中山大學
SUN YAT-SEN UNIVERSITY

Chapter 4
Software Dynamic Testing

Software Testing: Approaches & Technologies
School of Data & Computer Science, Sun Yat-sen University

Outline

- 4.1 白盒测试
- 4.2 黑盒测试
- 4.3 灰盒测试
- 4.4 测试用例设计
- 4.5 单元测试
- 4.6 集成测试
- 4.7 确认测试
- 4.8 系统测试
- 4.9 动态测试工具

Outline

- 4.2 黑盒测试
  - 概述
  - 等价类划分
  - 边界值分析
  - 错误推测法
  - 随机测试
  - 因果图方法

4.2 黑盒测试

4.2.1 黑盒测试概述

– 黑盒测试的概念

✧ 黑盒测试把测试对象当作看不见内部细节的“黑盒”，在完全不考虑被测程序内部结构和处理过程的情况下，仅仅依据程序功能的需求规格设计测试用例，并推断测试结果的正确性。

◌ 黑盒测试也称为功能测试或数据驱动测试。黑盒测试要求导出执行被测程序所有功能需求的输入条件集，实现功能覆盖。

- 功能覆盖主要是需求覆盖，通过设计一定的测试用例，对每个需求点进行测试。
- 根据软件产品需求规格说明中的功能设计规格进行测试，以证实每个实现了的功能是否符合规格要求。

✧ 比较：白盒测试按照程序内部逻辑结构和编码结构来设计测试数据并完成测试，又称为结构测试或逻辑驱动测试。

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试的概念 (续)
  - 黑盒测试在被测软件的接口处进行。
    - 黑盒测试着眼于被测程序的外部结构，而不考虑其内部逻辑，测试主要针对软件界面和软件功能进行。
    - 黑盒测试站在软件使用者的角度，从输入数据与输出数据的对应关系出发进行测试。
  - 黑盒测试需要解决的问题：
    - 如何测试功能的有效性？
    - 如何测试系统的行为和性能？
    - 何种类型的输入会产生好的测试用例？
    - 系统是否对特定的输入值特别敏感？
    - 如何分隔数据类的边界？
    - 系统能够承受的数据速率和数据量？

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试的局限
  - ✧ 黑盒测试无法发现被测程序的外部特性设计问题或规格说明的规定有误。
  - ✧ 黑盒测试不能替代白盒测试，而是用来发现白盒测试以外的其他类型的错误，比如：
    - ◌ 功能错误或遗漏
    - ◌ 接口错误或界面错误
    - ◌ 数据结构或外部数据库访问错误
    - ◌ 性能错误
    - ◌ 初始化和中止错误

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试的局限 (续)
  - 黑盒测试需要在所有可能的输入/输出条件中确定测试数据，检查程序是否都能产生正确输出，以期发现被测程序中的错误。
    - 一般情况下难以实现。穷举测试数量太大，不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。
      - 假设一个程序 P 有输入量 X 和 Y 及输出量 Z。在字长为32位计算机上运行。若 X、Y 取整数，按黑盒方法进行穷举测试，其测试数据组数量可能为 232×232=264。
      - 如果测试一组数据需要1毫秒，一年工作 365×24 小时，完成 P 的所有测试大约需要5亿年。

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试方法
  - ✧ 黑盒测试需要一些产生测试用例的方法，用以产生有限的测试用例而覆盖足够多的“任何情况”。
  - ✧ 常用的黑盒测试方法：
    - ◌ 等价类划分法
    - ◌ 边界值分析法
    - ◌ 猜错法
    - ◌ 随机数法
    - ◌ 因果图方法
    - ◌ 。。。
  - ✧ 这些测试方法从更广泛的角度来进行黑盒测试，每种方法都力图能涵盖更多的“任何情况”，但又各有长处。
  - ✧ 综合使用这些方法，可以得到较好的测试用例。

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试要求
  - ✧ 黑盒测试既要考核“程序是否做了应该做的”，还要考察“程序是否没有做不应该做的”。
    - ◌ 同时还要考察被测程序在其他一些情况下是否正常，这些情况包括数据类型和数据值的异常等等。
  - ✧ 每个被测程序的特性或功能必须被一个测试用例或一个被认可的异常所覆盖。
  - ✧ 黑盒测试需要构造数据类型和数据值的最小集测试。
    - ◌ 用一系列真实的数据类型和数据值运行，测试超负荷、饱和及其他“最坏情况”结果。
  - ✧ 通过假想的数据类型和数据值测试系统排斥不规则输入的能力
  - ✧ 对影响系统性能的关键模块 (如基本算法) 的模块性能 (如精度、时间、容量等) 进行测试。

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试要求 (续)
  - 黑盒测试设计的测试用例集需要满足以下两个标准：
    - 所设计的测试用例能够减少为达到合理测试所需要设计的测试用例的总数；
    - 所设计的测试用例能够告诉我们是否存在某些类型的错误，而不是仅仅指出与特定测试有关的错误是否存在。

4.2 黑盒测试

4.2.1 黑盒测试概述

– 黑盒测试的通过准则

- ✧ 黑盒测试只有测试通过和测试失败两种结果。
- ✧ 在设计和执行测试用例时，先要进行通过测试，即在进行破坏性试验之前，考察被测软件基本功能是否能够实现。
- ✧ 通过测试实际上只是确认被测软件能做什么，而不去考验其能力如何。
- ✧ 测试人员只运用最简单、最直观的测试用例。
- ✧ 在确信被测软件正确运行之后，可采取各种手段 (例如压力测试) 找出软件缺陷。

