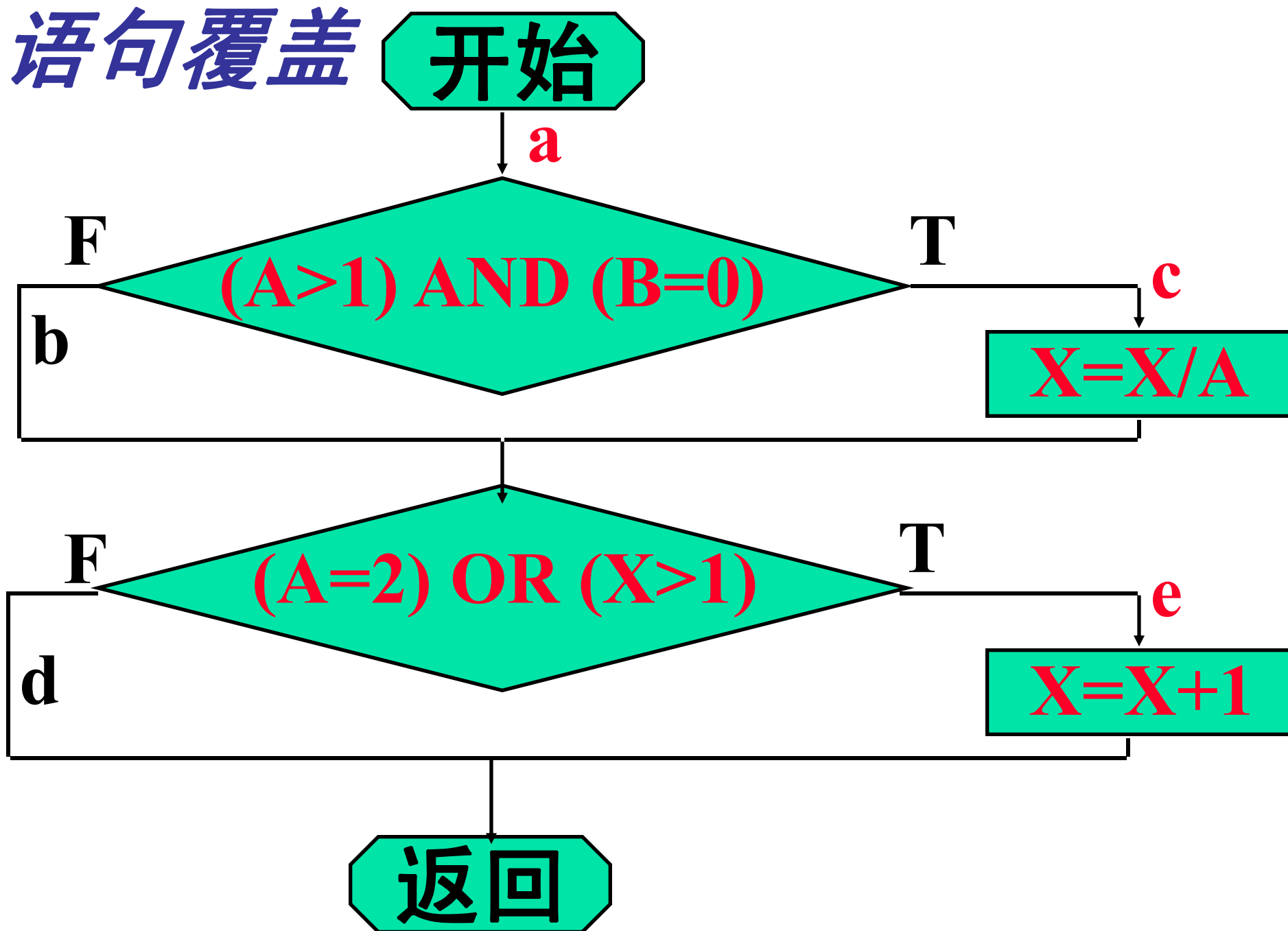
```
例: PROCEDURE SAMPAL  
  (A, B: REAL;  VAR X: REAL);  
  BEGIN  
    IF (A>1) AND (B=0)  
      THEN X:=X/A  
    IF (A=2) OR (X>1)  
      THEN X:=X+1  
  END;
```



