Chapter 18

- **Testing Conventional Applications**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

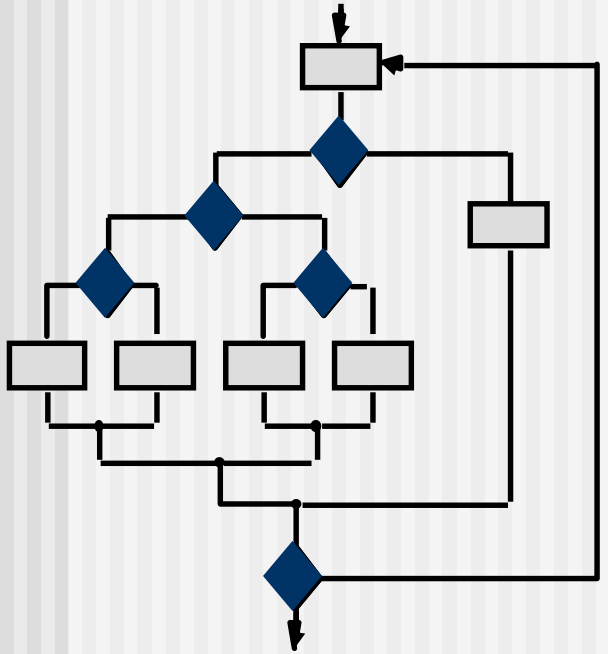
by Roger S. Pressman

关于测试的2个问题

（1）经过测试没有发现错误的软件就一定正确吗？

错误。测试只能证明软件是有错的,但不能证明软件是没有错误。

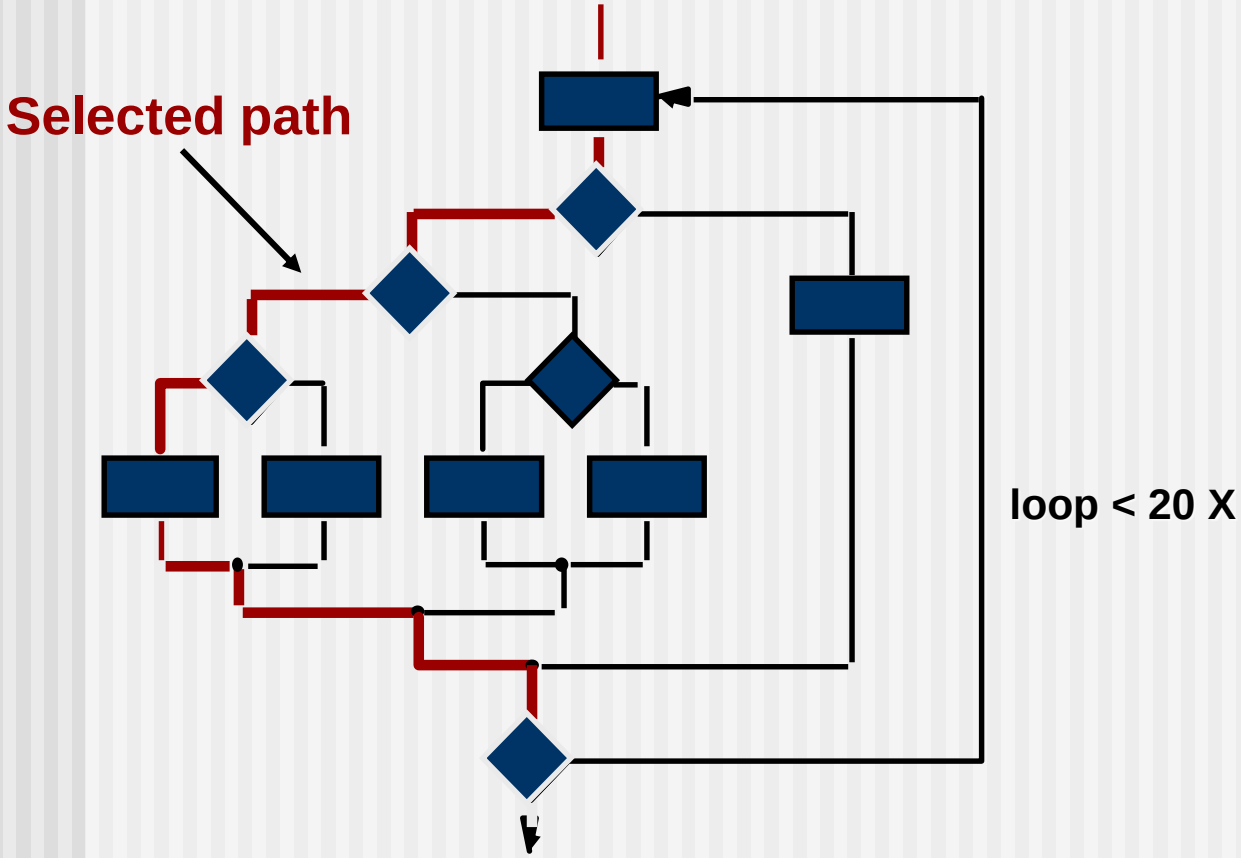
（2）可以通过穷举方式测试软件吗？



There are 10^{14} possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

loop < 20 X

Selective Testing



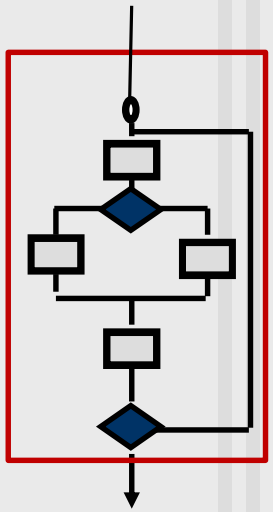
如何设计好的测试用例？

What is a “Good” Test

- 测试的有效性很大程度取决于选取的测试用例
  - A good A good test has a high probability of finding an error
  - test is not redundant.
