软件测试用例生成

Bixin Li
School of Computer Science and Engineering
Southeast University

软件测试用例设计

- 黑盒测试（基于功能的测试用例设计）
- 白盒测试（基于结构的测试用例设计）
- 自动测试用例生成方法
- 测试用例设计综合策略

1 黑盒测试

- 1.1 系统功能细分
- 1.2 等价类划分
- 1.3 因果图和判定表
- 1.4 边值分析

1.1系统功能细分

1. 2等价类划分

- 干什么——选取测试用例
- 怎么做——等价类划分的办法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据当作测试用例
- 在分析需求规格说明的基础上划分等价类，列出等价类表

等价类

- 等价类是指输入域的某个子集合
- 如果用等价类中的一个输入条件作为测试数据进行测试不能发现程序中的错误，那么使用等价类中的其它输入条件进行测试也不可能发现错误
- 也就是说，对揭露程序中的错误来说，等价类中的每个输入条件是等效的

有效等价类和无效等价类

- 在考虑等价类时，应该注意区别两种不同的情况：
- *有效等价类：有效等价类指的是对程序的规格说明是有意义的、合理的输入数据所构成的集合。在具体问题中，有效等价类可以有一个，也可以是多个。
- *无效等价类：无效等价类指对程序的规格说明是不合理的或无意义的输入数据所构成的集合。对于具体的问题，无效等价类至少应有一个，也可能有多个。

确定等价类原则

1．如果输入条件规定了取值范围或值的个数，则可确定一个有效等价类和两个无效等价类。

- 输入条件：…项数可以从1到999…
  - 有效等价类为　　“1≤ 项数≤ 999”
  - 无效等价类为　　“项数<1”及 “项数>999”
- 值的个数　　“学生选课允许2门至4门”
  - 有效等价类：　　选课2至4门
  - 无效等价类：　　只选一门课或未选课
  - 　　选课超过4门

确定等价类原则

2．输入条件规定了输入值的集合，或是规定了“必须如何”的条件，则可确定一个有效等价类和一个无效等价类。

例：

“必须如何”	“标识符以字母开头”
有效等价类：	以字母开头的字符串
无效等价类：	以非字母开头的字符串

3．如果确知，已划分的等价类中各元素在程序中的处理方式是不同的，则应将此等价类进一步划小。

确定测试用例（原则）

1．设计一个测试用例，使其尽可能多地覆盖有效等价类，重复这一步，最终使得所有有效等价类均被覆盖。

2．设计一个测试用例，使其只覆盖一个无效等价类，重复这一步，最终使得所有无效等价类均被覆盖。

一个例子

- FORTRAN编译系统的DIMENSION语句的语法规则：
  DIMENSION语句可以用来规定数组的维数，其形式为：
  DIMENSION ad[,ad]…
- 其中ad 为数组描述符，形式为n(d[,d]…)
  其中n为数组名，由1个到6个字母或数字组成，为首的必须是字母；d是维数说明符，数组维数最大为7最小为1，形式为：[lb:]ub，lb和ub分别表示数字下标的下界和上界，均为-65534 - 65535之间的整数，也可以是整数型变量名(但不可以是数组元素名)。若未规定lb，则其值认为为1，且ub>=lb。若已规定lb，则它可以为负数、零、正数。
- DIMENSION语句可写多行。

FORTRAN语言的DIMENSION语句

AD

DIMENSION ALPHA(2:5, 3:10),
BETA(-4:4, 1:7), ……

n d d

d = [lb:]ub

1~6 1~7 -65534~65535

字符数 维数 界值

[解]第一步　确定输入条件，列出等价类表

输入条件	有效等价类	无效等价类
数组描述符个数	1(1), >1(2)	无数组描述符　(3)
数组名字符个数	1－6（4）	0(5), >6(6)
数组名	有字母 (7)，有数字 (8)	有其他字符　(9)
数组名以字母开头	是 (10)	否 (11)
数组维数	1-7(12)	0(13), >7(14)
上界是	常数 (15)，整型变量 (16)	数组元素名 (17)，其他 (18)
整数变量名	有字母 (19)，有数字 (20)	其他 (21)
整数变量名以字母开头	是 (22)	否 (23)
上下界取值	-65,534 – 65,535(24)	< -65534(25), > 65535(26)
是否定义下界	是 (27)，否 (28)	
上界对下界关系	>(29), =(30)	< (31)
下界定义为	负数 (32), 0(3 3)，正数 (34)	
下界是	常数 (35)，整型变量 (36)	数组元素名 (27)，其他 (38)
语句多于一行	是 (39)，否 (40)	

例 2 的等价类表

第二步　确定测试用例

先设计一个测试用例，使其覆盖一个或多个有效等价类。

如：DIMENSION A(2)

能覆盖有效等价类1，4，7，10，12，15，24，28，29和40。

为覆盖其它有效等价类，需设计另外的测试用例。如：

DIMENSION A12345(I, 9, J4XXXX, 65535, 1,KLM, 100),
BBB(-65534: 100, 0: 1000, 10: 10, I: 65535)

它可覆盖其余的有效等价类（2，8，16，19，20，22，27，30，32，33，34，35，36，39）。

再设计其它测试用例，使每个只覆盖一个无效等价类，直至覆盖完为止。这些测试用例是（下面各行左端括号内的数字为等价类号）：

(3) DIMENSION
(5) DIMENSION (10)
(6) DIMENSION A234567(2)
(9) DIMENSION A.1(2)
(11) DIMENSION 1A(10)

确定测试用例（续）

(13) DIMENSION B
(14) DIMENSION B(4,4,4,4,4,4,4,4)
(17) DIMENSION B(4, A(2))
(18) DIMENSION B(4, , 7)
(21) DIMENSION C(I. , 10)
(23) DIMENSION C(10, 1J)
(25) DIMENSION D(-65535:1)
(26) DIMENSION D(65536)
(31) DIMENSION D(4:3)
(37) DIMENSION D(4(2): 4)
(38) DIMENSION D(. : 4)