## *(1) 语句覆盖*

使程序中每个语句至少执行一次

# 语句覆盖



只需设计一个测试用例：

输入数据：**A=2, B=0, X=4**

即达到了语句覆盖；

语句覆盖是**最弱**的逻辑覆盖



## (2) 判定覆盖 (分支覆盖)

使每个判定的真假分支都至少执行一次

判定覆盖

开始

a

F

$(A > 1) \text{ AND } (B = 0)$

T

c

$X = X / A$

b

F

$(A = 2) \text{ OR } (X > 1)$

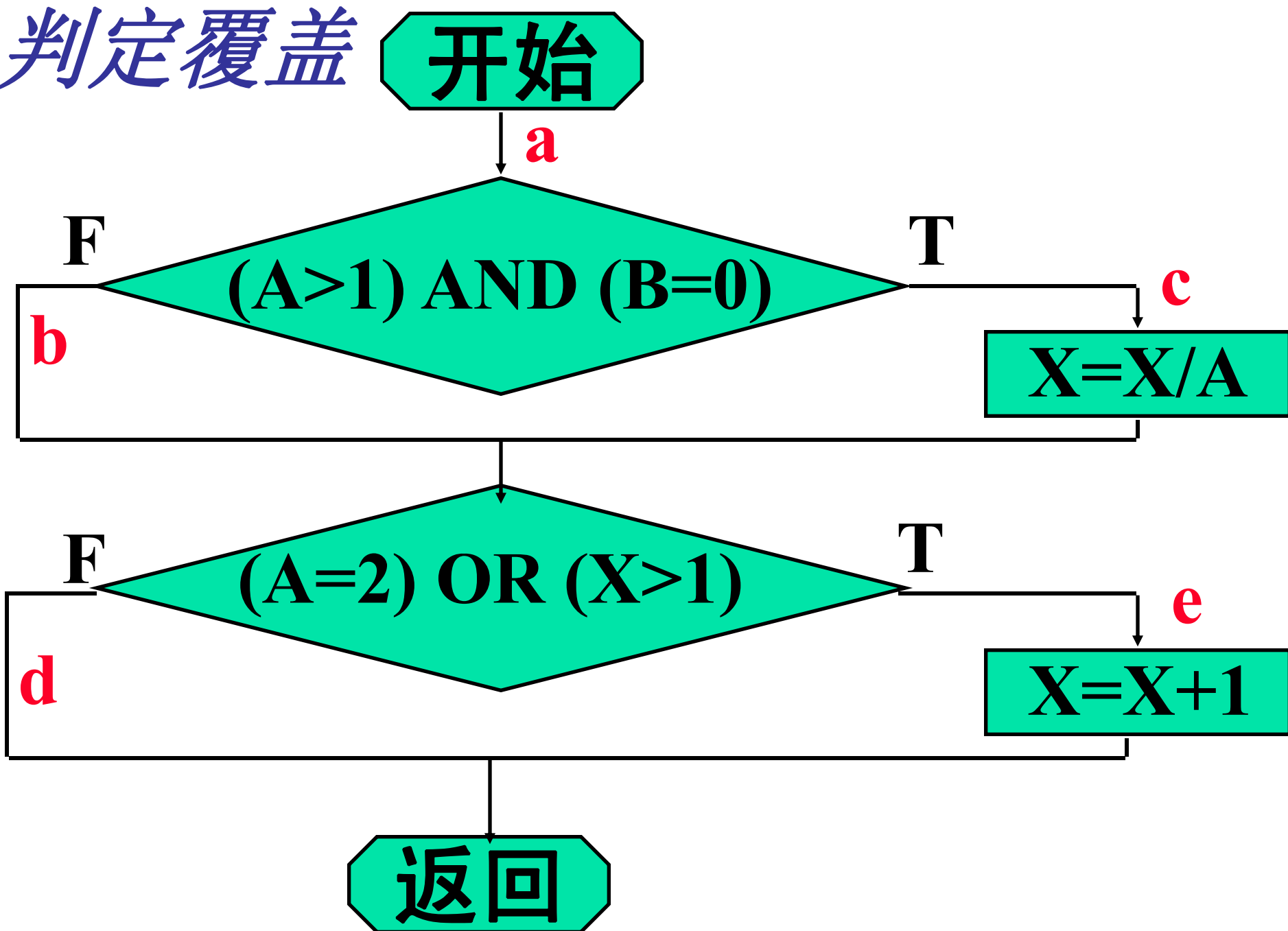
T

e

$X = X + 1$

d

返回



例：可设计两组测试用例：

★  $A=3$ ，  $B=0$  ，  $X=3$  可覆盖 **c**、 **d** 分支

★  $A=2$ ，  $B=1$  ，  $X=1$  可覆盖 **b**、 **e** 分支

两组测试用例可覆盖所有判定的真假分支

语句覆盖仍是**弱**的逻辑覆盖

### (3) 条件覆盖

使每个判定的每个条件的可能取值至少执行一次

## 第一判定表达式:

设条件	$A > 1$	取真	记为	$T1$
		假		$\overline{T1}$
条件	$B = 1$	取真	记为	$T2$
		假		$\overline{T2}$

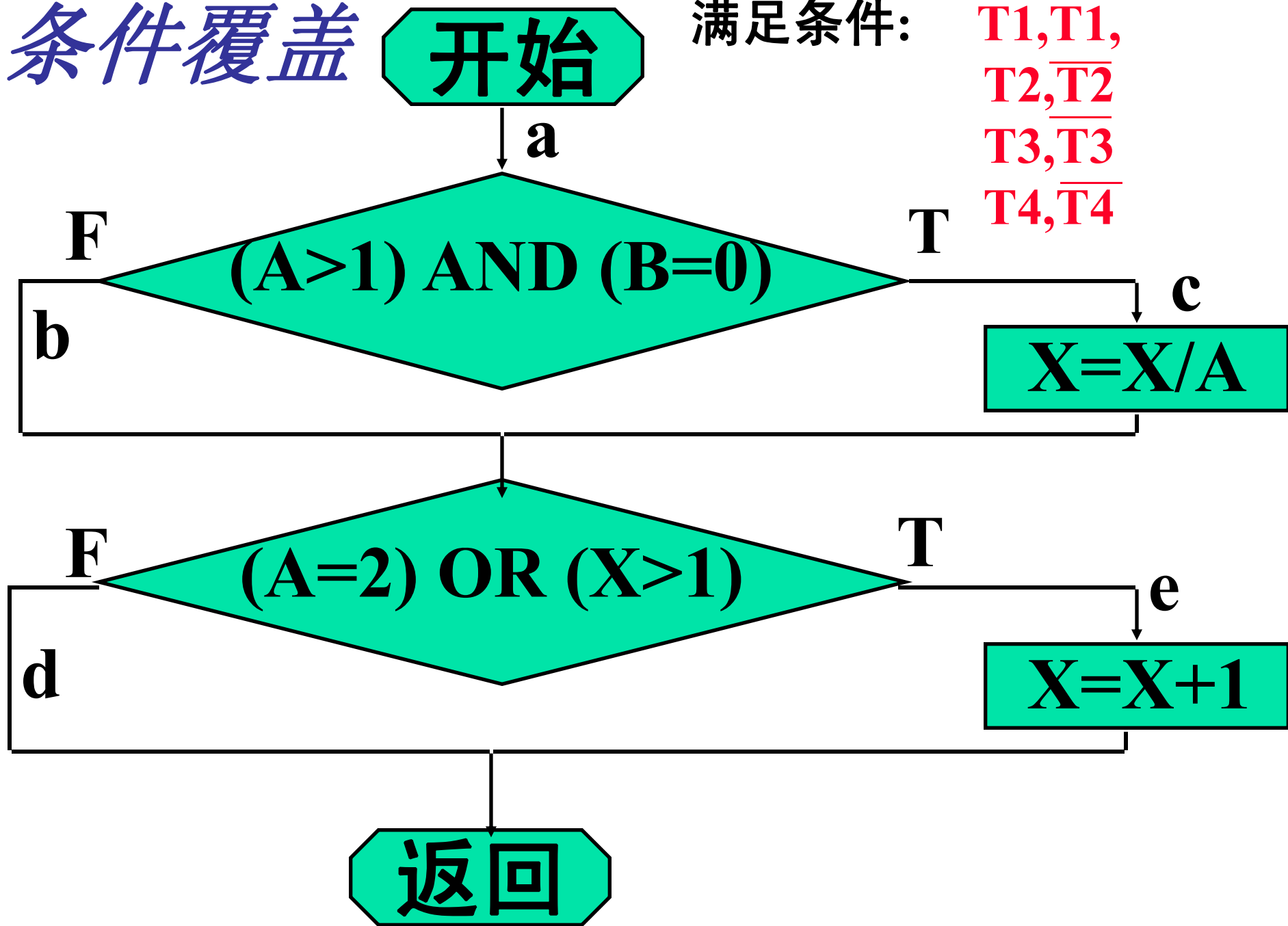
## 第二判定表达式:

设条件	$A = 2$	取真	记为	$T3$
		假		$\overline{T3}$
条件	$X > 1$	取真	记为	$T4$
		假		$\overline{T4}$

# 条件覆盖

满足条件:

$T1, \overline{T1},$   
 $T2, \overline{T2}$   
 $T3, \overline{T3}$   
 $T4, \overline{T4}$



测试用例 A B X	通过 路径	满足的 条件	覆盖 分支
1 0 3	abe	$\bar{T}1, T2, \bar{T}3, T4$	<b>b, e</b>
2 1 1	abe	$T1, \bar{T}2, T3, \bar{T}4$	<b>b, e</b>