4.2 黑盒测试

4.2.1 黑盒测试概述

- 黑盒测试的测试用例开发
  - ✧ 黑盒测试与被测程序如何实现无关。即使程序的实现发生变化，黑盒测试用例仍然应该可用 (可重用性，面向回归测试)。
  - ✧ 黑盒测试用例开发可以与软件开发同时进行，从而节省测试用例的开发时间。通过软件用例可以设计出大部分的黑盒测试用例。
  - ✧ 黑盒测试存在的一些问题：
    - ◌ 测试用例数量较大
    - ◌ 测试用例可能产生很多冗余
    - ◌ 功能性测试的覆盖范围难以达到完全覆盖

4.2 黑盒测试

4.2.2 等价类划分法

- 等价类划分是一种典型的黑盒测试方法。
  - ✧ 等价类划分方法完全不考虑被测程序的内部结构，只依据程序的规格说明来设计测试用例。
  - ✧ 等价类划分方法把所有可能的输入数据 (即程序的输入域) 划分成若干部分，然后从每一部分中选取少数有代表性的数据做为测试用例。
  - ✧ 使用等价类划分方法设计测试用例要经历划分等价类 (列出等价类表) 和确立测试用例两个步骤。

4.2 黑盒测试

4.2.2 等价类划分法

- 划分等价类
  - ✧ 一个输入等价类是指程序输入域的某个子集合，在该子集合中，各个输入数据对于揭露程序中的错误是等效的。测试某一个等价类的代表值就等同于对这个等价类的其它值的测试。
  - ✧ 等价类的划分有两种不同的情况：
    - ◌ 有效等价类：对于程序规格说明来说是合理的、有意义的输入数据构成的集合。
    - ◌ 无效等价类：对于程序规格说明来说是不合理的、无意义的输入数据构成的集合。
  - ✧ 设计测试用例时，要同时考虑有效等价类和无效等价类的用例设计。

4.2 黑盒测试

4.2.2 等价类划分法

- 划分等价类的原则
  - ✧ 如果输入条件规定了输入数据的取值范围或者值的个数，则可以确立一个有效等价类和两个无效等价类。
    - ◌ 例：在程序的规格说明中，输入条件规定 “... 项数可以从1到999 ...”，则有效等价类是 “1≤项数≤ 999”，两个无效等价类是 “项数<1”和“项数>999”。
  - ✧ 如果输入条件规定了输入数据的集合，或者是规定了“必须如何”的条件，则可以确立一个有效等价类和一个无效等价类。
    - ◌ 例：在 Pascal 语言中对变量标识符规定为“以字母打头的 ... 串”。那么所有以字母打头的串构成有效等价类，而不在此集合内的串 (不以字母打头的串) 的归于无效等价类。

4.2 黑盒测试

4.2.2 等价类划分法

- 划分等价类的原则 (续)
  - ✧ 如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。
  - ✧ 如果规定了输入数据的一组值，而且程序要对每个输入值分别进行处理，这时可为每一个输入值确立一个有效等价类，此外针对这组值确立一个无效等价类，它是所有不允许的输入值的集合。
    - ◌ 例：初等几何中对等腰三角形、平行四边形、梯形和圆形四类规则平面图形的面积分别采用相应的面积公式进行计算。
      - 可以确定4个有效等价类为等腰三角形、平行四边形、梯形和圆形，而由所有非上述四类图形的其它平面图形的输入值的集合构成一个无效等价类。

4.2 黑盒测试

4.2.2 等价类划分法

- 划分等价类的原则 (续)
  - ✧ 如果规定了输入数据必须遵守的规则，则可确立一个有效等价类 (符合规则) 和若干个无效等价类 (从不同角度违反规则)。
    - ◌ 例：Pascal 语言规定“一个语句必须以分号 ‘;’ 结束”。这时，可以确定一个有效等价类“以 ‘;’ 结束”，若干个无效等价类“以 ‘:’ 结束”、“以 ‘,’ 结束”、“以 ‘ ’ 结束”、“以 LF 结束” 等等。

4.2 黑盒测试

4.2.2 等价类划分法

- 确立测试用例
  - ✧ 确立了等价类划分之后，建立等价类表，列出所有划分得到的等价类，并按以下原则选择测试用例：
    - ◌ 为每一个等价类规定一个唯一编号；
    - ◌ 设计一个新的测试用例，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，直到所有的有效等价类都被覆盖为止；
    - ◌ 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

4.2 黑盒测试

4.2.2 等价类划分法

– 确立测试用例 (续)

✧ 例：在某 Pascal 语言版本中规定：

(1) 标识符是由字母开头，后跟字母或数字的任意字符组合构成；构成标识符的有效字符数为 1 - 8 个；标志符的字符总数不能超过80个；

(2) 标识符必须先说明，再使用；

(3) 在同一说明语句中，至少必须有一个标识符。

◌ 被测程序用于检查上述合规性。用等价类划分法设计该程序的测试用例。

4.2 黑盒测试

4.2.2 等价类划分法

– 确立测试用例 (续)

✧ 例：(续)

