第 3 章　软件测试用例的设计

在软件测试过程中，测试用例的设计是软件测试的灵魂。测试工程师就是借助测试用例的运行来检测被测软件的功能和性能。测试结果的广度和深度完全由测试用例来决定。完全覆盖测试是最容易想到的一类用例，它要求测试工作的力度和深度以及每一种现实中可能发生的操作都保证正确无误。软件测试中永远不可能做到穷举测试，然而又想使测试工作的效率达到最高，那么，如何兼顾工作量和工作效率往往成为测试工作中的瓶颈问题。如何测试，用什么方式来测试，在什么环境和什么样的条件下进行测试，如何减少测试的工作量和如何避免重复的测试等，测试用例都应该考虑在内。在测试工作中如何协调和同步，在测试用例也应该充分描述这些问题。

经典软件测试理论有很多测试用例设计方法，总的来说，这些方法可以分为两大类，一类是黑盒测试，一类是白盒测试。

随着面向对象技术的发展，尤其是 UML 语言的广泛应用，测试用例设计又遇到了许多新的问题。为了适应面向对象这一新的技术，人们提出了很多实用的面向对象测试用例设计方法。例如，为了解决面向对象技术中大量的消息传递，对于对象交互等方面的测试，可以借助 MM(Method/Message)图来进行测试用例设计。MM 图是一种新的路径测试方法，与普通的 DD(Decision-to-Decision path)路径图比较，它更凸显了对象的交互。

本章将系统的介绍黑盒、白盒及面向对象测试用例设计的基本内容，下节将会详细的讲解黑盒测试及其测试用例设计方法。

3.1　黑 盒 测 试

黑盒测试是一种常用的软件测试方法。据统计，将近 80％的软件缺陷都是利用黑盒测试完成的。那么什么是黑盒测试呢？黑盒测试强调了软件输入与输出之间的关系，它将被测软件看作一个打不开的黑盒，根据软件规格说明书设计测试用例，完成测试。

我们学习过“函数”的定义，函数是把一个集合(定义域)的值映射到另一个集合(值域)上面。从某种意义上，可以把软件(程序)的输入和输出看作是函数的定义域和值域。黑盒测试就是从这种观点出发，测试人员把软件(程序)看作一个打不开的黑盒，只关心输入与输出的结果。

本节主要介绍几种常用的黑盒测试方法，并通过实例介绍各种方法的运用。之后，在第 7 章将介绍几种常用的黑盒测试工具，并结合实例重点介绍 IBM Rational 系列的 Function Tester 测试工具的应用。

3.1.1　边界值测试

大量的软件测试实践表明，软件缺陷经常出现在物理数值的边界上，因此利用边界值分析方法进行软件测试，是一种很实用的方法，它具有很强的故障检测能力。这一测试方法适

用于分析独立的变量，而这一变量常常是物理量（如温度、速度等）。边界值测试方法所分析的物理量，可以从两方面来考虑，一方面是针对软件输入输出中所使用的数据，另一方面是程序内部所涉及到的一些关键性的物理数据。

举一个简单的例子，程序员由于疏忽，在进行程序设计时，将图 3.1 所示的程序段，写成图 3.2 所示的程序段，造成很难发现的软件缺陷。

```
import java.util.Scanner;
public class guessNumb{
  public static void main(String args[]){
    int y=39;
    Scanner reader=new Scanner(System.in);
    System.out.println("输入您猜测的数字（0-100）")
    int x=reader.nextInt();
    If(x<y){
      System.out.println("您猜测的数字偏小");
    }
    Else{
      If(x==y)
        System.out.println("太聪明了")
      Else
        System.out.println("您猜测的数字偏小")
    }
  }
}
```

图 3.1　程序段 1

```
import java.util.Scanner;
public class guessNumb{
  public static void main(String args[]){
    int y=39;
    Scanner reader=new Scanner(System.in);
    System.out.println("输入你猜测的数字（0--100）")
    int x=reader.nextInt();
    If(x<=y){
      System.out.println("您猜测的数字偏小");
    }
    Else{
      If(x==y)
        System.out.println("太聪明了")
      Else
        System.out.println("您猜测的数字偏小")
    }
  }
}
```

图 3.2　程序段 2

图 3.1 和图 3.2 提供两个程序段，类 guessNumb 为猜数游戏类。图 3.1 可以正确地得到用户想要的结果。图 3.2 为存在缺陷的程序段，但是如果不使用边界值“39”很难发现该缺陷。所以边界值是一个很实用的，并且是不可或缺的测试用例设计方法。

边界值分析方法的基本思想就是利用输入变量的边界处的值进行测试。一般情况下，

所谓边界值包括最小值、最大值、略大于最小值的值、略小于最大值的值以及正常值。

1. 边界值测试方法

1) 单变量边界值分析

边界值分析法主要着眼于输入空间的边界。其中最常用的方法莫过于“五点法”,也就是上面所说的取变量的最小值、最大值、略大于最小值的值、略小于最大值的值以及正常值这 5 个值。

例 3.1 分析整型变量 x(a≤x≤b),区间[a,b]为变量 x 的取值范围。

分析:利用边界值分析方法进行测试用例设计。

最小值:a(记为 min);

略大于最小值的值:a+1(min+);

正常值:c(a+1≤x≤b-1)(nom);

略小于最大值的值:b-1(max-);

最大值:b(max)。

利用以上 5 个值来进行五点法边界值测试。

此例是针对单一变量 x 进行边界值分析,显然用五点法就可以。但是针对物理量的不同,在选取“略大于最小值的值”和“略小于最大值的值”时要考虑步长的问题。

例 3.2 某个输入文件最多可容纳 255 条记录,根据五点法进行边界值分析。

分析:利用边界值分析方法进行测试用例设计。

(1) 将文件放入 1 条记录,测试应用程序(min);

(2) 将文件放入 2 条记录,测试应用程序(min+);

(3) 将文件放入 50 条记录,测试应用程序(nom);

(4) 将文件放入 254 条记录,测试应用程序(max-);

(5) 将文件放入 255 条记录,测试应用程序(max)。

例 3.3 某企业员工月工资从 1000 到 2000 不等,个人所得税费按月扣除,计算得最小金额从 0.00 元到最大金额 4646.89 元不等,根据五点法进行边界值分析。

分析:利用边界值分析方法进行测试用例设计。

(1) 测试扣除个人所得税 0.00(min);

(2) 测试扣除个人所得税 0.01(min+);

(3) 测试扣除个人所得税 2000(nom);

(4) 测试扣除个人所得税 4646.88(max-);

(5) 测试扣除个人所得税 4646.89(max)。

例 3.2 和例 3.3 同样是采用五点法进行边界值分析,在例 3.2 中所采用的步长为 1,而例 3.3 中所采用的步长为 0.01。步长的选择要根据实际情况来分析。

2) 多变量边界值分析

以上分析的问题都是针对于一个变量的边界值用例设计,而作为程序的输入(输出)往往都是多个变量同时参与计算过程中。那么当变量数大于 1 时,如何采用五点法进行边界值分析呢?