连同前面两个共计18个测试用例，他们覆盖了全部等价类。

经典例子

- “输入三个整数作为三边的边长构成三角形。当此三角形为一般三角形、等腰三角形及等边三角形时，分别做计算…”
- 注意输入和输出条件

		有效等价类型	号码	无效等价类	号码
输入条件	输入三个整数	整数	1	一边为非整数：a 为非整数	12
				一边为非整数：b 为非整数	13
				一边为非整数：c 为非整数	14
				两边为非整数：a,b 为非整数	15
				两边为非整数：b,c 为非整数	16
				两边为非整数：a,c 为非整数	17
				三边 a，b，c 均为非整数	18
		三个数	2	只给一边：只给 a	19
				只给一边：只给 b	20
				只给一边：只给 c	21
				只给两边：只给 ab	22
				只给两边：只给 b,c	23
				只给两边：只给 ac	24
				给出三个以上	25
		非零数	3	一边为零：a 为 0	26
				一边为零：b 为 0	27
				一边为零：c 为 0	28
				二边为零：a,b 为 0	29
				二边为零：b,c 为 0	30
				二边为零：a,c 为 0	31
				三边 a,b,c 均为 0	32
		正数	4	一边 <0：$a<0$	33
				一边 <0：$b<0$	34
				一边 <0：$c<0$	35
				二边 <0：$a<0$ 且 $b<0$	36
				二边 <0：$a<0$ 且 $c<0$	37
				二边 <0：$b<0$ 且 $c<0$	38
				三边均 <0：$a<0$ 且 $b<0$ 且 $c<0$	39
输出条件	构成一般三角形	$a+b>c$	5	$a+b<c$	40
				$a+b=c$	41
		$b+c>a$	6	$b+c<a$	42
				$b+c=a$	43
		$a+c>b$	7	$a+c<b$	44
				$a+c=b$	45
	构成等腰三角形	$a=b$ 且两边之和大于第三边	8		
		$b=c$ 且两边之和大于第三边	9		
		$a=c$ 且两边之和大于第三边	10		
	构成等腰三角形	$a=b=c$	11		

表 4.1 例 1 的等价类型表

有效等价类

- 覆盖有效等价类的测试用例：

a	b	c	覆盖等价类号码
3	4	5	（1）--（7）
4	4	5	（1）--（7），（8）
4	5	5	（1）--（7），（9）
5	4	5	（1）--（7），（10）
4	4	4	（1）--（7），（11）

无效等价类

a	b	c	覆盖等价类号码
2.5	4	5	12
3	4.5	5	13
3	4	5.5	14
3.5	4.5	5	15
3	4.5	5.5	16
3.5	4	5.5	17
4.5	4.5	5.5	18
3			19
	4		20
		5	21
3	4		22
	4	5	23
3		5	24
3	4	5	25
0	4	5	26
3	0	5	27
3	4	0	28

a	b	c	覆盖等价类号码
0	0	5	29
3	0	0	30
0	4	0	31
0	0	0	32
-3	4	5	33
3	-4	5	34
3	4	-5	35
-3	-4	5	36
-3	4	-5	37
3	-4	-5	38
-3	-4	-5	39
3	1	5	40
3	2	5	41
3	1	1	42
3	2	1	43
1	4	2	44
3	4	1	45

问题讨论

问　　题：给出下面的有效和无效等价类

- 输入条件：“标识符应以字母开头…”
- 输入条件：长度为1-20的字符串
- 输入条件：数据库中的值域, CHAR(20), NOT NULL

问题讨论

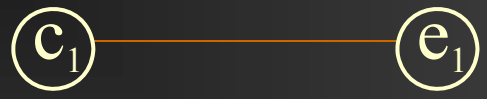
- 等价类划分与覆盖的关系?
- 覆盖到每个有效等价类和无效等价类是否意味着肯定正确?
- 100%覆盖是否意味着肯定正确？为什么？

1.3 因果图

- 因果图方法（Cause—Effect Graphics）:一种黑盒测试方法
- 方法的依据:
  需求规格说明中的因果关系
- 能够帮助我们按一定步骤，高效率地选择测试用例,同时还指出，程序规格说明描述中存在着的问题。