  - A good test should be “best of breed”

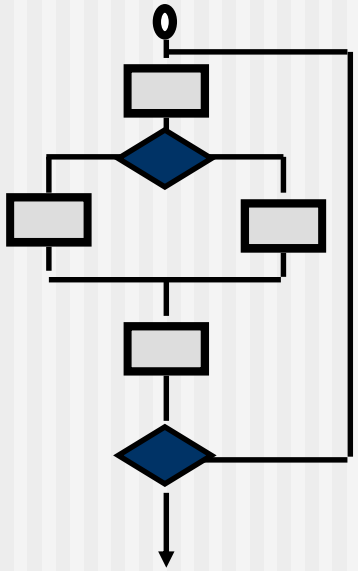
    一组相似测试用例中是最有效的

测试用例设计方法



- Any engineered product can be tested in one of two ways:
  - Knowing the specified function that a product has been designed
  - 对应黑盒法：测试者把测试对象看作一黑盒子，仅依据程序的功能需求，设计测试用例并推断测试结果。
  - Knowing the internal workings of a product internal operations are performed according to specifications
  - 对应白盒法：按照程序内部逻辑结构，设计测试数据并完成测试的一种测试方法。

White-Box Testing

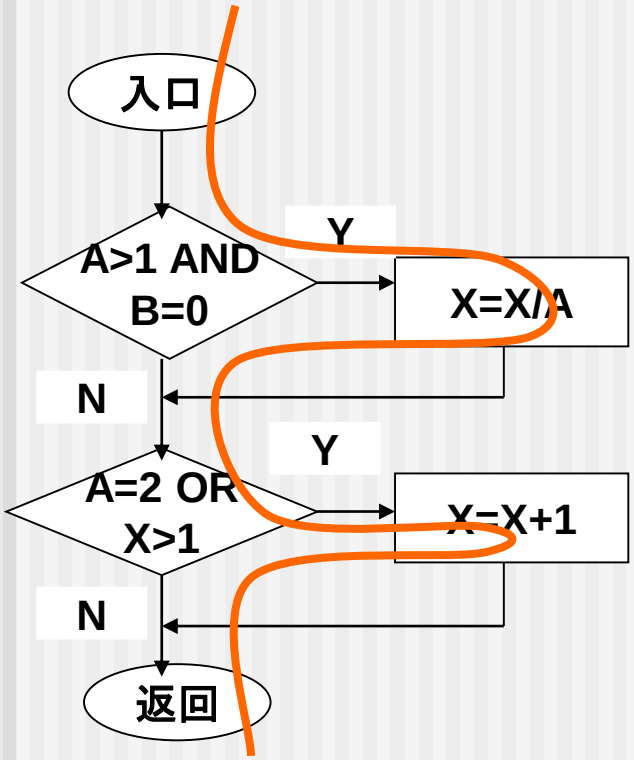


... our goal is to ensure that all statements and conditions have been executed at least once ...