两个测试用例覆盖了四个条件八种可能取值。

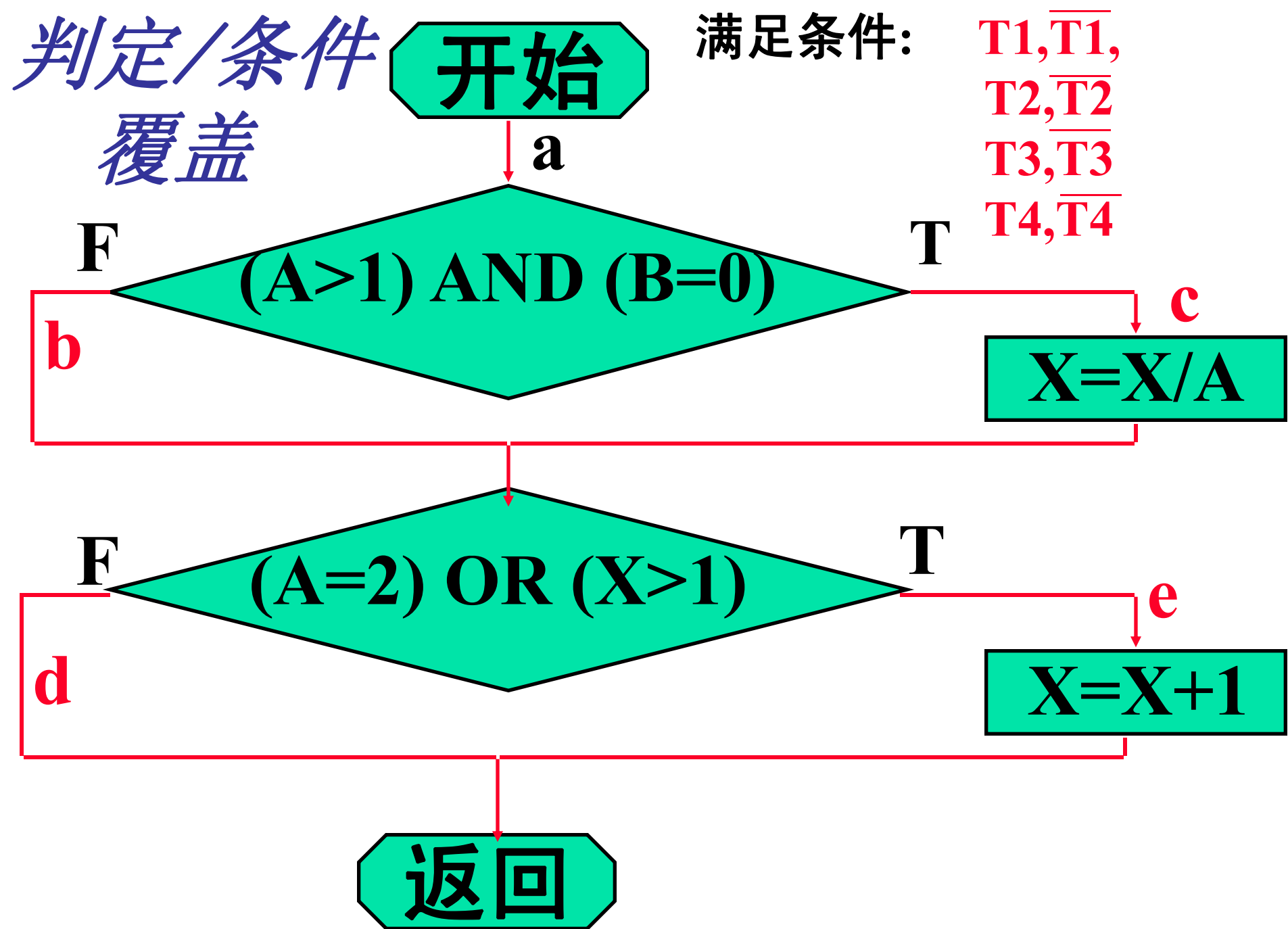
未覆盖c、d分支，不满足判定覆盖的要求。

条件覆盖不一定包含判定覆盖  
判定覆盖也不一定包含条件覆盖

## (4) 判定/条件覆盖

选取足够多的测试用例，使判断中的每个条件的所有可能取值至少执行一次，同时每个判断本身的所有可能判断结果至少执行一次。





测试用例 A B X	通过 路径	满足的 条件	覆盖 分支
2 0 4	ace	T1, T2, T3, T4	c, e
2 1 1	abd	$\overline{T1}$ , $\overline{T2}$ , $\overline{T3}$ , $\overline{T4}$	b, d
能同时满足判定、条件两种覆盖标准。 取值。			

测试用例 A B X	通过 路径	满足的 条件	覆盖 分支
2 0 3	ace	T1, T2, T3, $\bar{T}4$	c, e
2 1 1	abe	T1, $\bar{T}2$ , T3, $\bar{T}4$	b, e
1 0 3	abe	$\bar{T}1$ , T2, $\bar{T}3$ , T4	b, e
1 1 1	abd	$\bar{T}1$ , $\bar{T}2$ , $\bar{T}3$ , $\bar{T}4$	b, d

## (5) 条件组合覆盖

所有可能的条件取值组合至少执行一次

★  $A > 1, B = 0$

★  $A > 1, B \neq 0$

✳  $A \nlessgtr 1, B = 0$

✳  $A \nlessgtr 1, B \neq 0$

⊞  $A = 2, X > 1$

⊞  $A = 2, X \nlessgtr 1$

✧  $A \neq 2, X > 1$

✧  $A \neq 2, X \nlessgtr 1$

测试用例			通过	满足的	覆盖
A	B	X	路径	条件	分支
2	0	4	ace	T1, T2, T3, T4	c, e
2	1	1	abe	T1, $\bar{T}2$ , T3, $\bar{T}4$	b, e
1	0	2	abd	$\bar{T}1$ , T2, $\bar{T}3$ , T4	b, d
1	1	1	abd	$\bar{T}1$ , $\bar{T}2$ , $\bar{T}3$ , $\bar{T}4$	b, d

## **(6) 路径覆盖**

**覆盖每一个可能的路径**

测试用例 <b>A B X</b>	通过 路径	满足的 条件	覆盖 分支
<b>1 1 1</b>	<b>abd</b>	<b>T1, T2, T3, T4</b>	<b>b, d</b>
<b>1 1 2</b>	<b>abe</b>	<b><math>\bar{T}1, \bar{T}2, \bar{T}3, T4</math></b>	<b>b, e</b>
<b>3 0 1</b>	<b>acd</b>	<b>T1, T2, <math>\bar{T}3, \bar{T}4</math></b>	<b>c, d</b>
<b>2 0 4</b>	<b>ace</b>	<b>T1, T2, T3, T4</b>	<b>c, e</b>