因果图介绍

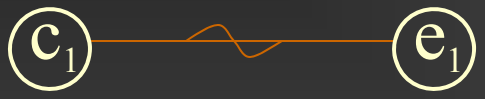
恒等



$c_1=1 \Rightarrow e_1=1$

$c_1=0 \Rightarrow e_1=0$

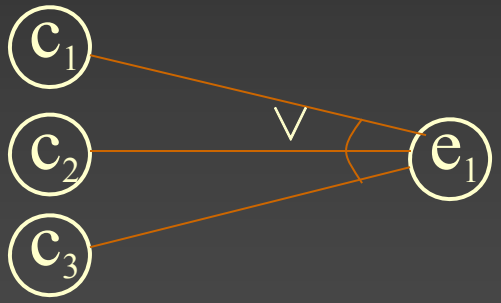
非



$c_1=1 \Rightarrow e_1=0$

$c_1=0 \Rightarrow e_1=1$

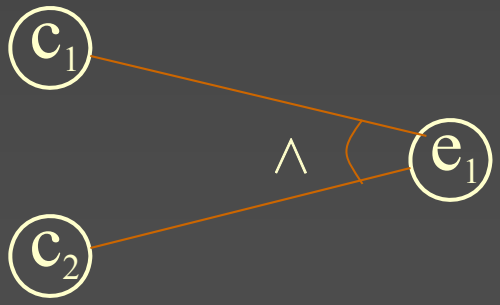
或



$c_1=1$ 或$c_2=1$ 或$c_3=1$ $\Rightarrow e_1=1$

否则 $\Rightarrow e_1=0$

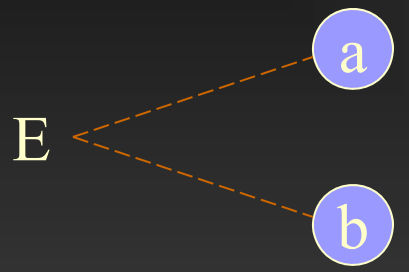
与



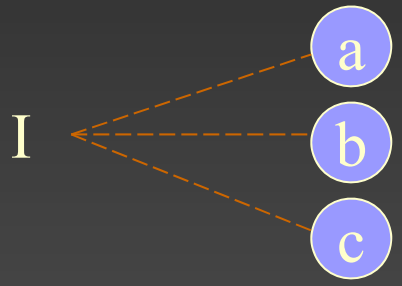
$c_1=1$ 且$c_2=1$ $\Rightarrow e_1=1$

否则 $\Rightarrow e_1=0$

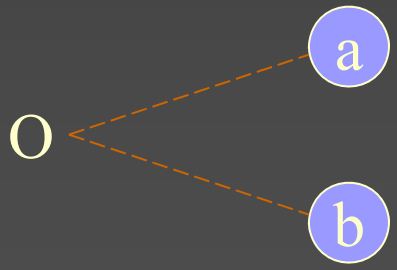
输入条件的约束



E约束（异）：a,b中至多有一个可能为1
即a和b不能同时为 1



I 约束（或）：a,b和c中至少有一个
必须是1,即a、b和c不能同时为0



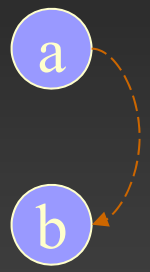
O约束（唯一）：a和b中必须有一个
且仅有一个为 1

输入条件的约束...

R

R约束（要求）：a是 1 时，b必须是 1
即不可能a是 1 时b为 0

输出条件的约束



M

M约束（强制）：若结果a是 1 时，
则结果b强制为 0

步　骤

①分析程序规格说明的描述中，哪些是原因，哪些是结果。原因常常是输入条件或是输入条件的等价类。而结果是输出条件。

②分析程序规格说明的描述中语义的内容，并将其表示成连接各个原因与各个结果的“因果图”。

步　骤...

③由于语法或环境的限制，有些原因和结果的组合情况是不可能出现的。为表明这些特定的情况，在因果图上使用若干个特殊的符号标明约束条件。

④把因果图转换成判定表。

⑤把判定表中每一列表示的情况写成测试用例。

例　子

- 软件规格说明书:

“...第一列字符必须是A或B，第二列字符必须是一个数字，在此情况下进行文件的修改。但如果第一列字符不正确，则给出信息L，如果第二列字符不是数字，则给出信息M。...”

原因和结果

原因：

1——第一列字符是A；

2——第一列字符是B；

3——第二列字符是一数字。

结果：

21——修改文件；

22——给出信息L；（第一列字符不正确）

23——给出信息M。（第二列字符不是数字）

因果图和具有约束的因果图

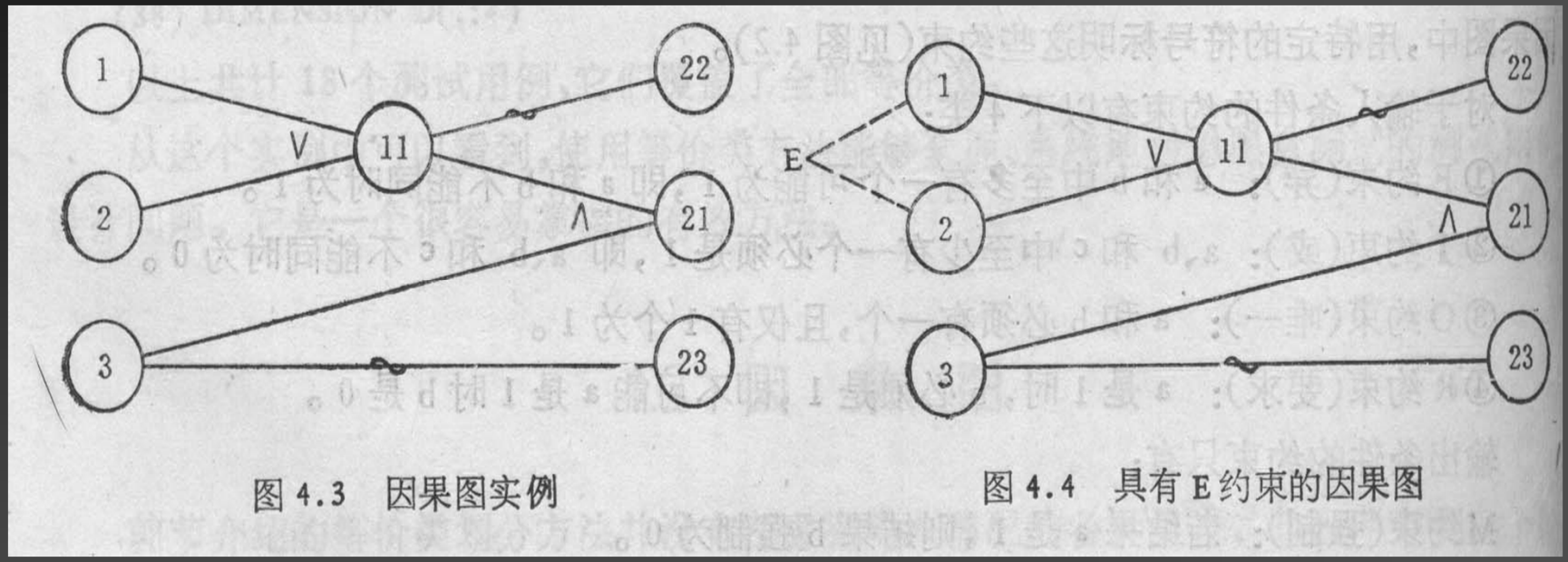


图 4.3　因果图实例

图 4.4　具有 E 约束的因果图

- 11为中间节点；
- 考虑到原因1和原因2不可能同时为1，因此在因果图上施加E约束。

根据因果图建立如下的判定表

		1	2	3	4	5	6	7	8	
条件	1	1	1	1	1	0	0	0	0	原因
	2	1	1	0	0	1	1	0	0	
	3	1	0	1	0	1	0	1	0	
	11	//////	//////	1	1	1	1	0	0	
动作	22	//	//	0	0	0	0	1	1	结果
	21	//	//	1	0	1	0	0	0	
	23	//	//	0	1	0	1	0	1	
测试用例		// //////	// //////	A3 A8	AM A?	B5 B4	BN B!	C2 X6	DY P;	