- 该方法把测试对象看做一个透明的盒子，它允许测试人员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。
- 目标是使得程序中各元素（如语句、判定）尽可能被覆盖到。
- 白盒测试一般在测试过程的早期执行。

语句覆盖法

- 语句覆盖：指选择足够的测试用例，使得被测程序中每条语句至少执行一次。
- 是一种比较弱的测试标准。



A	B	X	期望
2	0	4	X=3（红线）

Basis Path Testing

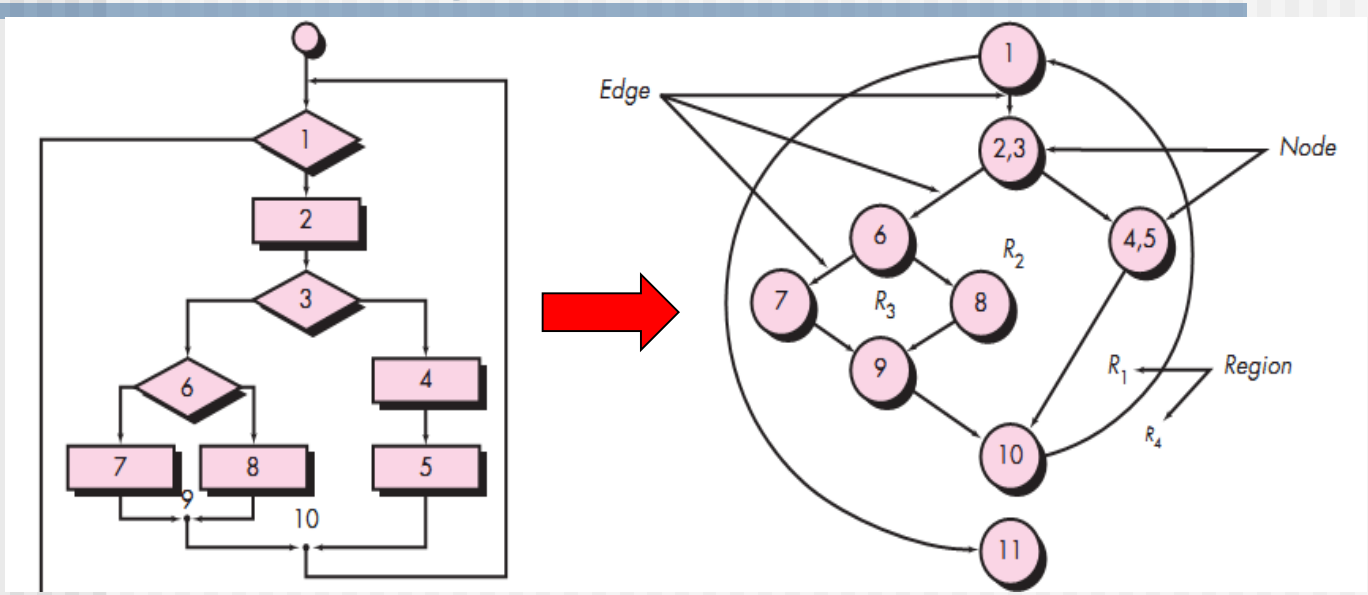
如果程序中含有循环结构，完全的路径覆盖是不可能的。

基本路径法：在程序控制流图的基础上，通过导出基本可执行路径（注: 不是所有可能路径）集合，从而设计测试用例。

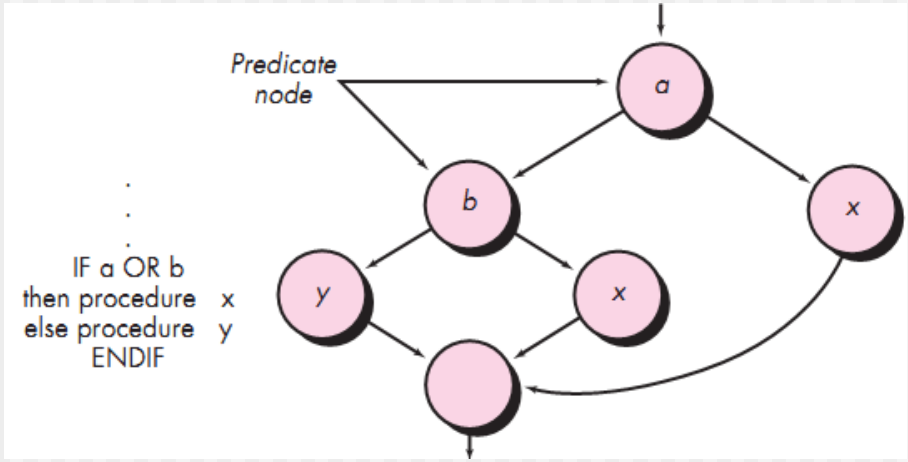
步骤如下：

- Using the design or code as a foundation, draw a corresponding flow graph.
- Determine the cyclomatic complexity of the resultant flow graph.