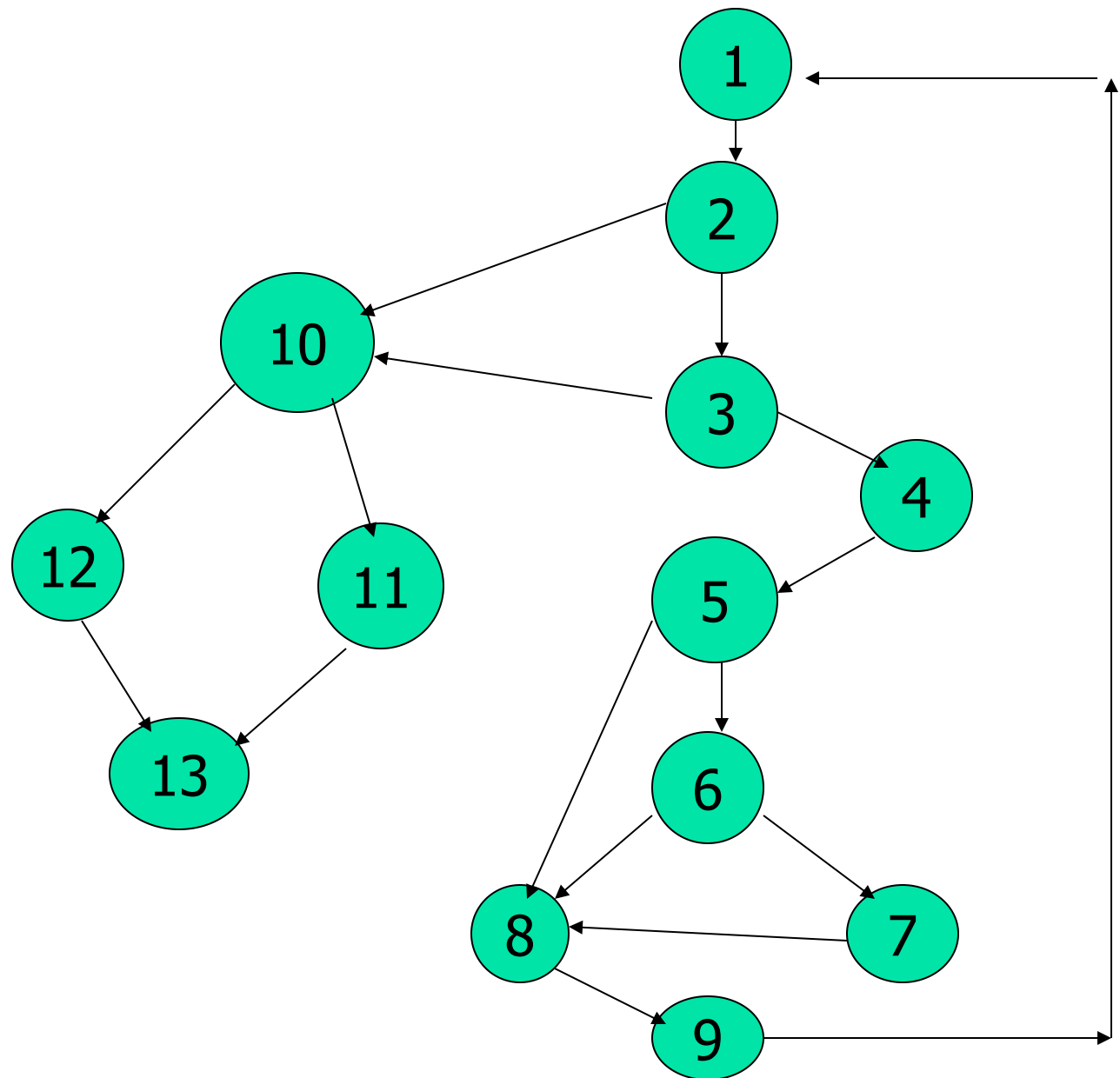
# 基本路径测试法

通过分析由控制构造的环路的复杂性，导出基本路径集合，从而设计测试用例，保证这些路径至少通过一次。

## 基本路径测试步骤：

- 导出程序流程图的拓扑结构-流图(程序图)
- 计算流图G的环路复杂度 $V(G)$
- 确定只包含独立路径的基本路径集
- 设计测试用例

# ■ 导出程序流程图的拓扑结构-流图





计算流图G的环路复杂度 $V(G)$

$$V(G) = \text{区域个数} = 6$$

$$V(G) = \text{边的条数} - \text{节点个数} + 2 = 17 - 13 + 2 = 6$$

$$V(G) = \text{判定节点个数} + 1 = 5 + 1 = 6$$

# 确定只包含独立路径的基本路径集

path1:1-2-10-11-13

path2:1-2-10-12-13

path3:1-2-3-10-11-13

path4:1-2-3-4-5-8-9-2-....

Path5:1-2-3-4-5-6-8-9-2-...

Path6:1-2-3-4-5-6-7-8-9-2-...

一条新路径必须包含一条新边。

这6条路径组成了一个基本路径集。6 (环路复杂度 $V(G)$ ) 是构成这个基本路径集的独立路径数的上界，也是设计**测试用例的数目**。

设计测试用例，保证基本路径集中每条路径的执行。

# § 6. 4黑盒测试的测试用例设计

## 6.4.1 等价类划分法

把所有可能的输入数据(有效的和无效的)划分成若干个等价的子集(称为等价类), 使得每个子集中的一个典型值在测试中的作用与这一子集中所有其它值的作用相同.

可从每个子集中选取一组数据来测试程序

# 如何划分等价类？

- 有效等价类(合理等价类)
- 无效等价类(不合理等价类)

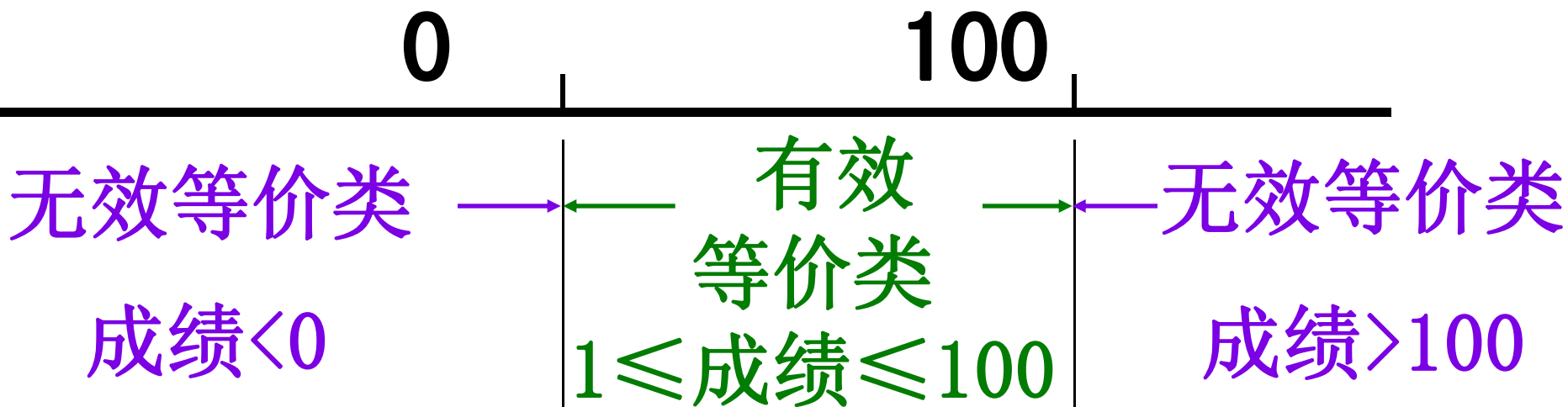
## 划分等价类的标准：

- 覆盖
- 不相交
- 代表性

# 划分等价类的规则

(1) 如果输入条件规定了取值范围，  
可定义一个有效等价类和两个无效等价类。

**例** 输入值是学生成绩，范围是0~100



# 划分等价类的规则:

(2) 如果输入条件代表集合的某个元素，则可定义一个有效等价类和一个无效等价类。

# 划分等价类的规则：

(3) 如规定了输入数据的一组值，且程序对不同输入值做不同处理，则每个允许的输入值是一个有效等价类，并有一个无效等价类（所有不允许的输入值的集合）。

例：输入条件说明学历可为：专科、本科、硕士、博士四种之一，则分别取这四种这四个值作为四个有效等价类，另外把四种学历之外的任何学历作为无效等价类

# 划分等价类的规则：

- (4) 如果规定了输入数据必须遵循的规则，可确定一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。
- (5) 如已划分的等价类各元素在程序中的处理方式不同，则应将此等价类进一步划分成更小的等价类。



# 用等价类划分法设计测试用例步骤：

- (1) 形成等价类表，每一等价类规定一个唯一的编号；
- (2) 设计一测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类，重复这一步骤，直到所有有效等价类均被测试用例所覆盖；
- (3) 设计一新测试用例，使其只覆盖一个无效等价类，重复这一步骤直到所有无效等价类均被覆盖；

# 第一步：等价类划分

“报表日期”输入条件的等价类表

输入等价类	有效等价类	无效等价类
报表日期的类型及长度	3位数字字符(1)	有非数字字符 (4) 少于6个数字字符 (5) 多于6个数字字符 (6)
年份范围	在2001~2005之间 (2)	小于2001 (7) 大于2005 (8)
月份范围	在1~12之间(3)	小于1 (9) 大于12 (10)

**第二步：为有效等价类设计测试用例**  
对表中编号为1, 2, 3的3个有效等价类  
用一个测试用例覆盖：

测试数据	期望结果	覆盖范围
200105	输入有效	等价类(1) (2) (3)

### 第三步：为每一个无效等价类设至少一个测试用例

测试数据	期望结果	覆盖范围
001MAY	输入无效	等价类(4)
20015	输入无效	等价类(5)
2001005	输入无效	等价类(6)
200005	输入无效	等价类(7)
200805	输入无效	等价类(8)
200100	输入无效	等价类(9)
200113	输入无效	等价类(10)



不能出现相同的  
测试用例

本例的10个等价类至少需要8个测试用例

# 例：对招干考试系统“输入学生成绩” 子模块设计测试用例

招干考试分三个专业, 准考证号第一位  
为专业代号, 如： 1-行政专业,  
2-法律专业,  
3-财经专业.