表中8种情况的左面两列情况中，原因①和原因②同时为1，这是不可能出现的，故应排除这两种情况。表的最下一栏给出了6种情况的测试用例，这是我们所需要的数据。

1.4 边值分析

- 在软件设计和程序编写中，常常对于规格说明中的输入域边界或输出域边界不够注意，以致形成一些差错。
- 实践证明，在设计测试用例时，对边界附近的处理必须给予足够的重视，为检验边界附近的处理专门设计测试用例，常常取得良好的测试效果。

边值分析遵循的原则

- **程序中边界值问题**

①如果输入条件规定了取值范围，或是规定了值的个数，应以该范围的边界内及刚刚超出范围的边界外的值，或是分别对最大、最小个数及稍小于最小、稍大于最大个数作为测试用例。

例如，如果程序的规格说明中规定：“重量在10公斤至50公斤范围内的邮件，其邮费计算公式为……”。作为测试用例，我们应取10及50，还应取10．01，49.99，9．99及50．01等。如果另一问题规格说明规定：“某输入文件可包含1至255个记录，……”，则测试用例可取1和255，还应取0及256等。

遵循以下几条原则

②针对规格说明的每个输出条件使用前面的第①条原则。

例如，某程序的规格说明要求计算出“每月保险金扣除额为0至1165．25元”，其测试用例可取0．00及1165．2、还可取一0．01及1165．26等。如果另一程序属于情报检索系统，要求每次”最多显示1-4条情报摘要”，这时我们应考虑的测试用例包括1和4，还应包括0和5等。

遵循以下几条原则

③如果程序规格说明中提到的输入或输出域是个有序的集合（如顺序文件、表格等），就应注意选取有序集的第一个和最后一个元素作为测试用例。

④分析规格说明，找出其它的可能边界条件。

例子

某个“为学生考试试卷评分和成绩统计”的程序，其规格说明指出了对程序的要求：程序的输入文件由80个字符的一些记录组成，这些记录分为三组：

① 标题

这一组只有一个记录，其内容为输出报告的名字。

②试卷各题标准答案记录

每个记录均在第80个字符处标以数字“2”。该组的第一个记录的第1至第3个字符为题目编号（取值为1—999）。第10至第59个字符给出第1至第50题的答案（每个合法字符表示一个答案）。该组的第2，第3个记录相应为第51至第100，第101至第150，…题的答案。

例子

③每个学生的答卷描述

该组中每个记录的第80个字符均为数字“3”。每个学生的答卷在若干个记录中给出。如甲的首记录第1至第9字符给出学生姓名及学号，第10至第59字符列出的是甲所做的第1至第50题的答案。若试题数超过50，则第2，第3纪录分别给出他的第51至第100，第101至第150……题的解答。然后是学生乙的答卷记录。