◌ 划分得到的15个等价类及编号如下：

输入条件	有效等价类	无效等价类
标识符个数	1个（1），多个（2）	0个（3）
标识符字符数	1～8个（4）	0个（5），>8个（6），>80个（7）
标识符组成	字母（8），数字（9）	非字母数字字符（10），保留字（11）
第一个字符	字母（12）	非字母（13）
标识符使用	先说明后使用（14）	未说明已使用（15）

4.2 黑盒测试

- 下面选取了9个测试用例，它们覆盖了所有的15个等价类。

① VAR x, T1234567: REAL; BEGIN x := 3.414; T1234567 := 2.732;	(1), (2), (4), (8), (9), (12), (14)
② VAR : REAL;	(3)
③ VAR x, : REAL;	(5)
④ VAR T12345678: REAL;	(6)
⑤ VAR T12345 ..... : REAL; 多于80个字符	(7)
⑥ VAR T$: CHAR;	(10)
⑦ VAR GOTO: INTEGER;	(11)
⑧ VAR 2T: REAL;	(13)
⑨ VAR PAR: REAL; BEGIN ..... PAP := SIN (3.14 * 0.8) / 6;	(15)

4.2 黑盒测试

- 下面选取了9个测试用例，它们覆盖了所有的15个等价类。

① VAR x, T1234567: REAL; BEGIN x := 3.414; T1234567 := 2.732; ……	(1), (2), (4), (8), (9), (12), (14)
② VAR : REAL;	(3)
③ VAR x, : REAL;	(5)
④ VAR T12345678: REAL;	(6)
⑤ VAR T12345 …… : REAL; 多于80个字符	(7)
⑥ VAR T$: CHAR;	(10)
⑦ VAR GOTO: INTEGER;	(11)
⑧ VAR 2T: REAL;	(13)
⑨ VAR PAR: REAL; BEGIN …… PAP := SIN (3.14 * 0.8) / 6;	(15)

用例对应的等价类编号

4.2 黑盒测试

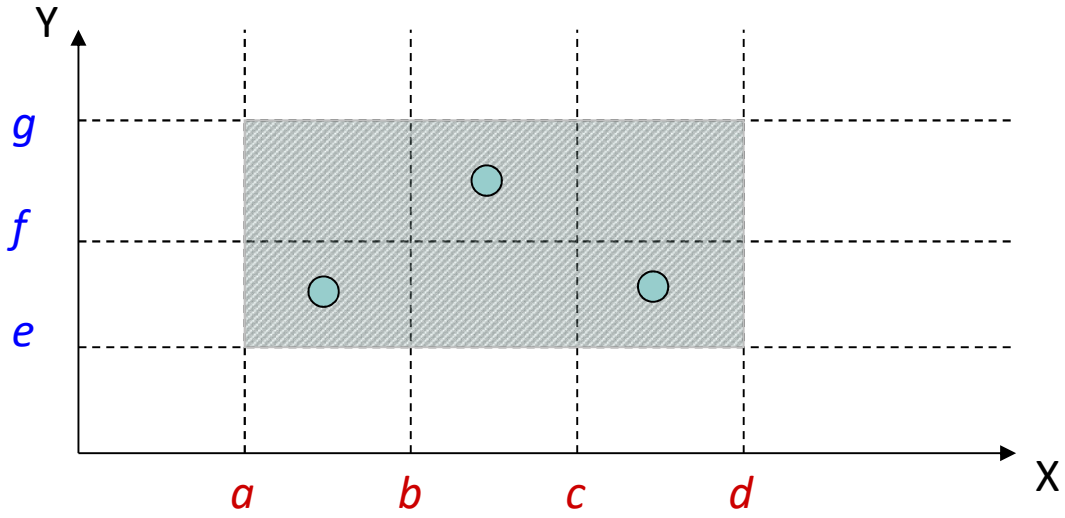
4.2.2 等价类划分法

- 弱/强、一般/健壮的等价类测试分类
  - ✧ 弱等价类测试
    - ◌ 针对单缺陷的等价类测试用例设计
  - ✧ 强等价类测试
    - ◌ 针对多缺陷的等价类测试用例设计
  - ✧ 一般等价类测试
    - ◌ 只覆盖有效等价类的测试用例设计
  - ✧ 健壮等价类测试
    - ◌ 同时覆盖有效等价类和无效等价类的测试用例设计
    - ◌ 健壮性：在异常情况下软件还能正常运行的能力。健壮性包括容错能力和异常恢复能力。容错性测试通常构造一些不合理的输入来诱导软件出错。