- Determine a basis set of independent paths.
- Prepare test cases that will force execution of each path in the basis set.

Flow Gragh



复合条件：



Basis Path Testing

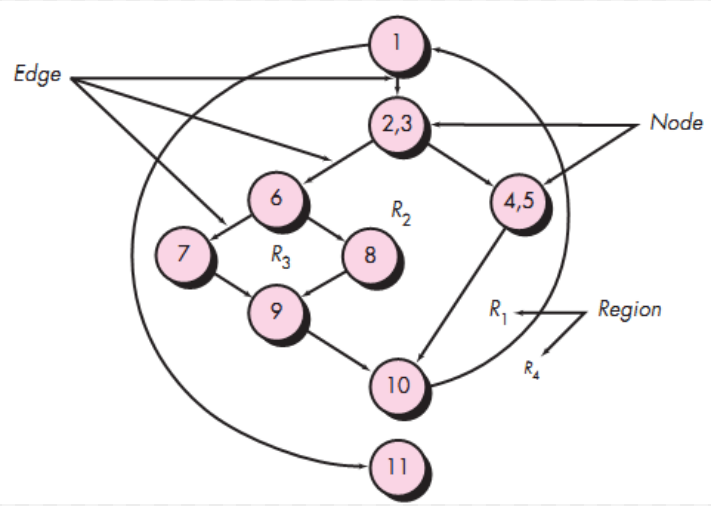
First, we compute the cyclomatic complexity：

number of simple decisions + 1

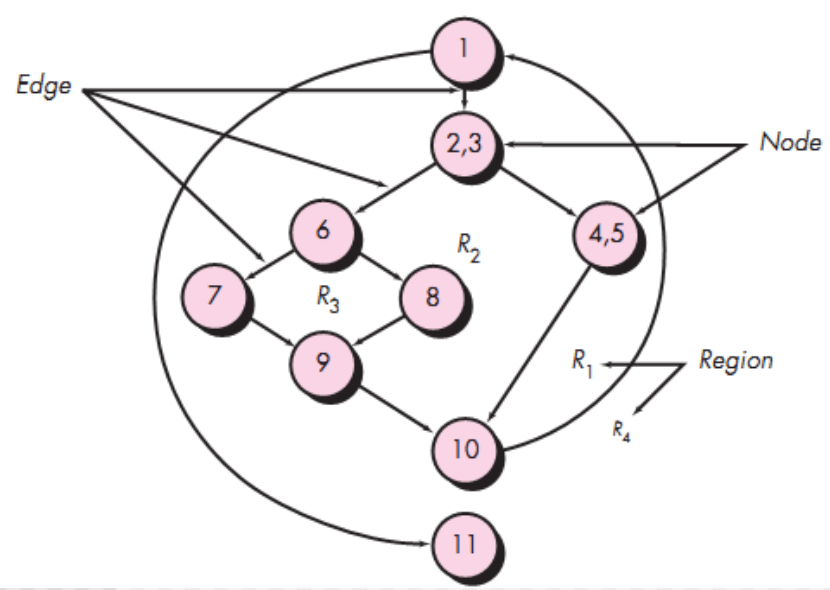
or

number of enclosed areas + 1

In this case, V(G) = 4



Independent Path



Next, we derive the independent paths:

Since V(G) = 4, there are four paths

Path 1: 1-11
Path 2: 1-2-3-4-5-10-1-11
Path 3: 1-2-3-6-8-9-10-1-11
Path 4: 1-2-3-6-7-9-10-1-11