若学生最多为200人，输入数据的形式如图4. 15所示。

例　子

学生考卷评分和成绩统计程序输入数据的形

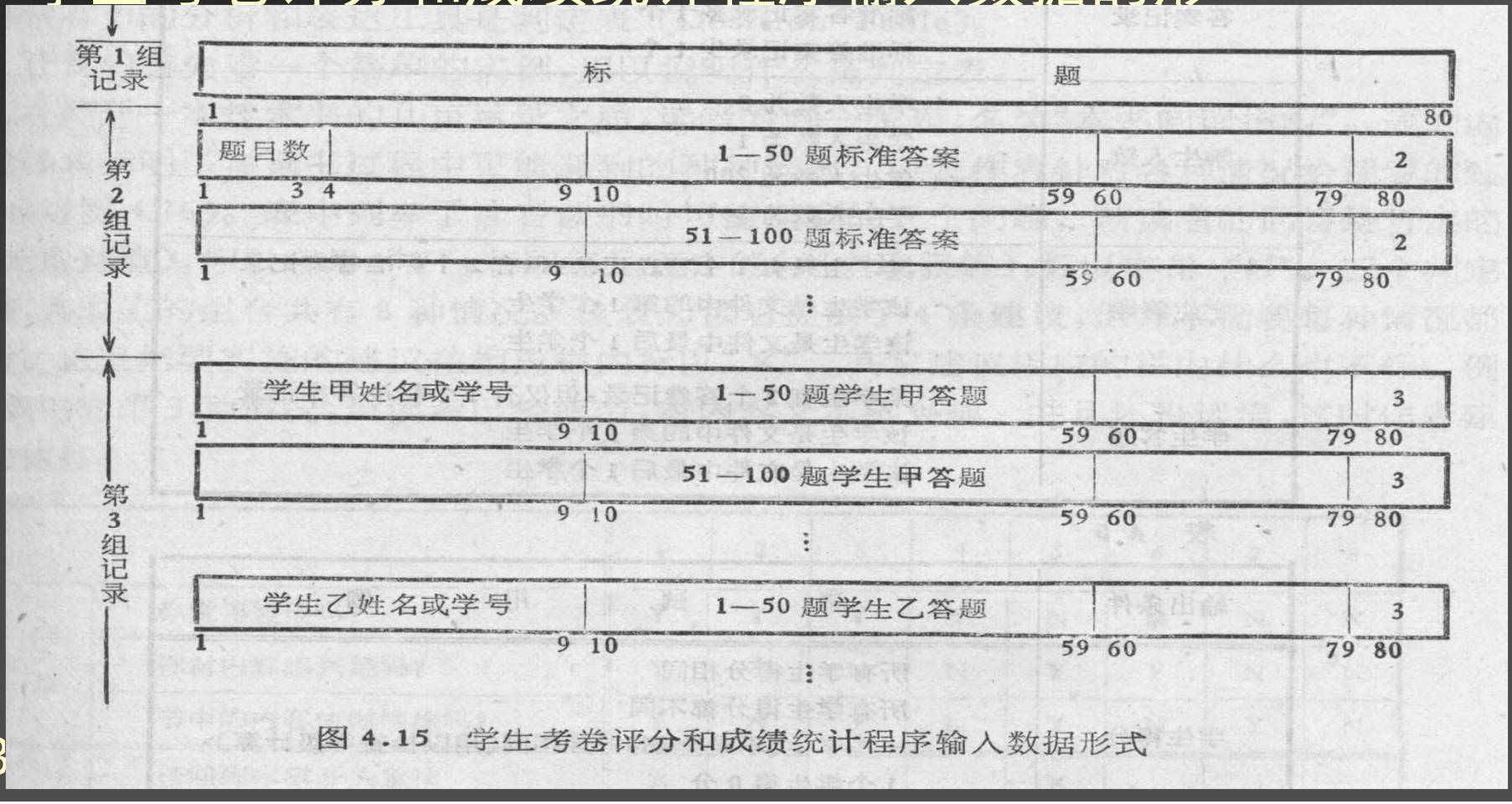


图 4.15　学生考卷评分和成绩统计程序输入数据形式

例　　子

该程序应给出4个输出报告，即：

①按学生学号排序，每个学生的成绩（答对的百分比）和等级报告。

②按学生得分排序，每个学生的成绩。

③平均分数，以及最高与最低分之差。

④按题号排序，每题学生答对的百分比

例　子

输入条件	测试用例
输入文件	空输入文件
标题	无标题记录 只有1 个字符的标题 具有80 个字符的标题
出题个数	出了1 个题 出了50 个题 出了51 个题 出了100 个题 除了999 个题 没有出题 题目是非数值量
答案记录	标题记录后没有标准答案记录 标准答案记录多1 个 标准答案记录少1 个
学生人数	学生人数为0 学生人数为1 学生人数为200 学生人数为201
学生答题	某学生只有1 个答卷记录，但有两个标准答案记录 该学生是文件中的第1 一个学生 该学生是文件中的最后一个学生
学生答题	某学生有2 个答卷记录，但只有1 个标准答案记录 该学生是文件中的第1 个学生

输出条件	测试用例
学生得分	所有学生得分相同 所有学生得分不同 一些学生（不是全部）得分相同（用以计算等级） 1 个学生得0 分 1 个学生得100 分
输出报告(1),(2)	1 个学生编号最小（检查排序） 1 个学生编号最大 学生数恰好使报告印满一页（检查打印） 学生数使报告1 页打印不够，恰好多1 人
输出报告（3）	平均值取最大值（所有学生都得满分） 平均值为0（所有学生都得0 分） 标准偏差取最大值（1 学生得0 分，1 学生得100 分） 标准偏差相同（所有学生得分相同）
输出报告(4)	所有学生都答对第1 题 所有学生都答错第1 题 所有学生都答对最后1 题 所有学生都答错最后1 题 题数恰好使报告打印在1 页上

讨　论

- 边值分析是最常用的测试用例设计方法。
- 软件开发人员如何有把握地说，“我的软件好像没问题了”。（猜测各种可能出现的情况）

讨　论

- 工程设计或者事务处理项目中经常使用判定表。
- 判定表在软件开发和测试中的作用。

讨　论

- 测试用例设计方法（逻辑驱动，等价类划分，因果图分析，边值分析，判定表驱动）：对需求进行分析，找到尽可能多的有代表性的测试用例；
- 不局限于一种方法，可以组合使用多种方法；
- 还可以采用其他有效的方法；