4.2 黑盒测试

4.2.2 等价类划分法

- 弱/强、一般/健壮的等价类测试分类 (续)
  - ✧ 弱一般等价类测试用例覆盖
    - ◌ 针对单缺陷，只覆盖有效等价类



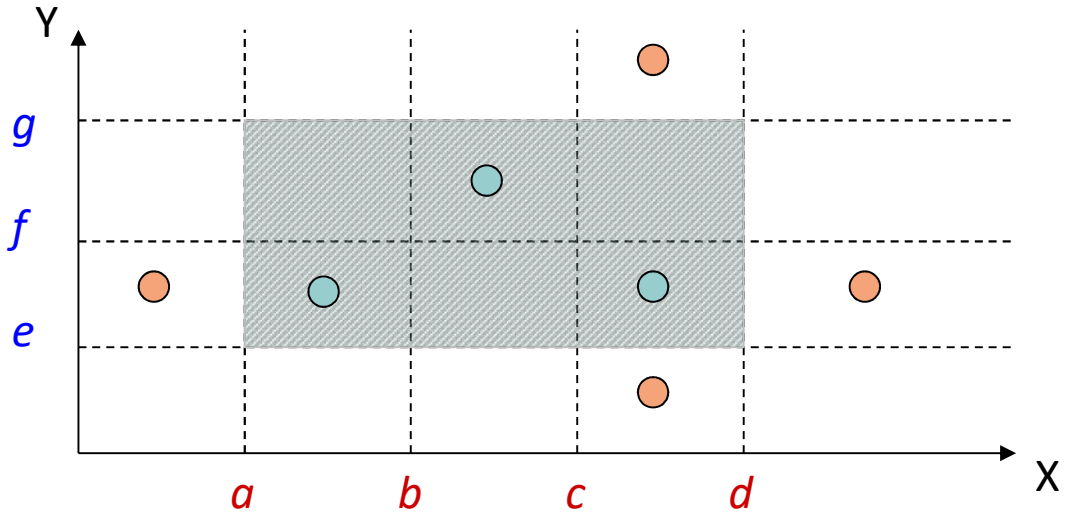
X 有效等价类 [*a*, *b*), [*b*, *c*), [*c*, *d*]，无效等价类 X<*a*, X>*d*

Y 有效等价类 [*e*, *f*), [*f*, *g*]，无效等价类 Y<*e*, Y>*g*

4.2 黑盒测试

4.2.2 等价类划分法

- 弱/强、一般/健壮的等价类测试分类 (续)
  - ✧ 弱健壮等价类测试用例覆盖
    - ◌ 针对单缺陷，覆盖有效等价类和无效等价类



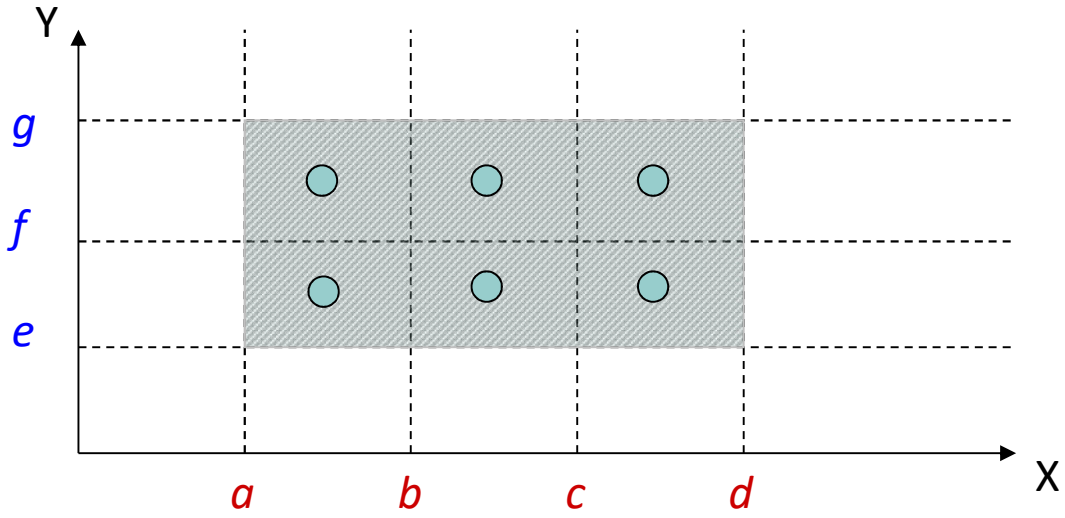
X 有效等价类 [*a*, *b*), [*b*, *c*), [*c*, *d*]，无效等价类 X<*a*, X>*d*

Y 有效等价类 [*e*, *f*), [*f*, *g*]，无效等价类 Y<*e*, Y>*g*

4.2 黑盒测试

4.2.2 等价类划分法

- 弱/强、一般/健壮的等价类测试分类 (续)
  - ✧ 强一般等价类测试用例覆盖
    - ◌ 针对多缺陷，只覆盖有效等价类



X 有效等价类 [*a*, *b*), [*b*, *c*), [*c*, *d*]，无效等价类 X<*a*, X>*d*

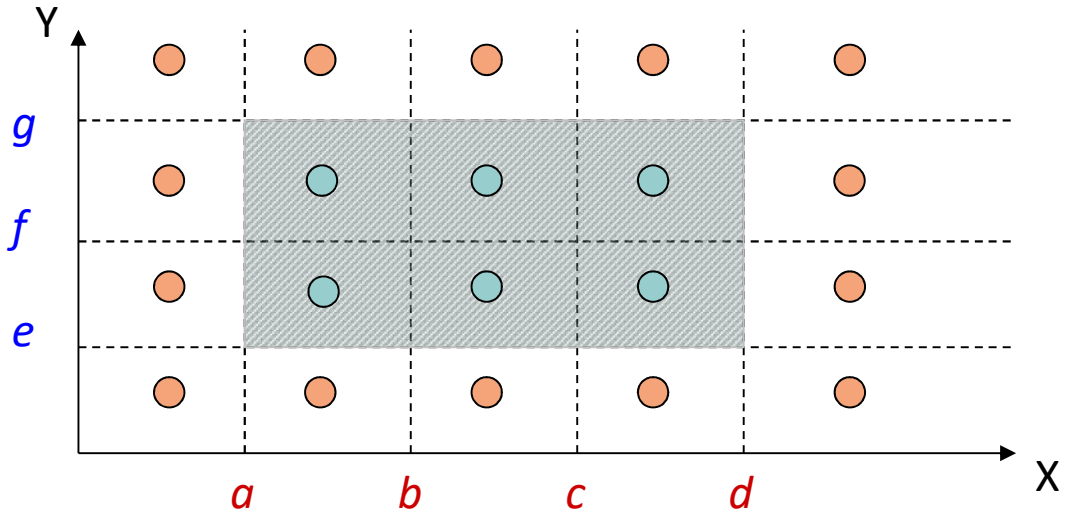
Y 有效等价类 [*e*, *f*), [*f*, *g*]，无效等价类 Y<*e*, Y>*g*