行政专业准考证号码为：110001~111215

法律专业准考证号码为：210001~212006

财经专业准考证号码为：310001~314015

# 例：准考证号码的等价类划分

有效等价类：

$$(1) \quad 110001 \sim 111215$$

$$(2) \quad 210001 \sim 212006$$

$$(3) \quad 310001 \sim 314015$$

无效等价类：

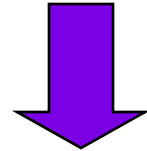
$$(4) \quad -\infty \sim 110000$$

$$(5) \quad 111216 \sim 210000$$

$$(6) \quad 212007 \sim 31000$$

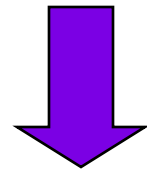
$$(7) \quad 314016 \sim +\infty$$

等价类划分即把输入空间分解成一系列子域，软件在一个子域内的行为应是等价的。



软件错误分为两类：

{	计算错误
	域错误

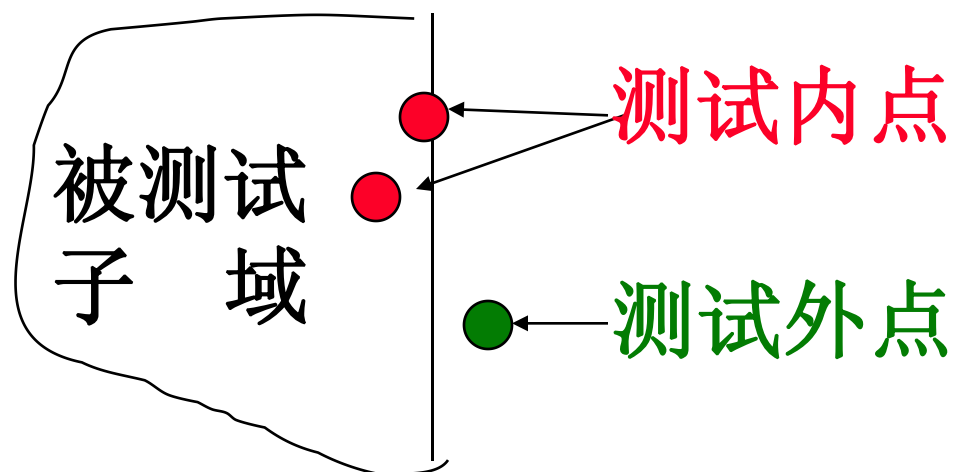


- ◆ 针对计算错误的测试方法
- ◆ 针对域错误的测试方法：测试域边界划定的正确性

## 6.4.2 边界值分析法

### 边界值分析法与等价类划分法区别

- (1) 边界值分析不是从某等价类中随便挑一个作为代表，而是使这个等价类的每个边界都要作为测试条件。
- (2) 边界值分析不仅考虑输入条件，还要考虑输出空间产生的测试情况



软件边界与悬崖很类似



# 边界条件类型

如果软件测试问题包含确定的边界, 那么数据类型可能是:

- 数值
- 字符
- 位置
- 数量
- 速度
- 地址
- 尺寸
- .....

还要考虑数据类型的特征:

- 第一个/最后一个
- 最小值/最大值
- 开始/完成
- 空/满
- 最慢/最快
- 相邻/最远
- 超过/在内
- .....

# 测试边界线

- 测试临近边界的合法数据, 以及刚超过边界的非法数据.
- 越界测试通常简单地加1或很小的数(对于最大值)和减1或很小的数(对于最小值).

“报表日期”边界值分析法测试用例				
输入条件	测试用例说明	测试数据	期望结果	选取理由
报表日期的类型及长度	1个数字字符	5	显示出错	仅有1个合法字符
	5个数字字符	20015	显示出错	比有效长度少1
	7个数字字符	2001005	显示出错	比有效长度多1
	有1个非数字字符	2001.5	显示出错	只有1个非法字符
	全部是非数字字符	MAY---	显示出错	6个非法字符
	6个数字字符	200105	输入有效	类型及长度均有效
日期范围	在有效范围边界上选取数据	200101	输入有效	最小日期
		200512	输入有效	最大日期
		200100	显示出错	刚好小于最小日期
		200513	显示出错	刚好大于最大日期
月份范围	月份为1月	200101	输入有效	最小月份
	月份为12月	200112	输入有效	最大月份
	月份<1	200100	显示出错	刚好小于最小月份
	月份>12	200113	显示出错	刚好大于最大月份