2 白盒测试方法

- 程序结构分析
- 逻辑覆盖
- 程序插装

2.1 程序结构分析

- 控制流分析
- 数据流分析
- 信息流分析

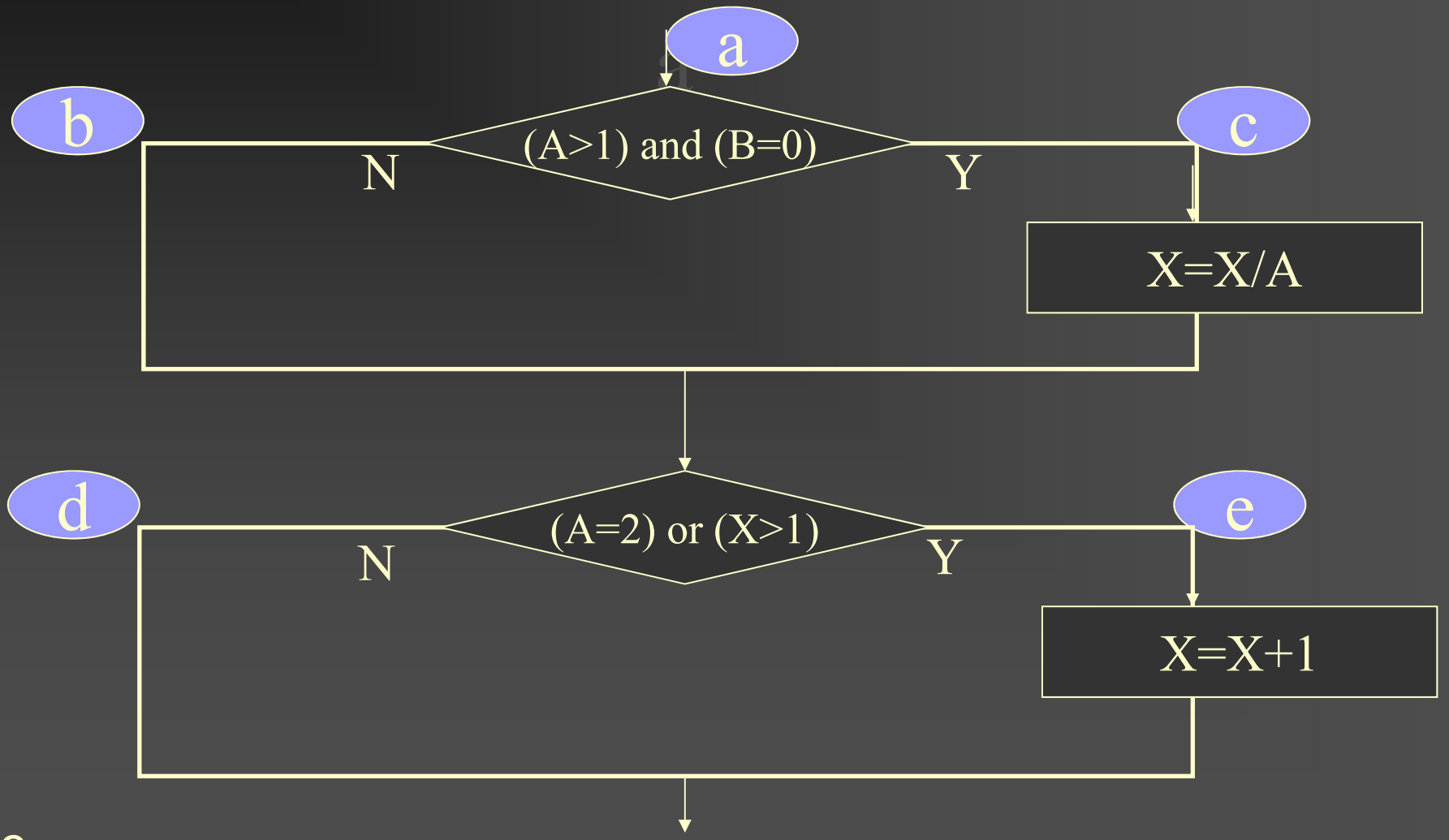
2.2 逻辑覆盖

语句覆盖
判定覆盖
条件覆盖
判定/条件覆盖
条件组合覆盖
路径覆盖

逻辑覆盖

- 人员使用逻辑驱动测试方法，主要想对程序模块进行如下的检查：
  - 对程序模块的所有独立的执行路径至少测试一次；
  - 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；
  - 在循环的边界和运行界限内执行循环体；
  - 测试内部数据结构的有效性，等。

被测程序段流程图



2.2.1 语句覆盖

- 程序中每一可执行语句至少执行一次

A=2
B=0　　a c e
X=3

- 此例仅用一个测试用例，通常需要多个

语句覆盖的缺陷

- 从程序中每个语句都得到执行这一点来看，语句覆盖的方法似乎能够比较全面地检验每一个语句。
- 判定条件错误检查不到
- “语句覆盖”是很不充分的一种标准,仍然很弱。

2.2.2 判定覆盖

- 程序中每个判定的取真分支和取假分支至少执行一次（也称分支覆盖）

{ A=2, B=0, X=3 } a c e 及 { A=1, B=0, X=1 } a b d

或

{ A=3, B=0, X=3 } a c d 及 { A=2, B=1, X=1 } a b e

判定覆盖

除了双值的判定语句外，还有多值判定语句，如CASE语句，所以“判定覆盖”更一般的含义是：使得每一个判定获得每一种可能的结果至少一次。

“判定覆盖”比“语句覆盖”更强。上述两组测试用例不仅满足了“判定覆盖”，同时还做到了“语句覆盖”。

如果程序段中的第2个判断条件X＞1如果错写为X＜1，使用上述测试用例5，照样能按原路径执行（abe），而不影响结果。

只达到判定覆盖仍无法确定判断内部条件的错误。需要有更强的逻辑覆盖准则去检验判断内的条件。

2.2.3 条件覆盖

- 使程序的判定中每个条件的真假取值至少满足一次

例中设条件 $\begin{pmatrix} A>1 \\ B=0 \\ A=2 \\ X>1 \end{pmatrix}$ 取真表为 $\begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix}$ 取假表为 $\begin{pmatrix} \overline{T}_1 \\ \overline{T}_2 \\ \overline{T}_3 \\ \overline{T}_4 \end{pmatrix}$

A B X	经历	覆盖分支	覆盖条件
2 0 3	a c e	c e	T_1 T_2 T_3 T_4
1 0 1	a b d	b d	$\overline{T}_1$ T_2 $\overline{T}_3$ $\overline{T}_4$
2 1 1	a b e	b e	T_1 $\overline{T}_2$ T_3 $\overline{T}_4$
		4分支	4条件8取值

条件覆盖

另一例：

A B X	经历	覆盖分支	覆盖条件
1 0 3	a b e	b e	$\overline{T}_1$ T_2 $\overline{T}_3$ T_4
2 1 1	a b e	b e	T_1 $\overline{T}_2$ T_3 $\overline{T}_4$
		2分支	4条件8取值

覆盖了条件的测试用例不一定覆盖分支。事实上，它只覆盖了4个分支中的两个（b和e）。为解决这一矛盾，需要兼顾条件和分支。

2.2.4 判定/条件覆盖

- 判定 / 条件覆盖要求设计足够的测试用例，使得判定中每个条件的所有可能（真 / 假）至少出现一次，并且每个判定本身的判定结果（真 / 假）也至少出现一次。

2.2.5 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。

上例中需考虑4个条件的8种组合

① $A > 1, B = 0$ $T_1 T_2$

② $A > 1, B \neq 0$ $T_1 \overline{T}_2$

③ $A \leq 1, B = 0$ $\overline{T}_1 T_2$

④ $A \leq 1, B \neq 0$ $\overline{T}_1 \overline{T}_2$

⑤ $A = 2$ $X > 1$ $T_3 T_4$

⑥ $A = 2$ $X \leq 1$ $T_3 \overline{T}_4$

⑦ $A \neq 2$ $X > 1$ $\overline{T}_3 T_4$

⑧ $A \neq 2$ $X \leq 1$ $\overline{T}_3 \overline{T}_4$

A B X	覆盖组号	经历	覆盖条件
2 0 3	① ⑤	a c e	$T_1 T_2 T_3 T_4$
2 1 1	② ⑥	a b e	$T_1 \overline{T}_2 T_3 \overline{T}_4$
1 0 2	③ ⑦	a b e	$\overline{T}_1 T_2 \overline{T}_3 T_4$
1 1 1	④ ⑧	a b d	$\overline{T}_1 \overline{T}_2 \overline{T}_3 \overline{T}_4$