4.2.2 等价类划分法

– 弱/强、一般/健壮的等价类测试分类 (续)

✧ 强健壮等价类测试用例覆盖

◌ 针对多缺陷，覆盖有效等价类和无效等价类



X 有效等价类 [*a*, *b*), [*b*, *c*), [*c*, *d*]，无效等价类 X<*a*, X>*d*

Y 有效等价类 [*e*, *f*), [*f*, *g*]，无效等价类 Y<*e*, Y>*g*

4.2 黑盒测试

4.2.3 边界值分析法

- 概述
  - ✧ 边界值分析法是对等价类划分方法的补充。
  - ✧ 经验显示，大量的错误发生在输入或输出范围的边界上，而不是在输入范围的内部。测试用例针对各种边界情况进行设计，可以查出更多的错误。
  - ✧ 例：在做三角形计算时，需要输入三角形的三个边长：A、B 和 C。 我们应注意到这三个数值应当满足

    $A>0$、$B>0$、$C>0$、$A+B>C$、$A+C>B$、$B+C>A$

    才能构成三角形。
    - ◌ 如果把六个不等式中的任何一个大于号“>”错写成大于等于号“>=”，就不能构成三角形。
    - ◌ 问题恰出现在容易被疏忽的边界附近。

4.2 黑盒测试

4.2.3 边界值分析法

- 概述 (续)
  - ✧ 用边界值分析方法设计测试用例，先要确定边界情况。应选取正好等于、刚刚大于、或刚刚小于边界的值做为测试数据，而不是选取等价类中的典型值或任意值做为测试数据。
  - ✧ 边界值的分析与确定比较复杂，要求测试人员有更多的经验和耐心。
- 等价类划分法与边界值分析法的比较
  - ✧ 等价类划分法的测试数据在各个等价类允许的值域内任意选取，而边界值分析法的测试数据必须在边界值附近选取。
  - ✧ 一般而言，用边界值分析法设计的测试用例要比等价类划分法更具有代表性，发现错误的能力也更强。

4.2 黑盒测试

4.2.4 错误推测法

- 概述
  - 错误推测法依靠测试人员的经验和直觉对被测程序中可能存在的各种错误进行推测，从而有针对性地编写测试用例。
- 基本思想
  - 列举程序中所有可能有的错误和容易发生错误的特殊情况，作为选择测试用例的依据。例如：
    - 输入数据和输出数据为0；
    - 输入表格为空格或输入表格只有一行。
  - 例：测试用于线性表排序的程序，推测需要测试的情况：
    - 输入的线性表为空表或表中只含有一个元素；
    - 输入表中所有元素已排好序；
    - 输入表已按逆序排好；
    - 输入表中部分或全部元素相同。

4.2 黑盒测试

4.2.5 随机测试法

- 概述
  - 随机测试指测试输入数据是从被测程序的所有可能输入值中随机选取的，是一种基本的黑盒测试方法。
  - 数据的随机选取可以采用随机模拟的方法，包括采用伪随机数发生器、硬件随机模拟器等方法产生输入数据。
  - 随机测试方法能够获得大量的测试数据。测试人员只需要规定输入变量的取值区间，提供必要的变换机制，使产生的随机数服从预期的概率分布。
- 随机测试法的局限性
  - 随机测试方法不能事先将测试的输入数据存入文档，在排错时难以重现测试中错误发生的过程，不便进行回归测试。
    - 补救的办法是将随机产生的测试数据记录备用。

4.2 黑盒测试

4.2.6 因果图法

- 概述
  - ✧ 因果图方法考虑被测程序输入条件之间的联系和组合关系，描述由多种条件的组合产生多个动作的结构形式，方便设计测试用例。
    - ◌ 因果图方法检查程序输入条件的各种组合情况，最终生成判定表，帮助我们按一定步骤，高效率地选择测试用例。
  - ✧ 因果图方法有助于指出程序规格说明描述中存在的问题。

4.2 黑盒测试

4.2.6 因果图法

- 因果图法生成测试用例的基本步骤
  - ✧ 对软件规格说明进行描述
    - ◌ 给每个原因 (即输入条件或输入条件的等价类) 和结果 (即输出条件) 赋予一个标识符。
  - ✧ 分析软件规格说明描述中的语义
    - ◌ 找出原因与结果之间、或者原因与原因之间的对应关系；
    - ◌ 根据这些关系，画出因果图。
  - ✧ 分析有关的语法或环境限制条件
    - ◌ 由于语法或环境限制，有些原因和结果因素的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。
  - ✧ 把因果图转换成判定表
  - ✧ 判定表的每一列就是测试用例设计的依据