下面讨论两个变量 x、y,并且给出 x 和 y 的边界条件(取值范围):

$$a \leqslant x \leqslant b; c \leqslant y \leqslant d$$

根据五点法的要求，选取(x_{min}, y_{nom})，(x_{min+}, y_{nom})，(x_{nom}, y_{nom})，(x_{max-}, y_{nom})，(x_{max}, y_{nom})，(x_{nom}, y_{min})，(x_{nom}, y_{min+})，(x_{nom}, y_{max-})，(x_{nom}, y_{max})。如图3.3所示。同样的道理，对于n个变量的程序，采用五点法将产生$4n+1$个测试用例。

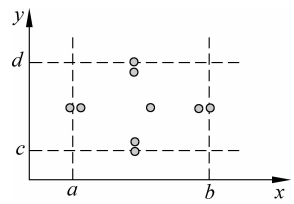


图3.3 多变量五点法测试用例设计

3）健壮性边界值分析

“健壮性”这个词，经常出现在软件测试领域，包括系统测试时的健壮性测试和这里的健壮性边界值分析。有关健壮性的测试往往是检测无效的未预料到的输入和输出。尤其在无效的输出方面，健壮性测试有着不可小觑的能力。

健壮性边界值分析也常常被称为“七点法”边界值测试。也就是对于单变量x(a≤x≤b)而言，除了考虑五点法中的五点：min、min+，nom，max−和max之外，还需要分析略小于最小值的值min−和略大于最大值的值(max+)。同样的道理，对于多变量健壮性边界值分析时，假设变量数n，需要设计$6n+1$个测试用例。

2. 软件输入输出中存在的边界问题

在进行软件输入输出边界测试时，需要注意以下一些方面。

(1) 测试的数据类型，其中包括数值、字符、位置、数量、速度、地址和尺寸等。

(2) 测试数据的边界特征值，如第一个/最后一个、开始/完成、空/满、最慢/最快、相邻/最远、最小值/最大值、超过/在内、最短/最长、最早/最迟、最高/最低等。此外还包括以下几项：

① 一些特殊的字符：@、'、{、/、:等，以及特殊字符的ASCII码。

② 常用的边界还包括：数字0～9，对应ASCII码48～57；大写字母A～Z，对应ASCII码65～90；小写字母a～z，对应97～122。

③ 对于图形设计类程序，还需考虑显示范围的边界、光标最上端和光标最下端的边界等内容。

输入输出边界值分析经常和等价类方法一同使用，具体内容在3.1.2节介绍。

3. 程序内部所涉及的边界问题

在图3.2所示的程序段中，软件缺陷并非是输入输出变量导致的，也就是说，在进行软件测试时，还需要考虑程序内部某些关键变量的边界值。

在进行程序内部变量边界值测试时，需要注意以下几个方面的内容。

(1) 计算机和软件的基础是二进制数。因此与2的乘方相关的值是作为边界条件的重要数据。

例如，在通信软件中，带宽或者传输信息的能力总是受限制的，因此软件工程师会尽一切努力在通信字符串中压缩更多数据。其中一个方法就是把信息压缩到尽可能小的单元中，发送这些小单元中最常用的信息，在必要时再扩展为大一些的单元。假设某种通信协议支持256条命令。软件将发送编码为一个双位数据的最常用的15条命令；如果用到第16～256条命令，软件就转而发送编码为更长字节的命令。这样，软件就会根据双位/字节边界执行专门的计算和不同的操作。

(2) 对于数组型数据int a[]=new int[20]，以a[0]以及a[19]作为其边界值。

(3) 对于循环语句

```
for(i=1;i<=n;i++){
```

```
循环体
}
```

第 0 次循环和第 n 次循环作为其边界条件。

(4) 对堆栈类型数据结构的数值进行测试时，对于该堆栈而言，为空(null)、满(full)分别可作为其边界条件。

当然，还有其他很多种涉及数值型、字符型、速度值以及其他物理量的数据，都可以采用边界值方法对其进行测试。

4. 边界值法测试用例设计的局限性

边界值分析方法具有很强的缺陷发现能力，是一种设计简单，但很实用的测试方法。边界值设计测试用例方法适合于分析独立的变量，并且这些变量表示的是实际的物理量(如速度、时间、温度等)。也就是说，边界值分析方法所测试的变量要求是独立的并且是物理量。而边界值测试方法就是取这些物理量的临界值或极值作为测试数据。

例如，由于 1992 年 6 月 26 日菲尼克斯气温达到 122℉，菲尼克斯的天港国际机场被迫关闭，而此次事件仅仅因为机场部分设备接收的气温上限是 120℉。再举个例子，在飞机飞行过程中，如果飞机的飞行仰角达到其极限，很可能会造成严重的飞行事故。也就是说，在应用程序中，对于含有实际物理意义的变量进行边界值分析是非常重要的，对于某些变量来说也是必不可少的一个过程。

边界值分析方法并不是不能分析有依赖关系的变量，比如经典的三角形问题中(a，b，c 为输入的 3 条边)，依然可以采用边界值方法进行分析，但是由于边界值方法的特点，这种设计用例的过程并不能考虑到变量之间的关系，比如 a＜b＋c 这样的依赖关系。所以一般认为，使用边界值分析方法分析多变量时，多变量之间的关系是独立的，这也正是边界值分析方法存在的很大的缺陷。

3.1.2 等价类测试

不论使用哪种测试用例设计方法，其根本意义都是要减少测试用例数量。然而在减少测试用例数量的同时，还希望测试工作找出更多的错误。因此测试工程师就想到借助于“等价类”的思想来建立测试用例集，使得在减少测试用例绝对数量的同时能够保证测试用例的质量。

首先讨论等价类的基本概念以及它在测试过程中的重要意义。在分析等价类之前，需要了解等价关系的概念。

等价关系 R 为非空集合 A 上的关系，如果 R 是自反的、对称的和传递的，则称 R 为 A 上的等价关系。那么什么是等价类呢？假设 x 为 A 上的元素，那么 x 的等价类记为 $[x]_R$，并满足：$[x]_R=\{y \mid y\in A \wedge xRy\}$，那么等价类中的元素均满足等价关系。

显然，如果将集合 A 看作针对某一个功能所设计的测试用例集合，那么 x 等价类内的元素都是等价的。也就是说，理论上只取 $[x]_R$ 中某一个测试用例来进行测试，和取 $[x]_R$ 中每一个测试用例进行测试所得到的测试结果应该是相同的。这样，利用等价类的概念可以找到冗余的测试用例，用以减少测试用例的绝对数量。