条件组合覆盖

满足“条件组合覆盖”的测试用例是一定满足“判定覆盖”、“条件覆盖”和“判定/条件覆盖”的。

这一程序段共有四条路径。以上4个测试用例固然覆盖了条件组合，同时也覆盖了4个分支，但仅覆盖了3条路径，却漏掉了路径acd。

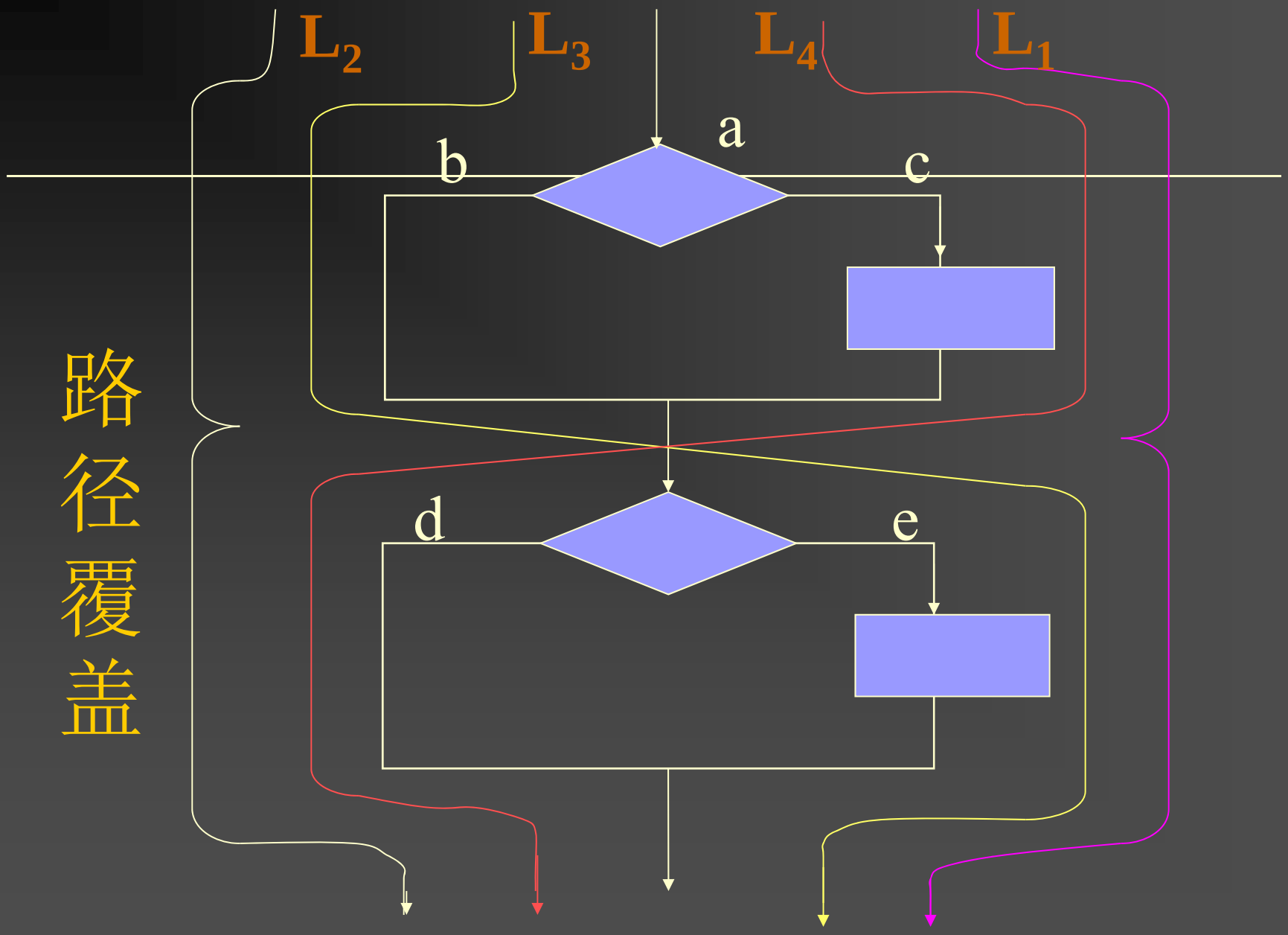
前面讨论的多种覆盖准则，有的虽提到了所走路径问题，但尚未涉及到路径的覆盖，而路径能否全面覆盖在软件则试中是个重要问题，因为程序要取得正确的结果，就必须消除遇到的各种障碍，沿着特定的路径顺利执行。如果程序中的每一条路径都得到考验，才能说程序受到了全面检验。

2.2.6 路径覆盖

覆盖程序中所有可能的路径

A B X	覆盖路径
2 0 3	a c e L_1
1 0 1	a b d L_2
2 1 1	a b e L_3
3 0 1	a c d L_4

路径覆盖



路径覆盖

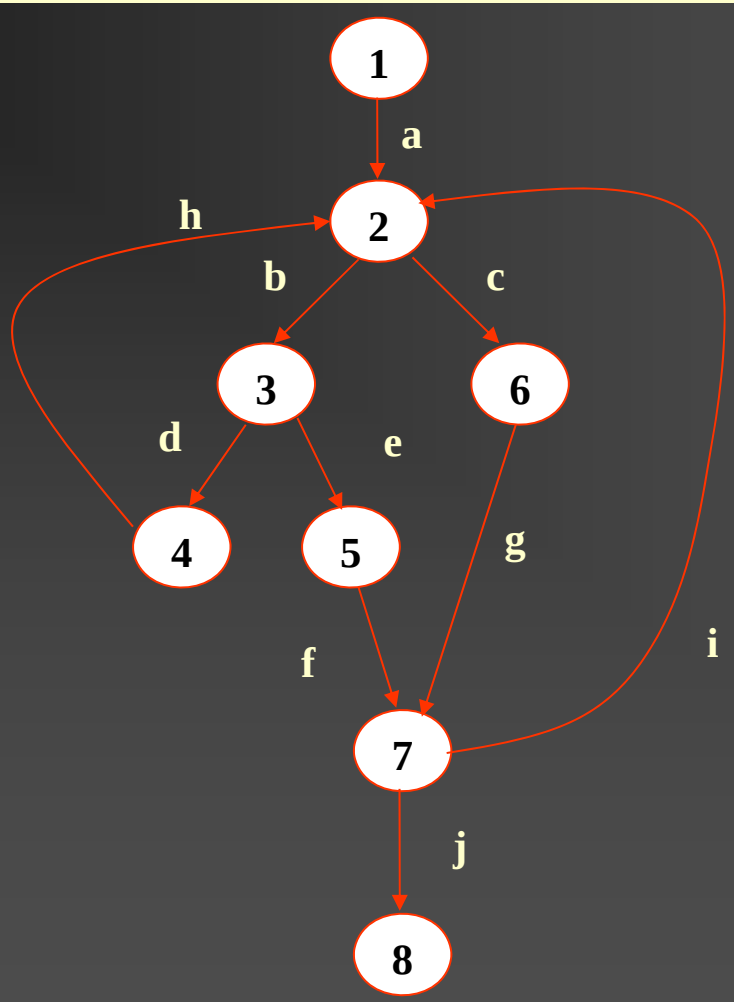
这里所用的程序段非常简短，只有4条路径。但在实际问题中，一个不太复杂的程序，其路径数都是一个庞大的数字，要在测试中覆盖这样多的路径是无法实现的。为解决这一难题只得把覆盖的路径数压缩到一定限度内，例如，程序中的循环体只执行了一次。