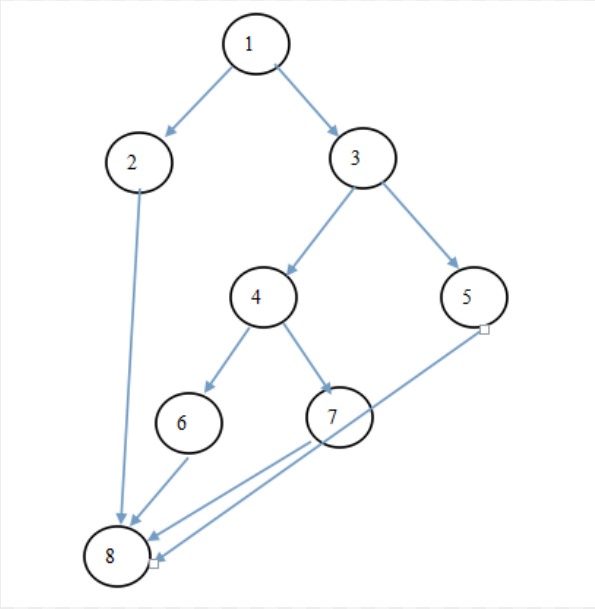
独立路径：一条路径，至少包含一条定义在该路径之前不曾用到的边（对应程序中至少一条新语句或条件）

Finally, we derive test cases to exercise these paths.

Exercise

判断任意年份是否为闰年，需要满足以下条件中的任意一个：
① 该年份能被 4 整除同时不能被 100 整除；
② 该年份能被400整除。

```
Int IsLeap(int year)
{
if (year % 4 == 0)                1
 {
  if (year % 100 == 0)            3
    {
      if ( year % 400 == 0)       4
        leap = 1;                 6
      else
        leap = 0;                 7
    }
  else
        leap = 1;                 5
 }
 else
    leap = 0;                     2
 return leap;                     8
}
```