## 6.4.3 错误推测法(error guessing)

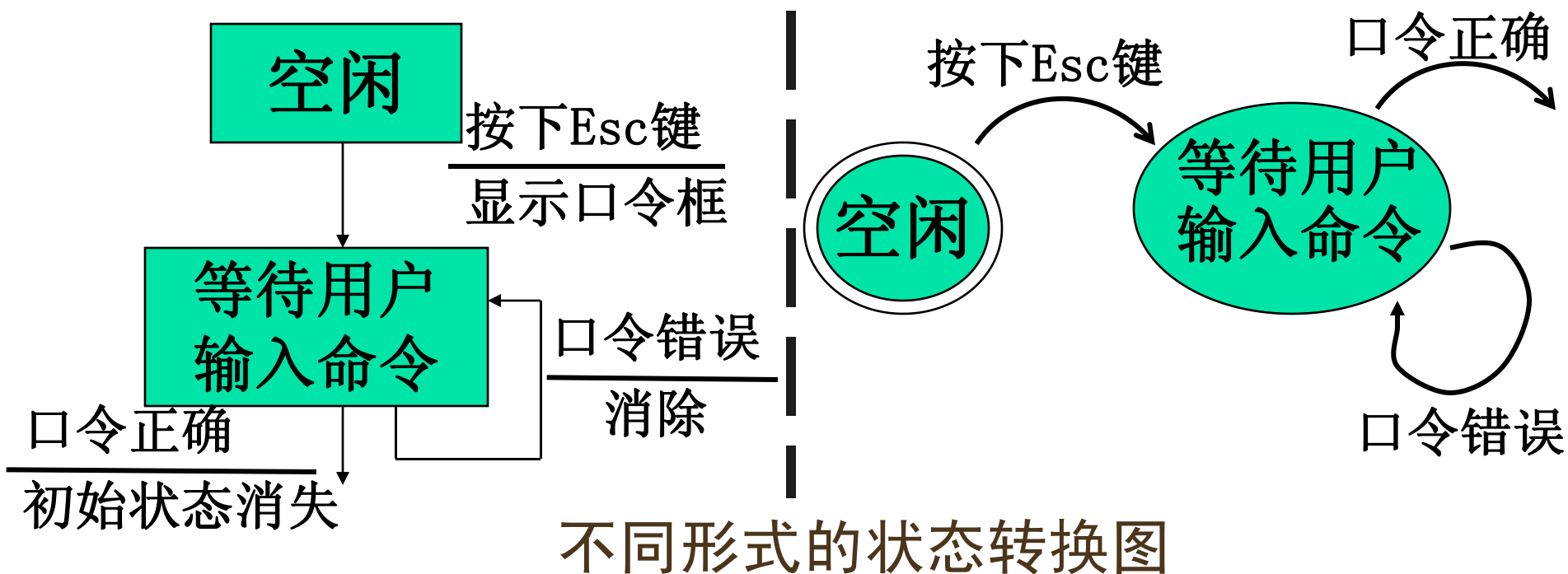
根据经验来设计测试用例的方法  
例如，数据测试中的：

- 缺省值
- 空白
- 空值
- 零值
- 无

## 6.4.4 状态测试

软件必须测试程序的状态及其转换。

- 测试软件的逻辑流程
- 建立状态转换图
- 减少要测试的状态及转换的数量



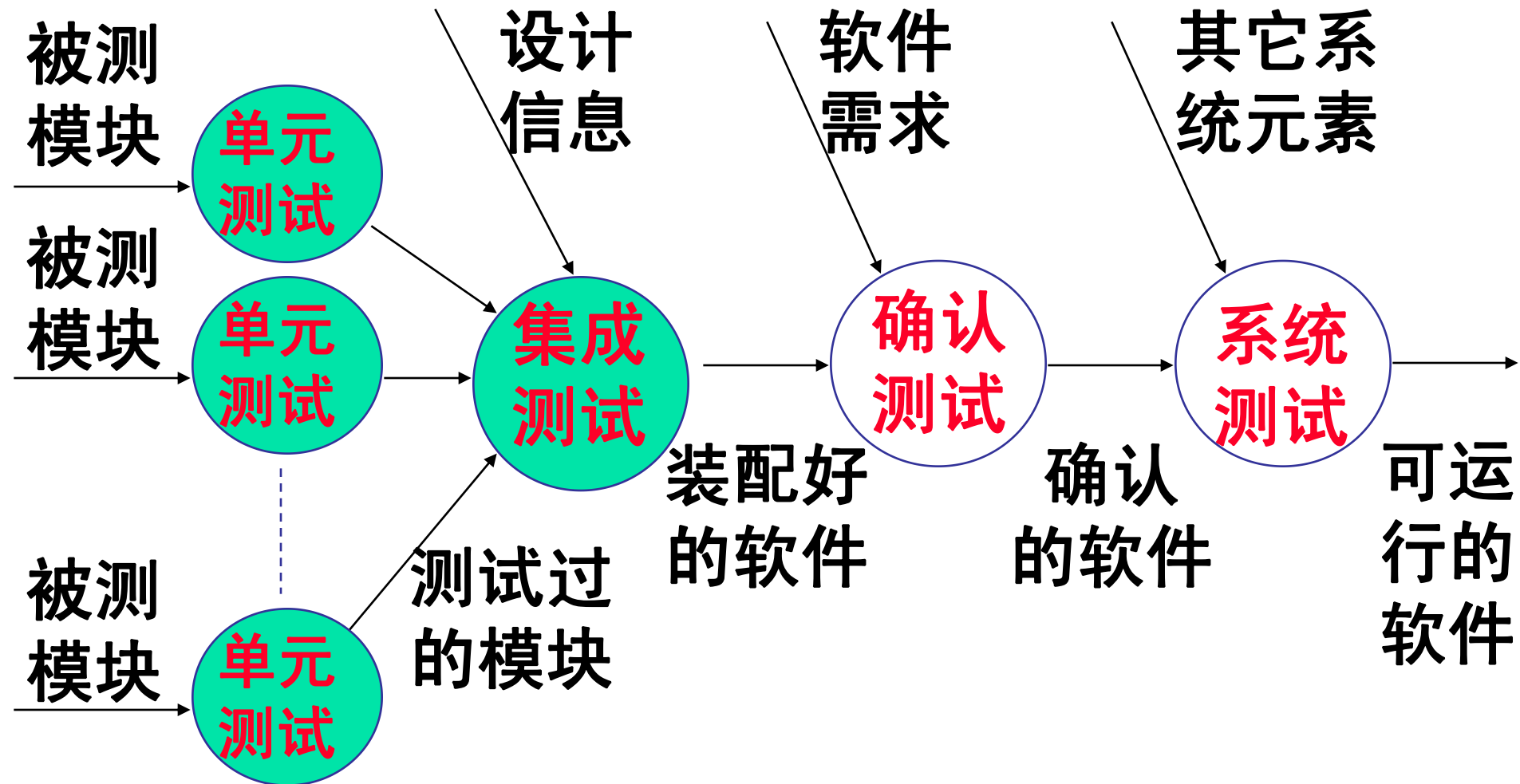
# 失败状态测试

找到测试软件失败的案例。

- 竞争条件和时序错乱
  - 重复
  - 压迫
  - 重负
- } 应联合使用，同时进行

# § 6. 6软件测试的步骤

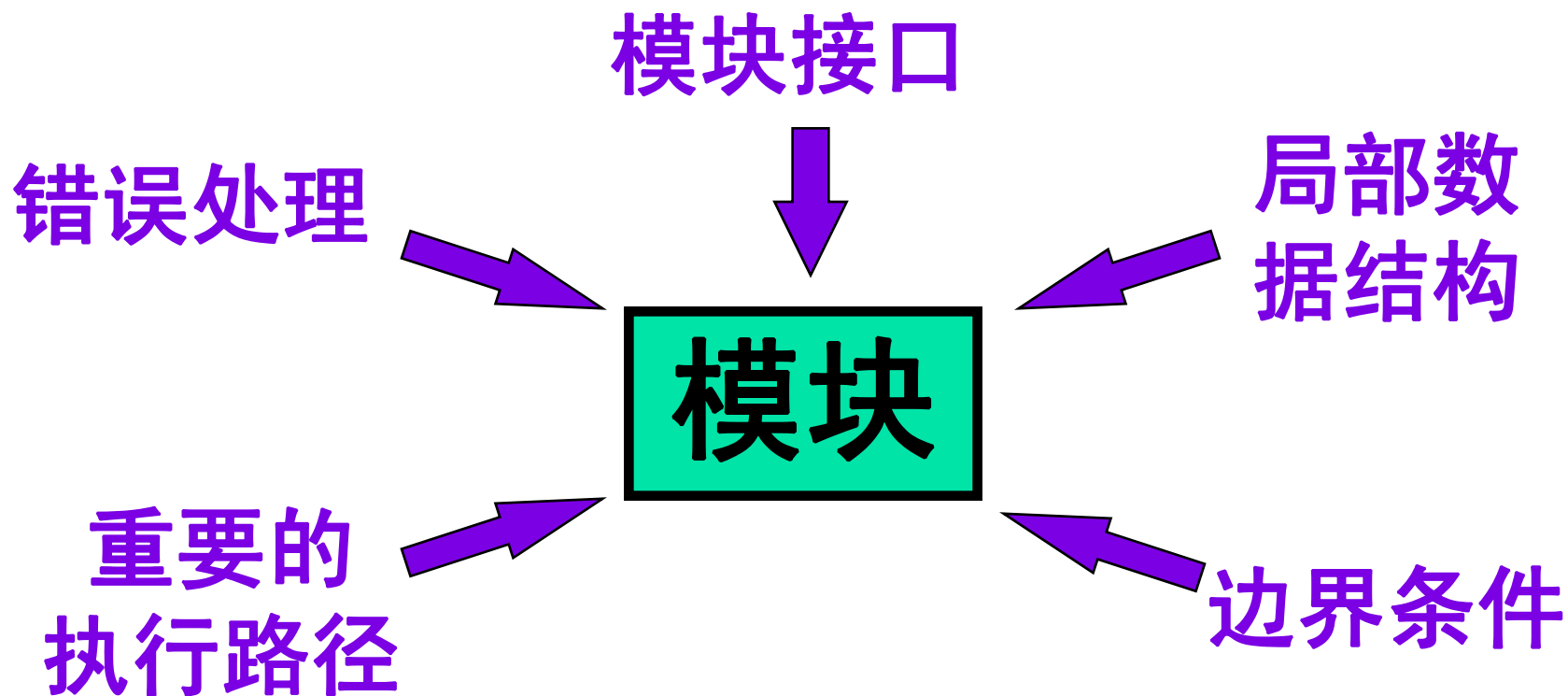
## 软件测试的过程



## 6.6.1 单元测试

### 一. 单元测试的内容

主要对模块的五个基本特性进行评价





## 6.6.2 集成测试(组装测试)

### 集成测试需考虑的问题：


- ❏ 数据穿越接口可能丢失.
- ❏ 一模块可能破坏另一模块功能.
- ❏ 子功能组装可能未产生所要求的主功能.
- ❏ 全程数据结构可能出问题.
- ❏ 误差累积问题.