那么如何来进行等价类划分呢，在进行等价类划分时一定要注意两点，一个是划分的完备性，另一个是冗余性。一般情况下，在进行测试的过程中，使用等价类划分方法，都是针对

程序的输入输出域来进行等价类划分，从而设计等价类和相应的测试用例。

1. 等价类划分方法

在了解等价类划分的定义之后，先来看一个等价类划分的小例子。

例 3.4 图 3.4 是一个简单的 login 界面，在绝大部分 MIS 系统中，都需要用到的 login 这一简单又很重要的模块。

图 3.4 登录界面

图 3.4 为学生信息管理系统的登录界面，在规格说明书中对登录模块的描述如下。

要求 1：用户名使用学生学号，学号要求由 9 位数字组成，如 090705101。

要求 2：密码使用 4～8 位字符串。字符串由大小写字母、下划线“_”或数字组成。

很明显，该例子中使用等价类划分，可以针对输入域进行等价类分析。

分析：

(1) 要求用户名(也就是学号)由 9 位数字组成，等价类划分如下。

等价类 1：9 位数字，为一组等价类(090705101；070405206；100705102)；

等价类 2：非 9 位数字，为一组等价类(0907051；10070)；

等价类 3：用户名中含有字母和其他字符。

(2) 要求密码使用 4～8 位字符串，等价类划分如下。

等价类 4：4～8 位字符串，为一组等价类；

等价类 5：非 4～8 位字符串，为一组等价类。

(3) 要求字符串由大小写字母、下划线“_”或数字组成，等价类划分如下。

等价类 6：字符串包含大小写字母、下划线“_”或数字；

等价类 7：字符串包含特殊字符(空格、¥、#、@等)。

从例 3.4 可以看出，等价类划分方法就是将输入域(输出域)进行等价划分。划分过程一般是按照规格说明书的内容，由测试工程师根据实际情况来操作的。也就是说，不同的测试人员设计出的等价类不一定是相同的。因此，在具体操作的时候，整个等价类设计过程要遵循这样一个原则：在划分等价类时要考虑到无效和未预料到的情况，这一点在输出域等价类划分过程中尤其重要。根据这一原则，把等价类划分为有效等价类和无效等价类。

(1) 有效等价类：符合程序规格说明书，有意义的、合理的输入(输出)数据所构成的集合。

(2) 无效等价类：不符合程序规格说明书，不合理的或者无意义的输入(输出)数据所构成的集合。

一般在具体的问题中，有效等价类可以是一个，也可以是多个；而无效等价类至少应有一个。对于例 3.4，很明显，等价类 1、4、6 为有效等价类，等价类 2、3、5、7 为无效等价类。

另外，在等价类划分的时候一定要注意划分的完备性和非冗余性。完备的划分保证了测试用例能够覆盖所有的输入域，没有遗漏；非冗余性使得划分更加合理，测试用例质量更高。

2. 等价类划分原则

虽然等价类划分是一种随机性比较强的测试方法，不同测试人员会得到不同的划分结果，但是还是有一定规律可循的。这些规律是很多测试人员在工作中总结提炼而得到的，有一定的通用性，这对初学者而言是很有帮助的。

(1) 区间划分：规格说明对数据规定了明显的取值范围。比如，血压值为 50～200，学生学号为 070000001～120000001。对这一类数据，等价类可取区间内(有效等价类)数值和区间外(无效等价类)数值。

(2) 集合划分：规格说明对数据规定了明确的集合，比如管理员(admin 和 superadmin)。等价类可取集合内(有效等价类 admin)和集合外(无效等价类 Tom)不同集合值。

(3) 特殊规则划分：规格说明规定了数据必须遵守一定的限制条件。比如，根据例 3.4 中的要求 2，可以确定等价类 4 满足规则说明，等价类 5 不满足规则说明。

(4) 细分等价类：等价类在具体程序处理时发现并不真正等价。比如，学号 090705101、070405206 和 100705102 分别为该校 2009 级学生、2007 级学生和 2010 级学生，那么上述学号在选课、学籍管理等功能模块中并不等价。此时可以根据具体情况来细分相应的等价类。

3. 等价类划分测试用例设计

在进行测试用例的具体设计时，也要考虑到有效等价类和无效等价类。一般情况下，我们希望一个测试用例能够覆盖较多的有效等价类；同时，一个测试用例覆盖并且只能覆盖一个无效等价类，这是由有效等价类和无效等价类的特点决定的，其目的是防止第一个无效等价类的测试会屏蔽或者终止其他无效等价类的测试执行。比如，例 3.4 设计的测试用例如表 3.1 所示。

表 3.1 测试用例

测试用例	输　　入	期 望 输 出	覆盖等价类
T1	用户名：090705101 密码：a_bcde	满足登录条件数据库查询匹配情况	1,4,6
T2	用户名：0907051 密码：a_bcde	提示：用户名为 9 位的学号	2
T3	用户名：0907051a 密码：a_bcde	提示：用户名为 9 位数字串	3
T4	用户名：0907051a 密码：a_b	提示：密码为 4～8 位字符串	5
T5	用户名：0907051a 密码：a@b	提示：密码为 4～8 位字符串，不能包含特殊字符	7

4. 等价类划分方法小结

一般进行等价类划分需要两个步骤：

(1) 根据规格说明书的内容对输入域(输出域)划分等价类。

(2) 根据划分的等价类进行测试用例设计。

具体等价类执行步骤如下：

(1) 划分等价类，将每一个等价类进行编号。

(2) 标记等价类是有效等价类还是无效等价类。

(3) 设计测试用例。

① 设计有效等价类测试用例，尽可能多地覆盖所有有效等价类。

② 设计无效等价类测试用例，一个无效等价类测试用例只能覆盖一个无效等价类。

利用等价类法测试，结果如表 3.2 所示。读者可按照表 3.2 的形式设计测试用例，以防遗漏某一等价类式测试条件。

表 3.2　等价类表

输入(输出)条件	编号	有效等价类	无效等价类

等价类划分经常和边界值分析联合起来使用。如果等价类划分的数据是独立的物理数据，也就是满足边界值测试条件的话(比如“血压(50～200)”)，可以取等价类的边界值(50，60，75，190，200)。

3.1.3　因果图

等价类划分和边界值分析是很有效的测试方法，但是当多变量之间关系不是独立的，有复杂的逻辑关系的时候，这两种方法就显得束手无策了。举个简单的例子，某传感器误差 Δx 随着温度的影响累计增大，可表示为 $\Delta x = f(\Delta c)$，其中 Δc 为温度的变化，当 Δx 超过阈值 d 时发生失效，那么等价类和边界值方法在处理这些问题时，很难发现这种类型的失效。为了很好地表示出输入数据之间的组合逻辑关系，可以借用“因果图”这一有效的方法。

因果图法是由美国 IBM 公司的 Elemendorf 在吸收了硬件测试中自动生成逻辑组合电路测试等技术的基础上于 1973 年提出的，它是进行功能测试时把功能说明书形式化的一种表示方法。因果图是一套严谨的知识表示方式，它类似于数字逻辑电路，可以清晰地表达组合数据之间的布尔逻辑关系。使用因果图方法表示数据，不但有助于测试工作的进行，还可以发现规格说明中描述不完整或者存在二义性的内容。一般在使用因果图来表示输入域或输出域时，往往还需借助决策表来进行测试用例设计。关于决策表的内容在 3.1.4 节将会作详细的介绍。