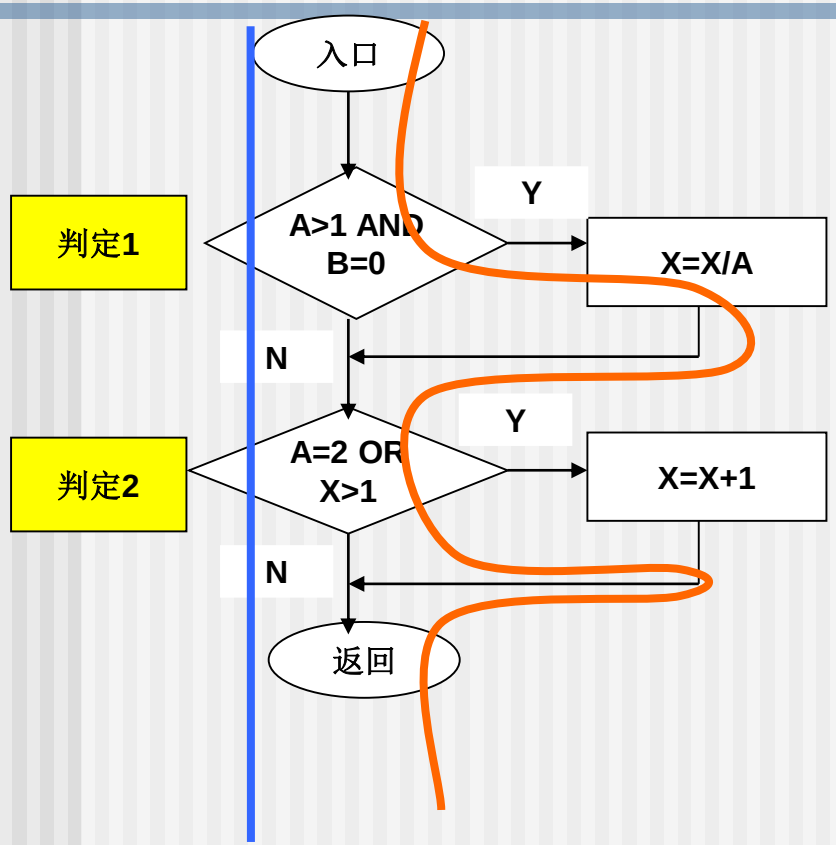


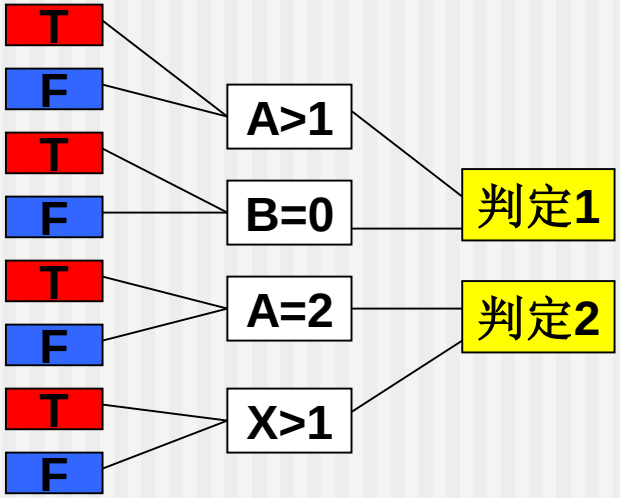
year	覆盖路径	期望	说明
1999	128	**0**	不能被4整除
2000	13468	**1**	能被4，100，400整除
1900	13478	**0**	能被4，100整除，不能被400整除
2004	1358	**1**	能被4整除，不能被100整除

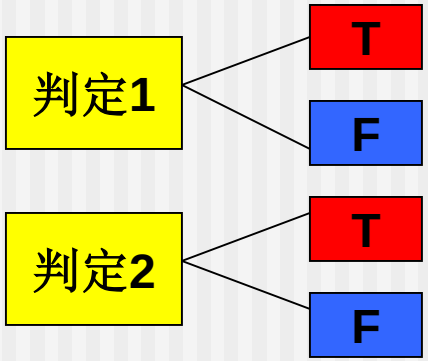
Control Structure Testing

- Condition testing — a test case design method that exercises the logical conditions contained in a program module
  - 设计测试用例确保至少条件C为真为假执行各一次。
  - 如果C是复合条件，则同时还要满足每个简单条件至少执行一次。
- Data flow testing — selects test paths of a program according to the locations of definitions and uses of variables in the program
  - 设计测试用例使得每个变量定义-使用链至少覆盖一次

Condition Testing





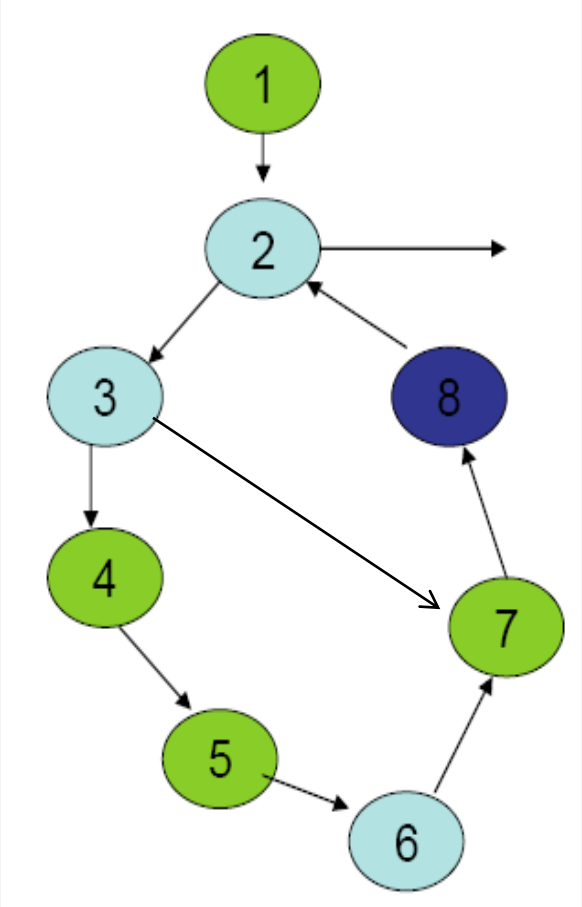


A	B	X	期望
2	0	4	(X=3)红线
1	1	1	(X=1)蓝线

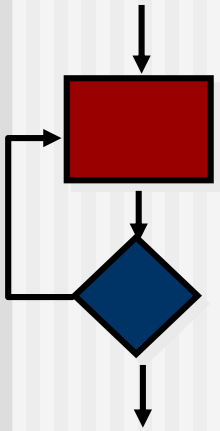
Data Flow Testing

```
1  a=5; // 定义 a
2  While(C1) {
3    if (C2){
4          b=a*a;//使用 a
5          a=a-1;//定义且使用a
6    }
7    print(a); //使用 a
8    Print(b);}//使用 b
```