即使对于路径数很有限的程序已经作到了路径覆盖，仍然不能保证被测程序的正确性，还必须附以其它测试方法。

带有循环的路径覆盖

- 路径数计算
- 基本路径测试
  - ✓ 不进入循环
  - ✓ 只进入一次循环

例子



例子（续）

路径表达式	状态序列
abefj	1-2-3-5-7-8
acgj	1-2-6-7-8

例子（续）

- 只进入一次循环

abdhbefj	1-2-3-4-2-3-5-7-8
abdhcgj	1-2-3-4-2-6-7-8
acgibefj	1-2-6-7-2-3-5-7-8
acgicgj	1-2-6-7-2-6-7-8
abdhbefibefj	1-2-3-4-2-3-5-7-2-3-5-7-8
abdhbeficgj	1-2-3-4-2-3-5-7-2-6-7-8
abdhcgibefj	1-2-3-4-2-6-7-2-3-5-7-8
abdhcgicgj	1-2-3-4-2-6-7-2-6-7-8
acgibdhbefj	1-2-6-7-2-3-4-2-3-5-7-8
acgibdhcgj	1-2-6-7-2-3-4-2-6-7-8

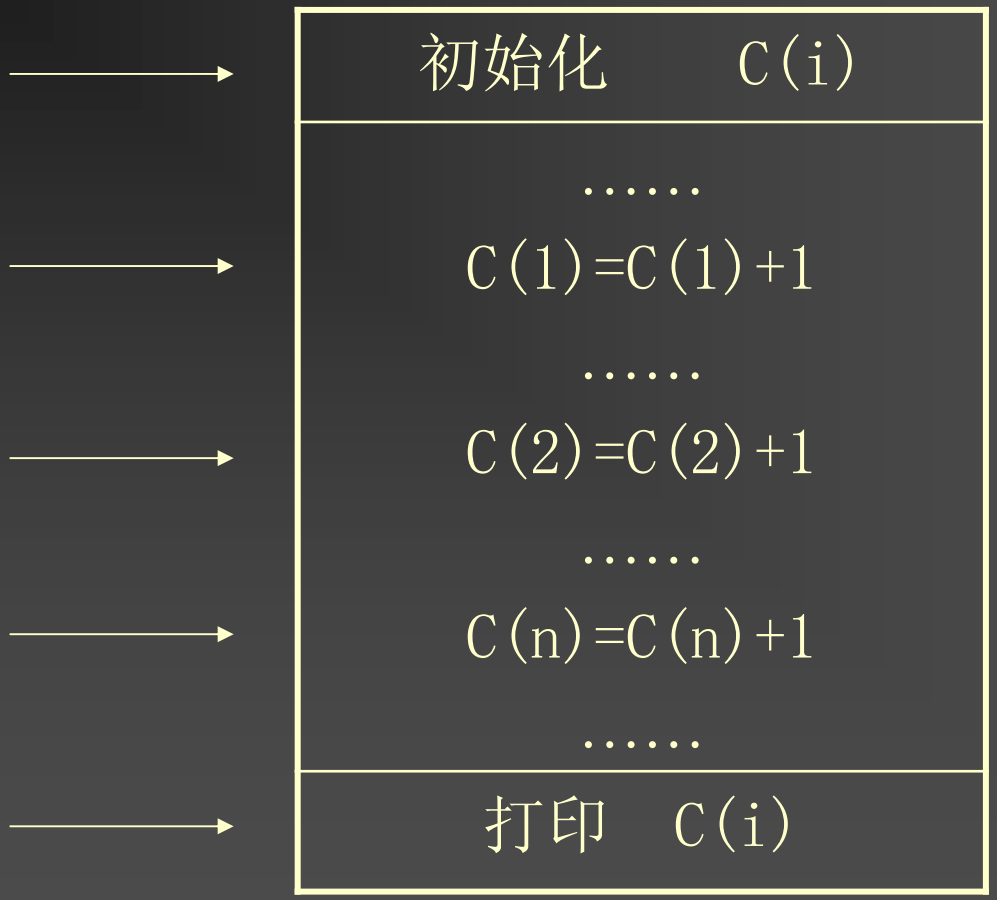
2.2.7 练习

- 导出测试用例步骤
  - ✓ 从代码导出流图
  - ✓ 确定流图的圈复杂度
  - ✓ 确定分支或路径的基本集
  - ✓ 导出测试用例
  - ✓ 与预期结果进行比较。

2.3 插装

- 插入打印/log语句打印关心的信息
- 插入记数语句看覆盖性
- 插入断言看正确性

实施



插装程序中插入的语句

3 自动测试用例设计

- 提高工作效率
- 增强测试的准确性
- 消除人为的错误
- 提高测试的可重复性

怎样更快地发现bug

- 边界
- 代码重用的地方
- 刚修复bug的地方－冒烟测试
- 设计线索
- 负测试－negative test
- 用户易用性考虑
- 接口
- 逻辑上相关的地方