4.2 黑盒测试

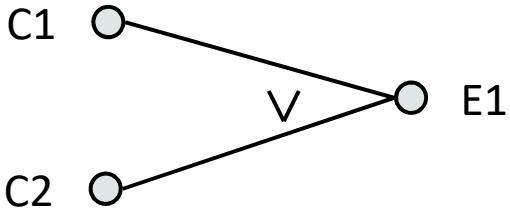
4.2.6 因果图法

– 因果图中使用的基本符号

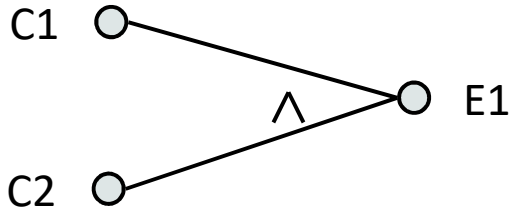
- ✧ 用 Ci 表示原因
- ✧ 用 Ei 表示结果
- ✧ 用结点表示状态，可取值 “0” 或 “1”
  - ◌ “0” 表示某状态不出现，“1” 表示某状态出现。
- ✧ 原因和结果之间的关系图解主要有：

恒等：C1 ↔ E1

非：C1 ↔ ¬ E1



或：C1∨C2 → E1



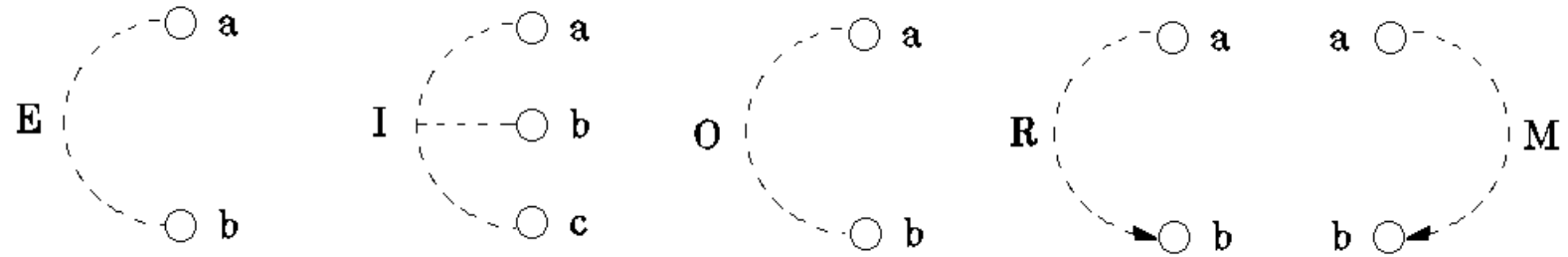
与：C1∧C2 → E1

4.2 黑盒测试

4.2.6 因果图法

– 因果图中使用的表示约束条件的符号

✧ 为表示原因与原因之间，结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号：



(1) E（互斥·排他） (2) I（包含·或） (3) O（唯一） (4) R（要求） (5) M(屏蔽)

✧ 互斥 E：a, b 两个原因不会同时成立，最多有一个可能成立；

✧ 包含 I：a, b, c 三个原因中至少一个必须成立；

✧ 唯一 O：a 和 b 当中必须有一个，且仅有一个成立；

✧ 要求 R：当 a 出现时，b 必须也出现；

✧ 屏蔽 M：当 a 为1时，b 必须是0；当 a 为0时，b 值不定。

4.2 黑盒测试

4.2.6 因果图法

- 例：设计一个处理单价为5角钱的饮料的自动售货机软件的测试用例。软件规格说明如下：
  - ✧ 操作者投入5角钱或1元钱的硬币，按下〖橙汁〗或〖啤酒〗的按钮，则送出相应的饮料 (不考虑饮料不足的情况)。
  - ✧ 若售货机没有零钱找，则一个显示〖零钱找完〗的红灯亮。
    - ◌ 此时操作者投入1元硬币并按下按钮后，不送出饮料，退还1元硬币。
  - ✧ 若售货机有零钱找，则显示〖零钱找完〗的红灯灭。
    - ◌ 此时操作者投入1元硬币并按下按钮后，送出饮料，退还5角硬币。

4.2 黑盒测试

4.2.6 因果图法

– 例：(续)

1. 分析需求说明，列出原因和结果清单

- 原因清单 (输入条件)
  - C1 售货机可找零
  - C2 投入1元硬币
  - C3 投入5角硬币
  - C4 按下〖橙汁〗橙汁按钮
  - C5 按下〖啤酒〗按钮
- 结果清单 (输出结果)
  - E21 〖零钱找完〗灯亮
  - E22 退还1元硬币
  - E23 退还5角硬币
  - E24 送出橙汁饮料
  - E25 送出啤酒饮料