1. 因果图中使用的符号

由于因果图表示的数据主要体现了数据之间的布尔逻辑关系，所以可借助数字逻辑电路的与(∧)、或(∨)、非(～)等符号表示。在因果图中，用圆圈表示节点(原因或者结果)，用直线连接相应的节点。

在连接因果的直线上面可以标出节点之间的关系。如果是原因节点和结果节点，那么节点之间的关系主要有以下 4 种：

(1) 恒等：如果原因为真，结果必为真。

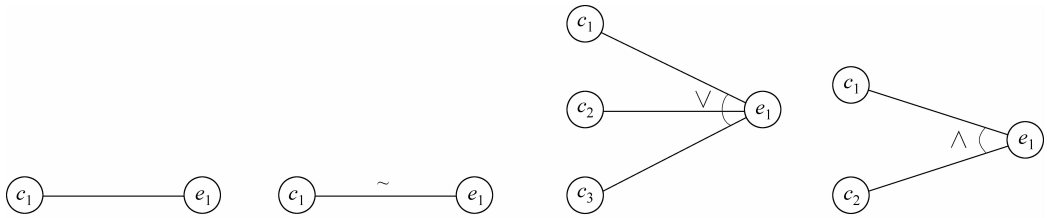
(2) 非：如果原因为真，结果必为假。

(3) 或：如果原因组合(如 $c_1 \vee c_2 \vee c_3$)为真，结果为真。

(4) 与：如果原因组合(如 $c_1 \wedge c_2$)为真，结果为真。

节点间的上述 4 种关系用因果图来表示如图 3.5 所示。

因果图除了明确给出原因和结果之间的关系，还给出原因之间的约束关系。原因之间的约束关系有以下 4 类。



(a) 恒等　(b) 非　(c) 或　(d) 与

图 3.5　因果图中原因和结果之间的关系

(1) Exclusive(E)：原因 a、b 不可能同时设置为 1。也就是说 a、b 之间存在排斥的关系。在实际情况中，原因之间的 E 关系比较多，比如，a 字符串首字符为字母 a～z，b 字符串首字符为数字。a、b 两种情况不可能同时成立，存在排斥关系。

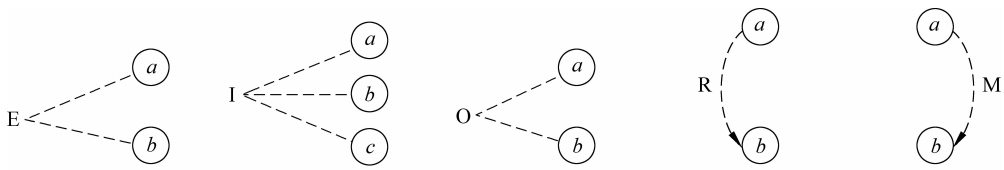
(2) Inclusive(I)：原因 a、b、c 至少有一个为 1。a、b、c 不能同时为 0。

(3) Only one(O)：原因 a、b 必须有一个为 1，且仅有一个为 1。

(4) Require(R)：原因 a 是 1 时，要求 b 必须为 1。也就是说，a 为真对 b 有制约关系。

有时结果之间也需要建立约束关系，如果结果 a 为 0，则 b 强制为 0。这种约束关系用 Mask(M)来表示。

因果图中的约束关系如图 3.6 所示。



(a) 异或　(b) 或　(c) 唯一　(d) 要求　(e) 强制

图 3.6　因果图中的约束关系

2. 因果图示例

利用因果图进行测试时，首先要明确测试内容，做到有的放矢。把规格说明中需要测试的内容(比如图书管理系统中的图书信息查询功能)找到，利用因果图对规格说明书的内容进行形式化表示。“因”一般指输入条件或者输入条件的等价类；“果”一般指输出条件，或者后续将要进行的操作，或者系统状态转换等。

例 3.5　在某个嵌入式软件报警系统中，如果设备超速运行，则立即发出消息：“警告：该系统超速运行，误差将会增大!”；如果环境发生变化，则立即发出消息：“错误：系统环境不符合要求，请立即关闭软件!”；如果操作员发生误操作，则立即发出消息：“错误：请重新操作!”

分析：系统故障分为 3 类，即设备超速、环境不符合要求以及操作员误操作。

下面细分这 3 类故障。

(1) 设备超速：可能是由于加速度 $\alpha > a$(固定值)，或者速度 $v > b$(固定值)。

(2) 环境不符合要求：可能是清晰度太差导致软件运行不正常，或者环境的温度或湿度不符合要求。

(3) 操作员发生误操作：操作员操作顺序或者数据输入错误，从而导致系统状态转换错误，或者计算结果发生错误。

首先根据规格说明书确定原因和结果。

原因：c_1：加速度 $\alpha > a$。

c_2：速度 $v > b$。

c_3：清晰度太差。

c_4：温度不符合要求。

c_5：湿度不符合要求。

c_6：操作顺序错误。

c_7：输入数据错误。

c_8：系统状态转换错误。

c_9：计算结果错误。

结果：e_1："警告：该系统超速运行，误差将会增大！"

e_2："错误：系统环境不符合要求，请立即关闭软件！"

e_3："错误：请重新操作！"

然后找到原因和结果之间的因果关系，以及原因和原因之间的约束关系，得到因果图，如图 3.7 所示。

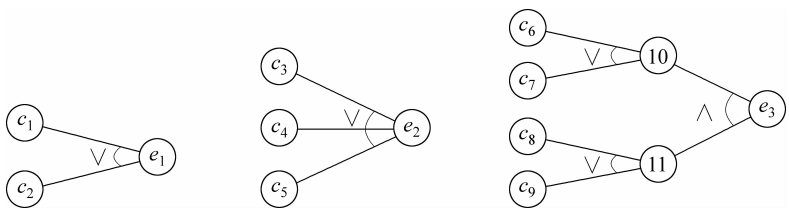


图 3.7　例 3.5 的因果图

以上只是因果图的一个简单的例子，事实上，因果图在处理复杂的问题时是很有效的一个方法，它不但能够有助于完成测试，还能很好地表示功能需求，发现功能需求中存在二义性和不完整的内容。那么接下来的测试用例设计，还需借助决策表或层次树来进行分析。3.1.4 节将介绍借助因果图和决策表对例 3.5 进行测试用例设计的方法。

3.1.4　决策表

决策表是最具逻辑性的测试方法，和因果图法有重叠的地方。决策表可以用来分析和表达多逻辑条件下执行不同操作的程序。自 20 世纪 60 年代以来，决策表一直被用来描述对象之间的复杂逻辑关系。