# 集成测试方法

# 通常采用黑盒测试技术

## 实施策略：

## 👉 非渐增式测试


**渐增式测试**
 {
 自顶向下结合 { 深度优先  
 自底向上结合 { 广度优先
 }

# 一. 非渐增式集成方式

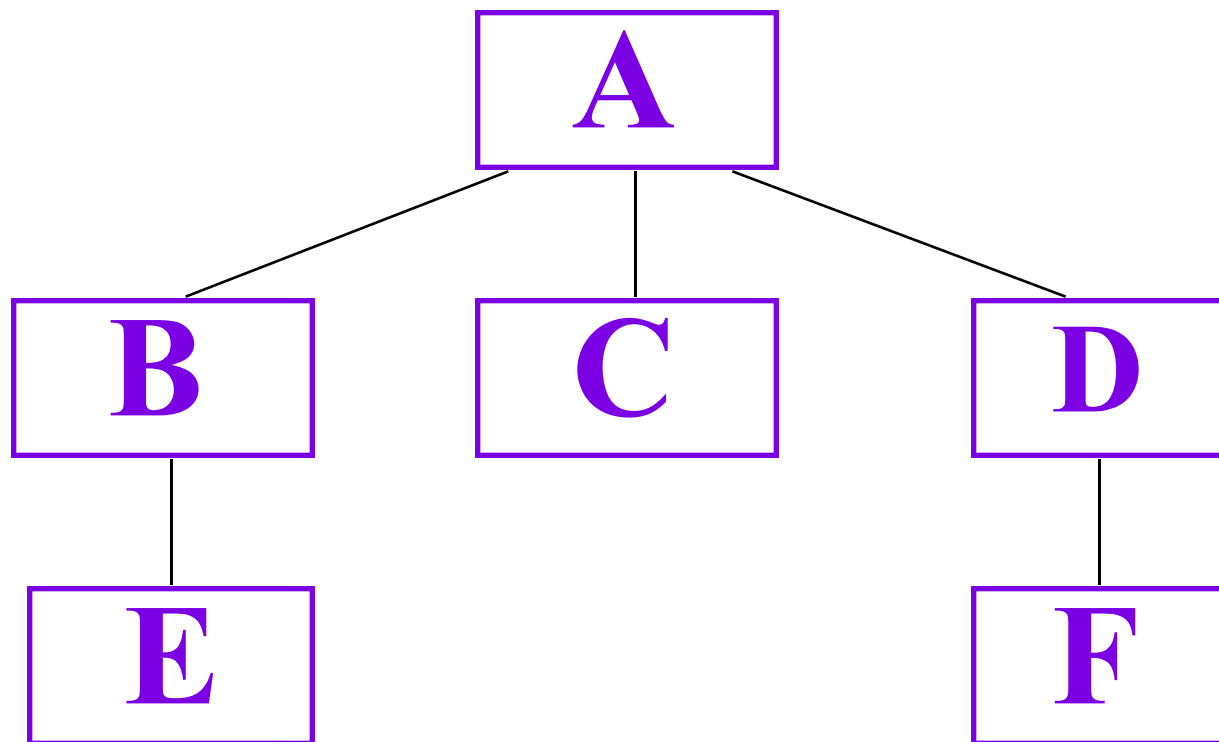
一次就把所有通过了单元测试的模块组合在一起进行全程序的测试.

**缺点:**发现错误难以诊断定位.  
又称“莽撞测试”.

## 二. 渐增式集成方式

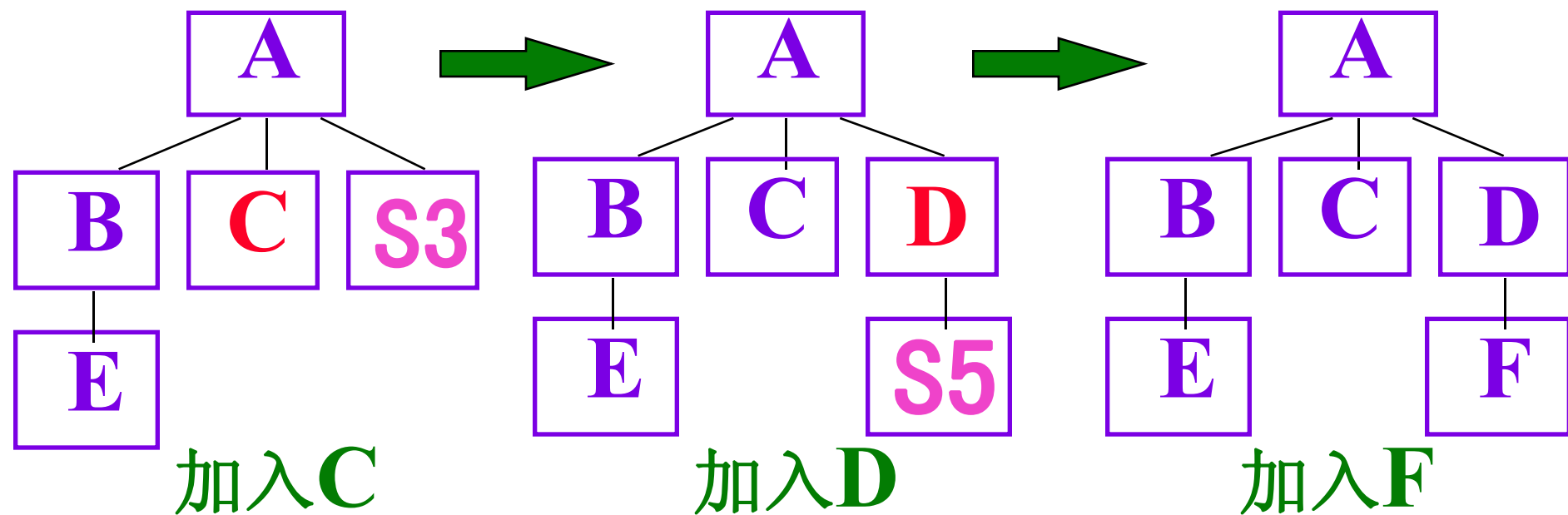
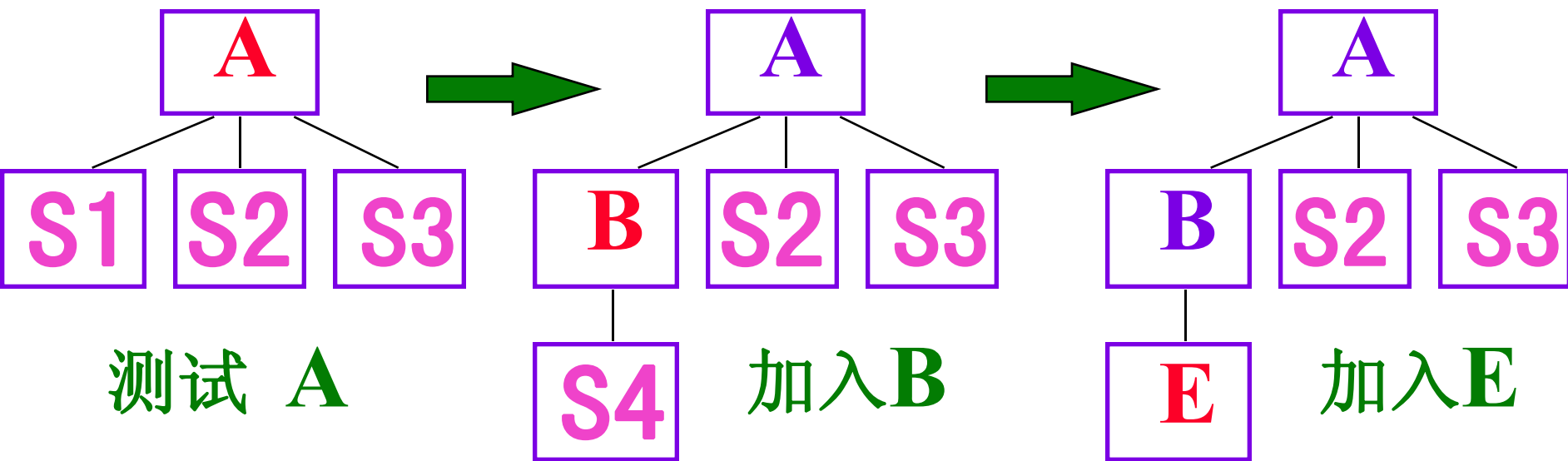
从一个模块开始，测一次添加一个模块，边组装边测试，以发现与接口相联系的问题。