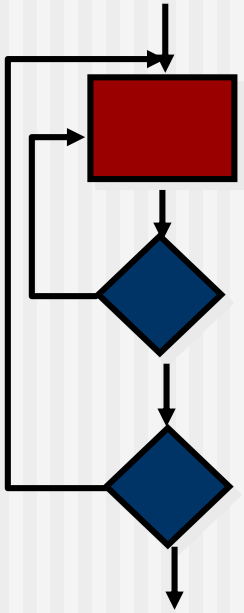
Du-path
1234
12345
1234567
1237



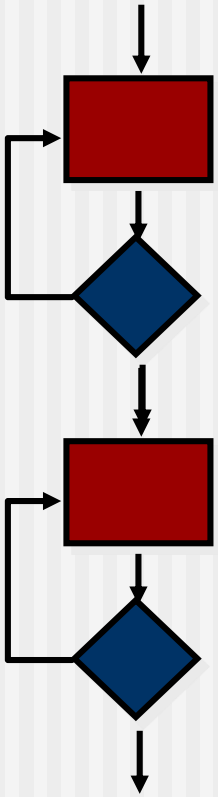
Loop Testing



Simple loop



Nested Loops



Concatenated Loops

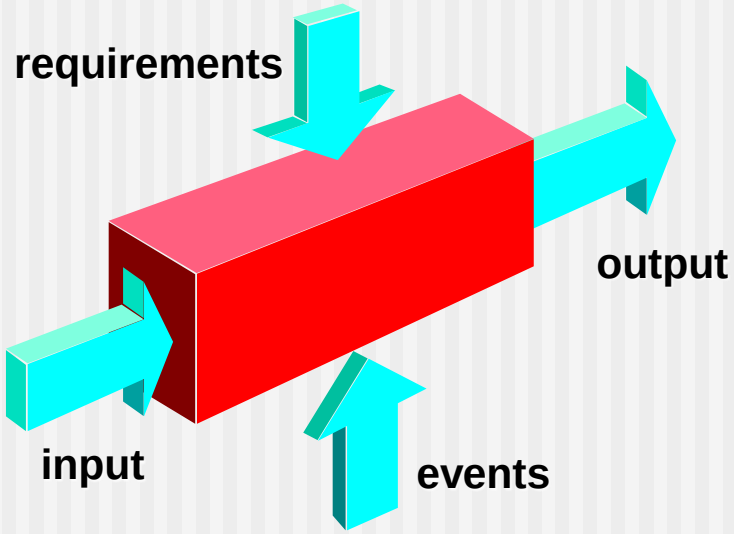
Loop Testing: Simple Loops

Minimum conditions—Simple Loops

1. **skip the loop entirely**
2. **only one pass through the loop**
3. **two passes through the loop**
4. **m passes through the loop m < n**
5. **(n-1), n, and (n+1) passes through the loop**

where n is the maximum number of allowable passes

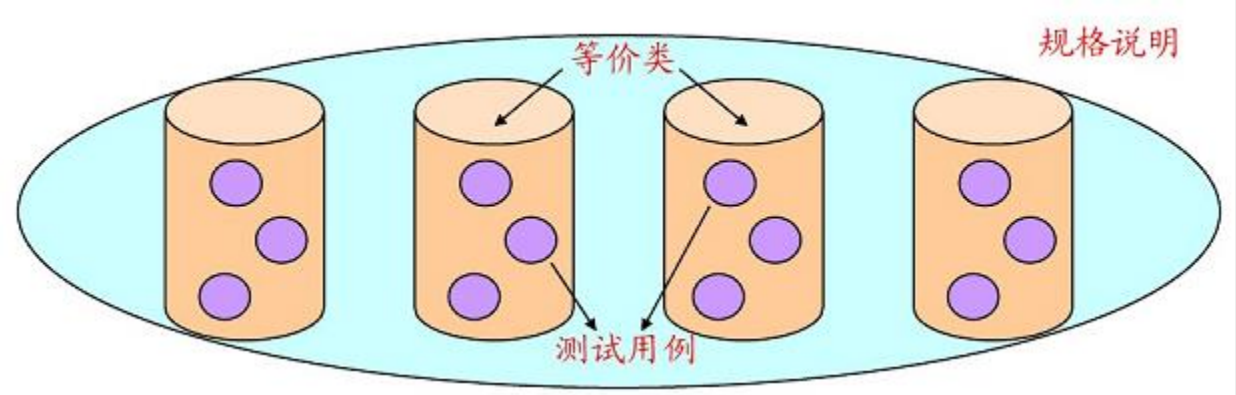
Black-Box Testing



- 这种方法是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构，只依据需求规格说明书，检查程序的功能是否符合它的功能说明。
- 黑盒测试一般在应用于测试的后期阶段。