1. 决策表基本概念

首先介绍决策表的基本内容。一个决策表由 4 部分组成，分别是条件桩、条件项、动作桩和动作项。

(1) 条件桩：列出问题的所有条件。

(2) 条件项：针对条件桩给出的条件列出所有可能的取值。

(3) 动作桩：列出问题规定的可能采取的操作。

(4) 动作项：指出在条件项的各组取值情况下应采取的动作。

另外，一个条件组合的特定取值和相应要执行的操作称为一条规则。在决策表中，一列就是一条规则。

下面通过一个简单的例子学习决策表的应用。

例 3.6 在图书管理系统中，还书模块，管理员录入书目信息。还书过程中利用读者借书的时间以及对书的损坏程度等因素来计算罚款数、读者信誉和读者级别，完成还书操作。

还书操作具体说明如下：

读者级别如表 3.3 所示。

表 3.3 读者级别

读者级别	借阅时间/月	超期罚款额/元/(天·本)	可借本数
SVIP	5	0.2	10
VIP1	2	0.3	5
VIP	1	0.5	3

损坏程度分为 3 个级别，分别为正常磨损、损坏和严重损坏(丢失)。

读者信誉分为 10 个层次，当读者按时还书并且书处于正常磨损状态，读者信誉自动加 1；如果书处于严重损坏或者丢失状态，读者级别自动降为 VIP 级；如果读者超期时间超过 1 年，读者级别也自动降为 VIP 级别；如果书处于损坏状态，信誉自动减 1。当读者达到最高信誉时，读者级别可以自动提升一级。

分析：该图书管理系统的还书操作计算较复杂，变量之间存在明显的逻辑关系。考虑使用决策表方法进行测试用例分析(这里假设用户级别为 VIP1。读者可自行设计 VIP 和 SVIP 级别的还书操作决策表)。

条件桩：问题的所有条件。

C1：读者借书时间；

C2：对书的损坏程度。

条件项：对条件桩给出的条件列出所有可能的取值。

C1：C 正常(Normal)，E 超期(Exceed)，S 严重超期(Super Exceed)；

C2：C 正常(Normal)，E 损坏(Destory)，S 严重损坏(Super Destory)。

动作桩：出现问题时按规定可能采取的操作。

A1：交罚款；

A2：信誉加 1；

A3：信誉减 1；

A4：降级为 VIP。

动作项：指出在条件项的各组取值情况下应采取的动作。

分析这个过程，建立决策表，根据规格说明填写动作项，并完成决策表的建立。本例的决策表如表 3.4 所示。

决策表适合于描述具有复杂逻辑条件的程序，它能够将复杂的问题按照各种可能的情况全部列举出来，表示形式简洁清晰。还可反过来帮助需求分析人员找到描述不准确或者

表 3.4 决策表

条件桩	条 件 项								
C1	Normal	Normal	Normal	Exceed	Exceed	Exceed	SuperExceed	SuperExceed	SuperExceed
C2	Normal	Destory	Super Destory	Normal	Destory	Super Destory	Normal	Destory	Super Destory
动作桩	动 作 项								
A1				√	√	√	√	√	√
A2	√								
A3		√			√				
A4			√			√	√	√	√

遗漏的需求。

决策表测试法适用于具有以下特征的应用程序：①if-then-else 逻辑突出；②输入变量之间存在逻辑关系；③涉及输入变量子集的计算；④输入与输出之间存在因果关系。

一般情况下，规格说明以决策表形式给出，测试工作可以直接在此基础上进行。另外，要注意，决策表虽然能表达逻辑关系复杂的变量，然而决策表不能体现出变量之间的顺序关系。也就是说，条件(规则)的排列顺序不会也不应影响执行的动作。假如条件项或者动作项存在顺序关系，则并不适合使用决策表这一方式来进行描述。

2. 测试用例的设计

测试用例就是在决策表的基础上完成的，一般情况下，一条规则就是一个测试用例。例如，对于例 3.5，可以设计出 9 个测试用例。设计结果如表 3.5 所示。

表 3.5 测试用例

测试用例	规则号	输 入 数 据	预 期 输 出
Case1	1	VIP1 用户借书 20 天，无磨损	罚款数 0，信誉加 1；如果信誉值为 10，则提高级别为 SVIP
Case2	2	VIP1 用户借书 20 天，磨损	需罚款，信誉减 1；如果信誉值降为 0，则降低级别为 VIP
Case3	3	VIP1 用户借书 20 天，书丢失	需罚款，降级为 VIP
Case4	4	VIP1 用户借书 3 个月，无磨损	需罚款
Case5	5	VIP1 用户借书 3 个月，磨损	需罚款，信誉减 1；如果信誉值降为 0，则降低级别为 VIP
Case6	6	VIP1 用户借书 3 个月，书丢失	需罚款，降级为 VIP
Case7	7	VIP1 用户超期一年，无磨损	需罚款，降级为 VIP
Case8	8	VIP1 用户超期一年，磨损	需罚款，降级为 VIP
Case9	9	VIP1 用户超期一年，书丢失	需罚款，降级为 VIP

利用决策表进行测试用例设计时，应先分析需求，根据需求创建决策表，并完成测试用例设计。当然这样可以做出比较完整的测试用例。但是当需求比较复杂，尤其是条件桩较

多时，测试用例的数量还是相对较大的一个数字，有时是测试人员难以完成的。假如有条件桩数 n，并且每个条件只有两个取值(真、假)，那么就会得到 2^n 个规则。当 n 较大时，决策表很烦琐。实际使用决策表时，常常先将它化简。接下来讨论决策表如何进行化简。

决策表的化简是以合并相似规则为目标。即，若表中有两条以上规则具有相同的动作，并且在条件项之间存在极为相似的关系，便可以合并。合并后的条件项用符号“—”表示，说明执行的动作与该条件的取值无关，称为无关条件。

读者可以自行将例 3.6 的决策表进行化简，看看测试用例数量是否减少了很多。当然化简决策表就会减少测试用例数量，使得测试结果并不如化简前那么完善，但是也不失其代表性。当测试用例数量多时，可以根据测试人员的需求化简决策表。

3. 因果图和决策表

利用 3.1.3 节介绍的因果图，最终没有得到相应的测试用例。事实上，因果图只是清晰地表达了需求分析的内容。如果要得到测试用例，就必须借助于决策表，也就是需要将因果图转化成决策表。

在因果图中已经分析了“因”和“果”，“因”和“果”直接可以作为条件桩和动作桩。根据条件桩的取值得到条件项，利用条件项和因果图中原因与结果的关系，可以得到相应的规则，最终生成决策表。

3.2 黑盒测试策略

测试用例的设计方法不是单独存在的，具体到每个测试项目中都会用到多种方法，每种类型的软件各有其特点，因此要针对不同的软件采取不同的黑盒测试方法。在实际测试中，往往是综合使用各种方法才能有效地提高测试效率和测试覆盖率，这就需要认真掌握这些方法的原理，积累更多的测试经验，以有效地提高测试水平。