4.2 黑盒测试

4.2.6 因果图法

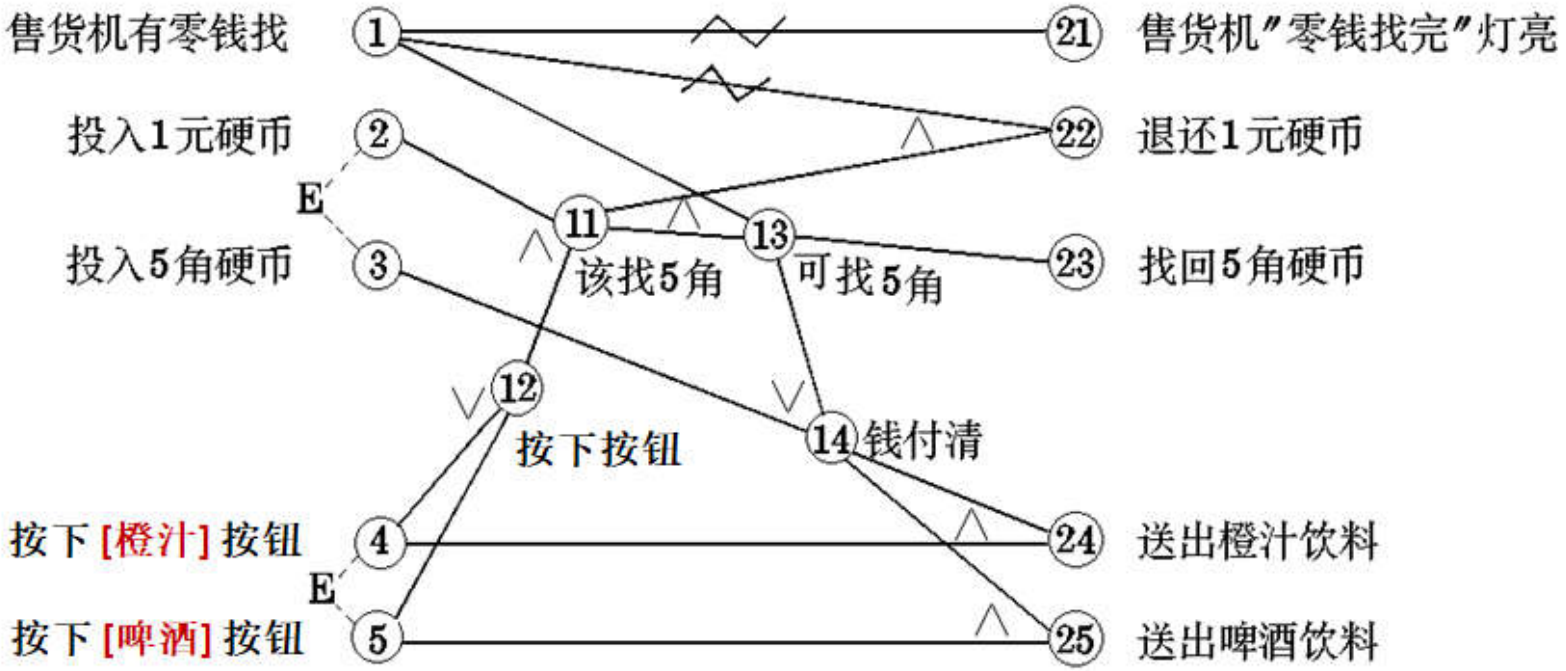
- 例：(续)

1. 分析需求说明，列出原因和结果清单 (续)
   - 建立中间结点，表示处理中间状态
     - T11 投入1元硬币且按下饮料按钮
     - T12 按下〖橙汁〗或〖啤酒〗按钮
     - T13 应当找5角零钱并且售货机有零钱找
     - T14 钱已付清
2. 画出因果图
   - 所有原因结点列在左边
   - 所有结果结点列在右边
   - 所有中间结点列在中间
   - 所有因果关系表示为连接图解
   - 加上必要的互斥约束条件 E
     - C2 与 C3、C4 与 C5 不能同时发生。

4.2 黑盒测试

4.2.6 因果图法

– 例：(续)



3. 因果图转换成判定表

- 按照上面的因果图建立规则库，对输入条件 C1-C5 的全部解释计算输出结果，得到 2^5=32 列的判定表。

4.2 黑盒测试

4.2.6 因果图法

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

- 判定表图例

4.2 黑盒测试

4.2.6 因果图法

3. 因果图转换成判定表 (续)

- 判定表中
  - 阴影部分表示违反约束条件的不可能出现的情况，可以删去对应列。
  - 第16列与第32列对应的输入条件 C2、C3、C4、C5 为0 (黄色部分)，表示操作者没有动作，可以删去对应列。
  - 最后根据剩下的16列作为确定测试用例的依据。

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

4.2 黑盒测试

4.2.6 因果图法

3. 因果图转换成判定表 (续)

- 判定表中
  - 绿色标示的16列作为确定测试用例的依据。

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

4.2 黑盒测试

4.2.6 因果图法

4. 判定表的分析

- 以判定表的第7列为例：
  - 输入11001，表示 C1 售货机可找零、C2 投入1元硬币、C5 按下啤酒按钮。
  - 输出00101，表示 E23 退还5角硬币、E25 送出啤酒饮料
  - 实现上述输入-输出过程的规则描述：