Equivalence Partitioning

- 等价类划分法：将程序的输入域划分为若干个数据类，从中生成测试用例。
- **假定**：对等价类中的代表值测试就等于对这一类值的测试。



等价类划分法设计实例

基本步骤：　(1)确定等价类;(2)为等价类选择一个测试用例

功能描述：某城市的电话号码由3部分组成。假定被测程序能接收一切符合下述规定的电话号码，拒绝所有不符合规定的电话号码。

- 地区码：空白或3位数字；
- 前缀：非‘0’或‘1’开头的3位数字；
- 后缀：4位数字。

(1) 确定等价类

输入条件	有效等价类	无效等价类
地区码	空白(1) 3位数字(2)	有非数字字符(5) 少于3位数字(6) 多于3位数字(7)
前缀	从200到999的3位数字(3)	有非数字字符(8) 少于3位数字(9) 多于3位数字(10)起始位为‘0’(11) 起始位为 ‘1’ (12)
后缀	4位数字(4)	有非数字字符(13) 少于3位数字(14) 多于3位数字(15)

等价类划分法设计实例

(2) 设计测试用例

- 设计测试用例，使其尽可能多的覆盖到尚未覆盖的有效等价类
- 设计测试用例，使其只覆盖一个尚未被覆盖的无效等价类

测试用例	测试范围	期望结果
276 2345	等价类(1) (3) (4)	有效
027 805 9321	等价类(2) (3) (4)	有效
20A 223 4356	等价类(5)	无效
剩下的10个用例	无效等价类(6)-(15)	无效

Boundary Value Analysis

- A greater number of errors occurs at the boundaries of the input domain rather than in the "center."
- Boundary value analysis leads to a selection of test cases that exercise bounding values
- 边界值应该尽可能选择等于边界，略小于边界，略大于边界的值
- 边界值法可与等价类划分法结合，作为对其的一种补充。

Boundary Value Analysis

例：边界值分析

- 一个输入文件应该包含1~255条记录，则应用边界值分析法设计测试用例。

- 可以用4组数据进行测试：
  - 含1条记录的文件
  - 含2条记录的文件
  - 含254条记录的文件
  - 含255条记录的文件

Orthogonal Array Testing

- Used when the number of input parameters is small and the values that each of the parameters may take are clearly bounded

- 正交数组测试通过挑选适量的、有代表性的点进行试验，能有效地减少测试用例数，节约测试成本。
- 该方法由日本口玄一博士提出并导出正交数组，从而使得测试数据具有“均匀分散、整齐可比”的特点。

Testing Example（1）

- 某大学刚考完某门课程，想通过“性别”、“班级”和“成绩”这3个查询条件对这门课程的成绩查询，查询条件包括：
  - “性别”=“男，女”
  - “班级”=“1班，2班”
  - “成绩”=“及格，不及格”
- 一般需要设计2X2X2=8个测试用例

Testing Example（2）

3是指的3个因子(变量)

- 利用正交数组法，测试用例个数是N=3*(2-1)+1=4
- 根据所确定的因子数和每个因子的取值数，选择合适的正交表，并进行设计。

2是指的每个因子最大取值数

$L_4(2^3)$

列号 / 试验号	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

序号	性别	班级	成绩
1	女	1班	及格
2	女	2班	不及格
3	男	1班	不及格
4	男	2班	及格

- 也可以通过工具产生用例，如allpairs

Model-Based Testing

- Analyze an existing behavioral model for the software or create one.
- Traverse the behavioral model and specify the inputs that will force the software to make the transition from state to state.
  - The inputs will trigger events that will cause the transition to occur.
- Review the behavioral model and note the expected outputs as the software makes the transition from state to state.
- Execute the test cases.

Model-Based Testing