以下是功能测试部分的各种黑盒测试方法的综合选择策略。

(1) 首先进行等价类划分，包括输入条件和输出条件的等价划分，将无限测试变成有限测试，这是减少工作量和提高测试效率最有效的方法。

(2) 在任何情况下都必须使用边界值分析方法。因为这种方法非常有效，设计出的测试用例发现程序错误的能力最强。

(3) 利用测试人员的经验可以在一些容易出现缺陷的地方追加测试用例，这需要依靠测试工程师的智慧和经验的积累。

(4) 如果程序的功能说明中含有输入条件的组合情况，尤其是各个输入条件之间存在依赖关系，则一开始就可选用因果图法和决策表法。

下面给出一些选取具体测试方法的简单标准：

(1) 如果变量引用的是物理量，可采用边界值分析测试和等价类测试。

(2) 如果变量引用的是逻辑量，可采用等价类测试和决策表测试。

(3) 如果变量是独立的，可采用边界值分析测试和等价类测试。

(4) 如果变量不是独立的，可采用决策表测试。

(5) 如果可保证是单缺陷假设，可采用边界值分析测试和健壮性测试。

(6) 如果可保证是多缺陷假设，可采用边界值分析测试和决策表测试。

(7) 如果程序包含大量例外处理,可采用健壮性测试和决策表测试。

黑盒测试技术是软件测试的主要方法之一。通过黑盒测试可以检查软件的每个功能是否都能正常运行,因此,黑盒测试也是从用户观点和需求的角度进行的测试。3.1节只介绍了一些常用的黑盒测试方法,当然还有其他的黑盒测试方法,需要读者在以后的学习中逐渐积累。下面对边界值分析、等价类和决策表这3种黑盒测试方法进行总结和比较。

边界值分析法是对程序输入或输出的边界值进行测试的一种黑盒测试方法。边界值分析法有其自身的特点。(1)所谓边界值,是基于输入和输出变量的定义域而言的,所以该方法不适合分析数据或逻辑关系。(2)用边界值方法进行测试用例设计,设计方法简单,工作量相对较小,然而生成的测试用例数量比较多,在执行测试时花费的时间会比较长。(3)由于边界值分析方法设计方法简单,所以很容易实现自动化,因此自动化工具对它支持得比较好。

等价类划分法是一种典型的、重要的黑盒测试方法,它将程序所有可能的输入数据划分为若干个等价类,然后从每个等价类中选取具有代表性的数据做为测试用例。等价类划分法的特点是:(1)等价类划分相对于边界值方法,更多地考虑了数据的依赖关系,所以设计方法相对复杂,工作量相对较大。(2)等价类划分所产生的测试用例数量属于中等。由于考虑了数据的关系,所以与边界值方法相比,测试用例数量要少一些,这就会节约测试执行的时间。(3)在标识等价类时需要更多的判断和技巧,等价类标识出以后的处理也是机械的。

决策表是分析和表达多逻辑条件下执行不同操作的情况。决策表最突出的特点是把复杂的问题一一罗列,便于理解,避免遗漏,可以方便地得到测试用例。另外决策表能够考虑到数据的逻辑依赖关系,可以得到完备的测试用例。也正是因为决策表这样的特点,使得这种方法设计工作量较大,不容易利用自动化工具实现。

图3.8分别从设计测试用例工作量和得到的测试用例数两个方面对边界值、等价类和决策表这3种方法进行对比。由于3种方法的复杂程度不同,所以设计测试用例工作量也不相同,其中决策表在测试过程中工作量最大,耗时最长,也相对难以自动化;然而该方法得到的测试用例数却是最少的,也就是执行测试用例时间上是最少的。

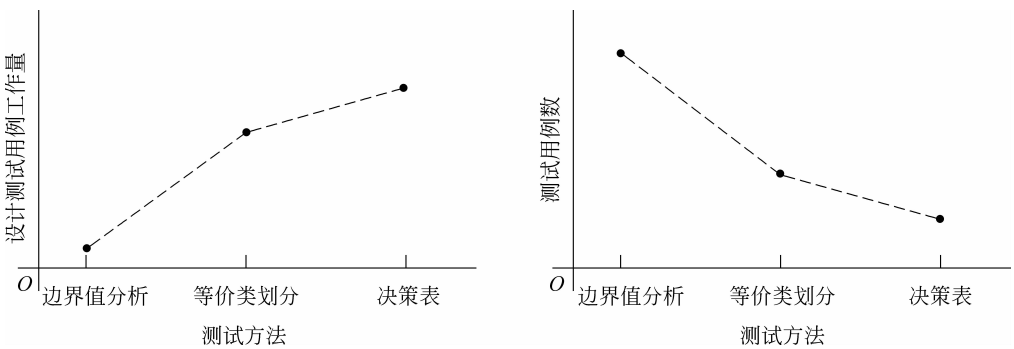


图3.8 黑盒测试方法比较

在实际应用中,需要结合所有黑盒测试的方法,利用程序结构或者变量的已知属性,选择处理这种属性的方法。在选择使用哪种黑盒测试方法的时候,经常需要考虑以下几个方面:①变量表示物理量还是逻辑量;②在变量之间是否存在依赖关系;③在实际应用中是

否有大量的例外处理，应结合这几种方法的特点，选择合适的黑盒测试方法。

3.3 白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是按照程序内部的结构测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看作一个透明的盒子，测试人员依据程序内部逻辑结构的相关信息，设计或选择测试用例，对程序的所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。

白盒测试的测试方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、域测试、符号测试、Z路径覆盖、程序变异等测试方法。这些方法又可以划分为两大类：静态测试方法和动态测试方法。其中软件的静态测试不要求在计算机上实际执行被测程序，主要以一些人工的模拟技术对软件进行分析和测试；而软件的动态测试是通过输入一组预先按照一定的测试准则构造的实例数据来动态运行程序，从而达到发现程序错误的目的。在动态分析技术中，最重要的技术是基本路径测试法。下面主要介绍路径测试、数据流测试和逻辑覆盖方法。

3.3.1 路径测试

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路(圈)复杂性，导出基本可执行路径集合，从而设计测试用例的方法。并且设计出的测试用例要保证在测试中程序的每个可执行语句至少执行一次。

1. 控制流图

白盒测试是针对软件内部逻辑结构进行的测试，因此，测试人员必须对软件内部结构、各个单元及相互之间的联系和程序的运行过程深入理解。因此，白盒测试是一项庞大、复杂的工作。为了更加突出程序的内部结构，便于测试人员理解源程序，但是又不过多地关注程序中的每一个处理及判断条件，可以对程序流程图进行简化，简化后的程序流程图称为控制流图(control flow graph)。控制流图主要由节点和边构成。一般的控制流图的结构如图3.9所示。控制流图中的每一个带编号的圆被称为一个节点，每个节点可以表示程序中的一条或多条语句，有分支的节点称为判定节点；每一条带箭头的线被称为一条边；由节点和边形成的空间称为区域。图3.9中共有3个区域：$2c3d4b$(区域1)，$1a2b4e5f$(区域2)和外围的区域(区域3)。