# 自顶向下结合方式举例:

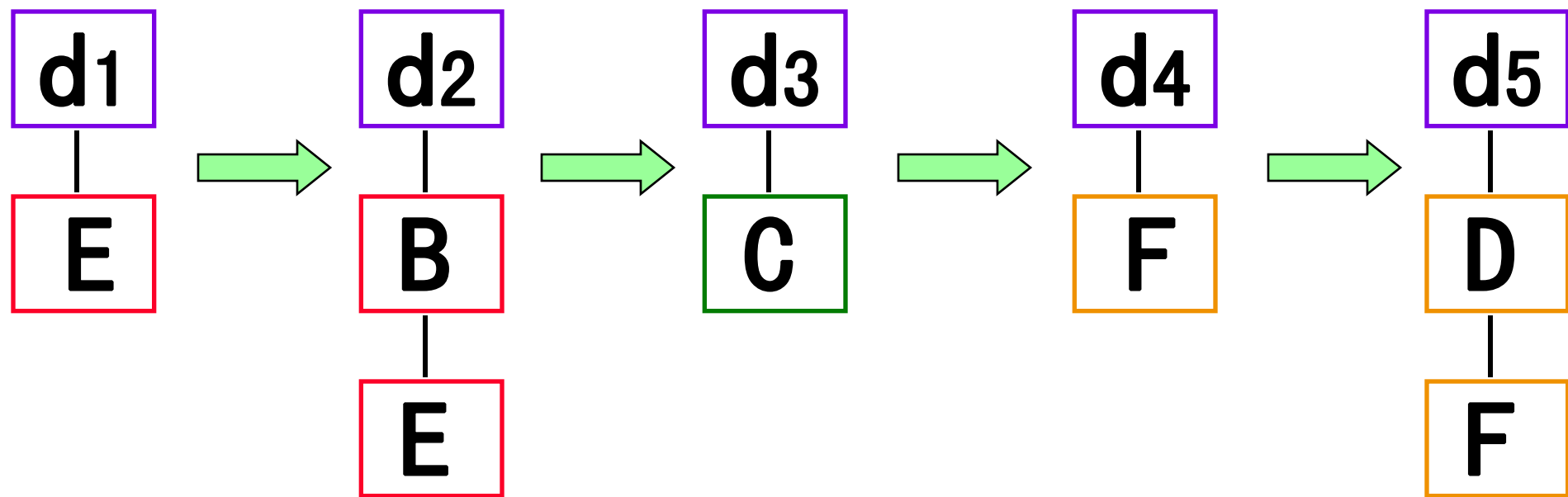
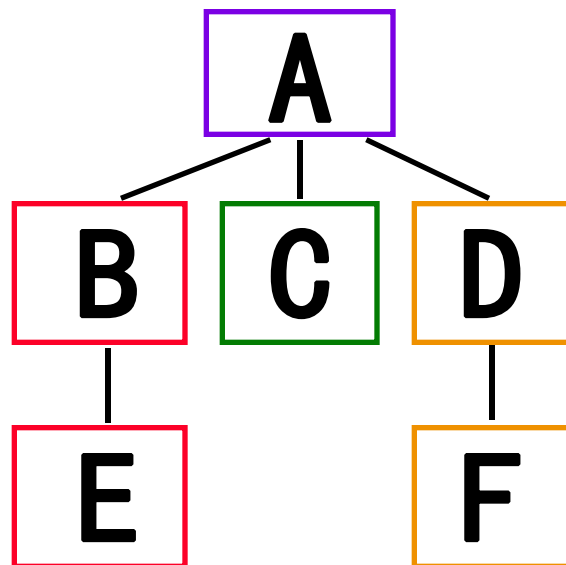


模块测试结合顺序 { 深度优先: A、B、E、C、D、F  
广度优先: A、B、C、D、E、F

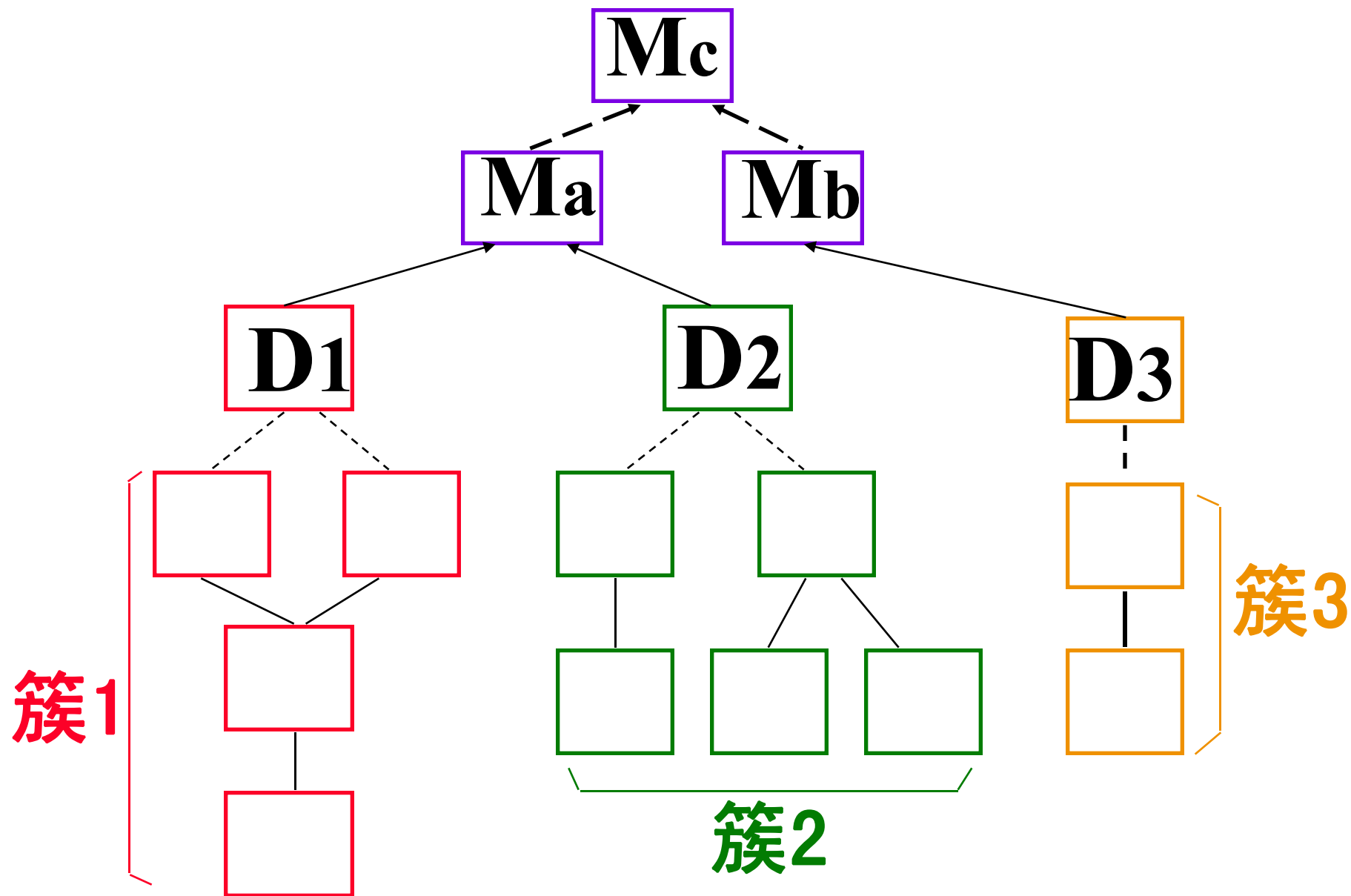
# 自顶向下结合方式举例: (深度优先)



# 自底向上结合方式举例:



# 自底向上结合方式举例:





# $\alpha$ 测试和 $\beta$ 测试

## $\alpha$ 测试 (Alpha)

在开发者的场所由用户进行, 在开发着关注和控制的环境下进行.

## $\beta$ 测试 (Beta)

最终用户在自己的场所进行.

# 系统测试

软件只是计算机系统的一个元素，软件最终要与其他系统元素（如新硬件、信息等）相结合，进行各种集成测试和确认测试。

# 用于系统测试的测试类型：

- (1) 恢复测试
- (2) 安全性测试
- (3) 强度测试
- (4) 性能测试