$C1 \wedge T11 \rightarrow T13$

$C2 \wedge T12 \rightarrow T11$

$C4 \vee C5 \rightarrow T12$

$C3 \vee T13 \rightarrow T14$

$C5 \wedge T14 \rightarrow E25$

$T13 \rightarrow E23$

  - 以 C1, C2, C5 为前提，应用上述规则，可以证明逻辑结论 E23 和 E25。

序号		1	2	3	4	5	6	7	8
条件	①	1	1	1	1	1	1	1	1
	②	1	1	1	1	1	1	1	1
	③	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0
	⑫						1	1	0
	⑬						1	1	0
	⑭						1	1	0
结果	㉑						0	0	0
	㉒						0	0	0
	㉓						1	1	0
	㉔						1	0	0
	㉕						0	1	0

4.2 黑盒测试

4.2.6 因果图法

- 小结
  - ✧ 因果图方法是一种非常有效的黑盒测试方法。
    - ◌ 能够生成没有重复性且发现错误能力强的测试用例；
    - ◌ 对输入、输出同时进行了分析。
  - ✧ 从因果图生成的测试用例包括了被测程序所有输入数据的“取真”与“取 假”的情况。
    - ◌ 构成相对较少的测试用例数目；
    - ◌ 测试用例数目随被测程序输入数据数目的增加而非线性地增加。
  - ✧ 在设计阶段就采用了判定表的项目，可以直接利用判定表设计测试用例。

Outline

- 4.1 白盒测试
- 4.2 黑盒测试
- 4.3 灰盒测试
- 4.4 测试用例设计
- 4.5 单元测试
- 4.6 集成测试
- 4.7 确认测试
- 4.8 系统测试
- 4.9 动态测试工具

4.3 灰盒测试

- **灰盒测试概述**
  - 灰盒测试方法提出的基础
    - 白盒测试过程中测试者可以看到被测的源程序，通过分析程序的内部结构，根据其内部结构设计测试用例。
      - 理想的白盒测试应该使选取的测试用例覆盖被测程序所有的路径，但这难以做到。
      - 白盒测试不关注被测程序的外部功能。
    - 黑盒测试是在完全不考虑被测程序内部结构和内部特征的情况下，测试者根据被测程序需求规格说明书设计测试用例，推断测试结果的正确性。
      - 黑盒测试用例的选择只考虑被测程序的输入以及相应的的输出，并没有考虑程序的内部结构。
      - 被测程序内部结构是否规范、结构化程度的好坏、系统的性能如何等都得不到测试。

4.3 灰盒测试

- **灰盒测试概述**
  - 灰盒测试方法提出的基础 (续)
    - ✧ 因此，要进行较全面的测试，可以考虑对被测程序同时采用白盒测试方法和黑盒测试方法。
    - ✧ 灰盒测试是一种综合测试方法，它将黑盒测试、白盒测试、回归测试和变异测试结合在一起，构成一种无缝测试技术。
      - ◌ 灰盒测试是一种软件全生命周期测试法，该方法通常深入到用 Ada/C/Fortran 或汇编语言开发的嵌入式应用软件代码中进行功能测试，或者与 Web 服务应用一起使用。

4.3 灰盒测试

- **灰盒测试概述**
  - 灰盒测试的基本思路
    - ✧ 基于被测程序运行时的外部表现，同时结合被测程序内部逻辑结构来设计测试用例。
    - ✧ 验证被测程序满足外部指标，而且被测程序的所有通道或路径都进行了检验。
    - ✧ 以被测程序的主要功能和主要性能为测试依据。
      - ◌ 根据程序流程图、功能说明书以及测试者的实践经验来设计测试用例。
    - ✧ 现代测试工程中，最常见的灰盒测试是集成测试。
      - ◌ 重点关注被测软件系统的各个模块之间的相互关联，即模块之间的互相调用、数据传递、同步/互斥等等。

4.3 灰盒测试

- **灰盒测试概述**
  - 灰盒测试的特点
    - ✧ 灰盒测试根据被测程序的需求规格说明文档进行测试用例的设计，这点类似于黑盒测试；但它需要深入到被测程序内部的特殊点来进行功能测试和结构测试。
      - ◌ 例如进行单元接口测试：通过将接口参数传递到被测单元中，检验软件在测试执行环境控制下的执行情况。
    - ✧ 灰盒测试通常在程序员做完白盒测试之后，功能测试人员进行大规模集成测试之前进行，并由专门测试人员实施。
    - ✧ 灰盒测试通过类似白盒测试的方法进行。
      - ◌ 需要了解代码工程的实现；
      - ◌ 通过编写代码，调用函数或者封装好的接口进行，但无需关心被测程序模块内部的实现细节，依然可把它当成一个黑盒。

4.3 灰盒测试

- **灰盒测试的特性**
  - 灰盒测试的优点
    - ✧ 能够进行基于需求的覆盖测试和基于程序路径覆盖的测试
    - ✧ 测试结果对应被测程序内部路径，便于定位、分析和解决缺陷
    - ✧ 能够保证所设计的黑盒测试用例的完整性，防止遗漏软件的一些不常用的功能或功能组合
    - ✧ 能够减少需求或设计不详细或不完整对测试造成的影响
  - 灰盒测试的不足
    - ✧ 投入的时间比单纯的黑盒测试多大约 20-40%
    - ✧ 对测试人员的要求较高
      - ◌ 灰盒测试要求测试人员清楚系统内部的模块构成，以及模块之间的协作关系。
    - ✧ 不如白盒测试深入
    - ✧ 不适用于简单的系统

Thank you!