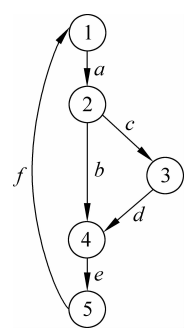


图3.9 控制流图

在路径测试中，先确定程序流程图，再将程序流程图转换为控制流图，转换的原则如下：控制流图中的每一个节点可以表示程序流程图中的以下3种内容：矩形框表示的处理；菱形框表示的两个甚至多个出口判断；多条流线相交的汇合点。程序流程图和转换的控制流图如图3.10所示。

如果判断中的条件表达式是由一个或多个逻辑运算符(OR，AND，NAND，NOR)连接的复合条件表达式，则需要改为一系列只有单条件的嵌套的判断。

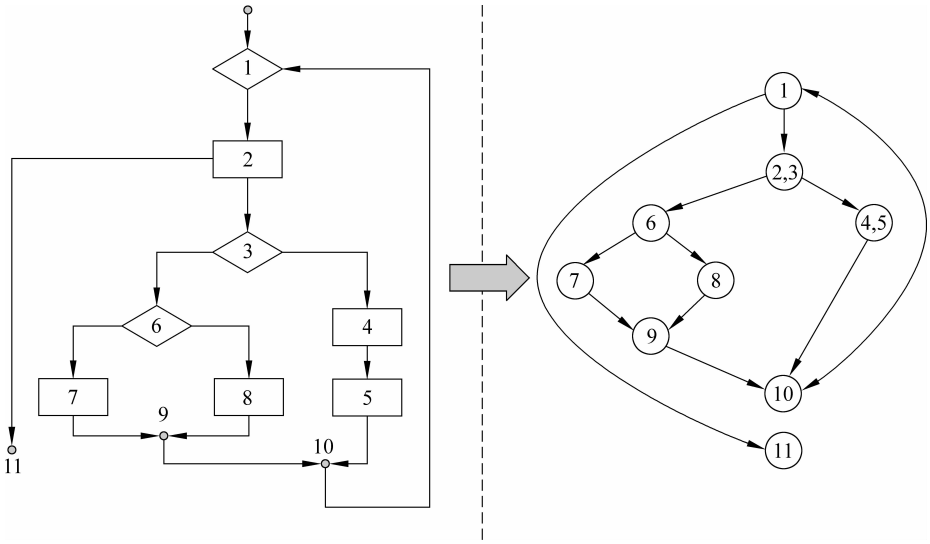


图 3.10　程序流程图和转换的控制流图

例如：

```
1 if a or b
2 x
3 else
4 y
```

对应的控制流图的逻辑结构如图 3.11 所示。

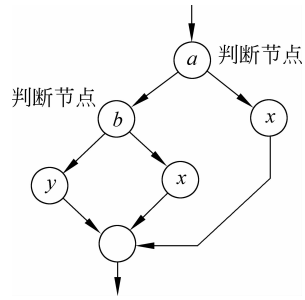


图 3.11　复合条件的逻辑结构

2. 环形(圈)复杂度

环形复杂度是一种为程序逻辑复杂性提供定量测度的软件度量，将该度量用于计算程序的基本的独立路径数目，是确保所有语句至少执行一次所必需的测试用例数量的上界。独立路径必须包含一条在定义之前未曾用到的边。环形复杂度的计算有如下几种方法。

方法一：流图中区域的数量对应于环形的复杂度。

方法二：给定流图 G 的环形复杂度 $V(G)$，定义为 $V(G)=E-N+2$，E 是流图中边的数量，N 是流图中节点的数量。

方法三：给定流图 G 的环形复杂度 $V(G)$，定义为 $V(G)=P+1$，P 是流图 G 中判定节点的数量。

因此，采用以上 3 种方法都可计算出图 3.10 的环形复杂度为 4，图 3.11 的环形复杂度为 3。

3. 路径测试方法应用举例

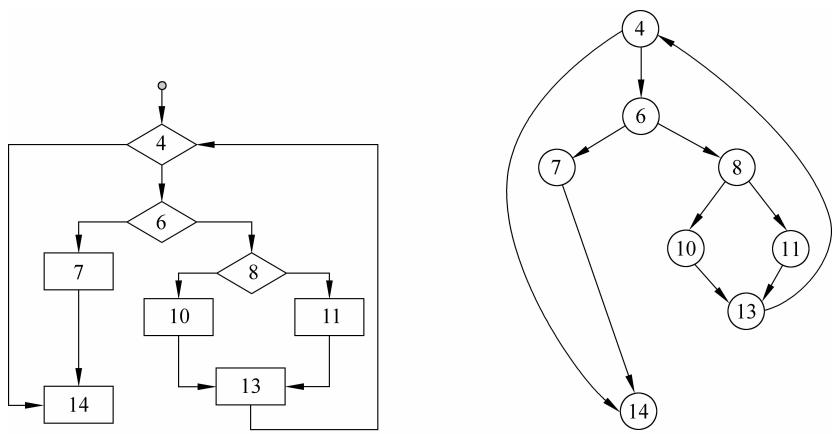
下面以具体的程序为例，介绍路径测试的基本应用。

```
Void Sort(int iRecordNum, int iType)
 1  {
 2      int x=0;
 3      int y=0;
```

```
4      while (iRecordNum->0)
5      {
6        if(0==iType)
7          x=y+2;
8        else
9          if(1==iType)
10           x=y+10;
11         else
12           x=y+20;
13     }
14 }
```

路径测试方法主要包括4个步骤。

第一步，画出程序的控制流图。该程序的控制流图如图3.12所示。



(a) 程序流程图　　(b) 控制流图

图3.12　示例程序的程序流程图和控制流图

第二步，计算环形复杂度，并确定独立路径。

根据以上介绍的环形复杂度的计算方法，可计算出 $V(G)=E-N+2=11-9+2=4$。从程序的环形复杂度可导出程序基本路径集合中的独立路径数目，这是确定程序中每个可执行语句至少执行一次所必需的测试用例数目的上界，即独立路径数为4。因此，该程序的独立路径如下。

路径1：4-14。

路径2：4-6-7-14。

路径3：4-6-8-10-13-4-14。

路径4：4-6-8-11-13-4-14。

第三步，导出测试用例：根据环形复杂度和程序结构，设计测试用例的数据输入和预期结果，如表3.6所示。

第四步，执行测试。

表 3.6　测试用例表

测试用例名称	测试数据	预期结果	测试路径
T1	iRecordNum=0	x=0,y=0	路径 1
T2	iRecordNum=1 iType=0	x=2,y=0	路径 2
T3	iRecordNum=1 iType=1	x=10,y=0	路径 3
T4	iRecordNum=1 iType=2	x=20,y=0	路径 4

3.3.2　数据流测试

路径测试可以测试程序中所有的条件和语句块，但是，这样仍然不能检测出程序中所有的错误。基于数据流的测试主要关注程序中数据的定义和使用，可以作为对基于控制流测试的补充。早期的数据流测试主要关注以下 3 种情况：变量已定义，但从未使用；使用了未定义的变量；变量在使用之前被重复定义。

例如，有如下程序片段：

```
 1  int x,y;                    //定义 x,y
 2  float z;
 3  input(x,y);
 4  z=0;
 5  if(x!=0)
 6     z=z+y;
 7  else z=z-y;
 8  if(y!=0)
 9     z=z/x;
10  else z=z*x;
11  output(z);
```

在程序的第 4 行对变量 z 进行初始化，在后面的语句中根据变量 x 和 y 的情况修改 z 的值。这里可以采用两个测试用例，x、y 和 z 变量的取值分别是：T1(0,0,0.0)，T2(1,1,0.0)。执行 T1 的路径为先执行第 7 行修改 z 的值，再执行第 10 行；执行 T2 的路径为先执行第 6 行修改 z 的值，再执行第 9 行。同样，z 在第 6、7、9、10 行也被定义了，但是，程序的执行可能是：在第 7 行被定义后，在第 9 行被使用了，那么，这就可能出现“除零”的情况(如 x=0,y=1)。如果采用数据流测试就可以发现程序中类似的错误。

1. 定义/使用测试

首先要明确一个假设，数据流的假设还是和路径的假设一致：程序 P 的程序图(有向图)为单入口、单出口，并且不允许有从某个节点到其自身的边。

DEF(v,n)：定义节点，变量 v 在节点 n(语句片段)处定义，包括输入语句、赋值语句(等号左侧)、循环语句及过程调用都是定义节点的例子，如果执行这些语句，变量的值往往会发生变化。如：int x；x=y+z，这两条语句都是对 x 的定义。

USE(v,n)：使用节点，变量 v 在节点 n 处被使用，包括输出语句、赋值语句(等号右侧)、条件语句、循环控制语句和过程调用语句都是节点的使用语句，如果执行这类语句，值不会被改变。如：System.out.println(x)，该语句中使用了变量 x。

P-use：当一个变量被用在分支语句的条件表达式中(如 if 和 while 语句)，则称为变量的 P-use。其中的 P 表示“谓词”。如：if(z＜0){…}。其中的条件表达式是变量 z 的 P-use。

C-use：如果一个变量被用在赋值语句的表达式和输出语句中，被当作参数传递给调用函数，或被用在下标表达式中，则称为变量的 C-use。其中，C 表示“计算”。如：y＝x＋1；function(x)，这两条语句是变量 x 的 C-use。

DU-path：定义使用路径，开始节点是 DEF(v,n)，结束节点是 USE(v,n)的路径。

DC-path：定义清除路径，当开始节点和结束节点中间没有其他的定义节点时为清除路径。

对于上面的程序，给出其定义节点和使用节点，如表 3.7 所示。

表 3.7　示例程序中变量的定义节点和使用节点

变量	定义节点	使用节点
x	3	5,9,10
y	3	6,7,8
z	4,6,7,9,10	6,7,9,10,11

表 3.8、表 3.9 和表 3.10 分别列出了变量 x、y 和 z 的定义使用路径(DU-path)，并标出对应的路径是否为定义清除路径(DC-path)。

表 3.8　变量 x 的定义使用路径

DU-path(开始节点,结束节点)	是否为 DC-path
3,5	Y
3,9	Y
3,10	Y

表 3.9　变量 y 的定义使用路径

DU-path(开始节点,结束节点)	是否为 DC-path
3,6	Y
3,7	Y
3,8	Y

在对变量 x、y 和 z 的分析中发现，变量 x 和 y 的定义使用路径比较简单，而且其定义使用路径也都是定义清除路径；而变量 z 则比较复杂，定义使用路径中出现了部分非定义清除路径，如开始节点和结束节点分别为 4 和 9 的路径为 p1＝＜4,5,6,8,9＞和 p2＝＜4,5,7,8,9＞，变量 z 在节点 4 被定义之后，有可能在节点 6 或 7 被重新定义之后，在节点 9 再使用该变量，因此以节点 4 开始和以节点 9 结束的这两条路径 p1 和 p2 都是非定义清除路径。在定义使用测试中，应该重点关注非定义清除路径，这样就可以发现类似“除零”的错误。

表 3.10　变量 z 的定义使用路径

DU-path(开始节点,结束节点)	是否为 DC-path	DU-path(开始节点,结束节点)	是否为 DC-path
4,6	Y	7,10	Y
4,7	Y	7,11	N
4,9	N	9,6	不可行
4,10	N	9,7	不可行
4,11	N	9,9	Y
6,6	Y	9,10	不可行
6,7	不可行	9,11	Y
6,9	Y	10,6	不可行
6,10	Y	10,7	不可行
6,11	N	10,9	不可行
7,6	不可行	10,10	Y
7,7	Y	10,11	Y
7,9	Y		

2. 程序片测试

程序片也叫程序切片,是一种程序分析和理解技术。它通过把程序减少到只包含与某个特定计算相关的那些语句来分析程序。其概念最早是 1979 年由 Mark Weiser 提出来的。他观察到,程序员在调试过程中脑海中就有关于程序的某种抽象,人们在调试一个程序时总是从错误语句开始,并沿着依赖关系跟踪到它影响的程序部分。程序片的发展基本成熟,在理论和应用方面的研究均取得了可喜的进展,特别是在程序的调试、测试、分解和集成、软件维护、代码理解以及逆向工程等领域具有广泛的应用。

程序片是确定或影响某个变量在程序某个点上的取值的一组程序语句。典型的程序分片算法有 Weiser 的基于数据流方程的算法、无定型分片算法、Bergeretti 的基于信息流关系的算法、基于程序依赖图的图形可达性算法、基于波动图的算法、参数化程序分片算法、并行分片算法和面向对象的分层分片算法等。在众多程序分片算法中,Weiser 的基于数据流方程的算法是程序分片的基础,下面给出片的定义。

定义 1:给定一个程序 P 和 P 中的一个变量集合 V,变量集合 V 在语句 n 上的一个片记做 $S(V,n)$,是 P 中对 V 中的变量值做出贡献的所有语句集合。

定义 2:给定一个程序 P 和一个给出语句及语句片段编号的程序图 $G(P)$,以及 P 中的一个变量集合 V,变量集合 V 在语句片段 n 上的一个片记做 $S(V,n)$,是 P 中在 n 以前对 V 中的变量值作出贡献的所有语句片段编号的集合。

假设片 $S(V,n)$是一个变量的片,即 V 只有一个元素 v。如果语句片段 n 是 v 的一个定义节点,则 n 包含在该片中;如果语句片段 n 是 v 的一个使用节点,则 n 不包含在该片中。