第十一章 白盒测试

- 1 白盒测试概述
  - —白盒测试定义
  - —白盒测试的目标
  - —白盒测试的实施步骤
- 2 白盒测试方法
- 小结

1 白盒测试概述

——白盒测试定义

- *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;*
- *白盒测试是把测试对象看作一个打开的盒子，测试人员依据程序内部逻辑结构相关信息，设计或选择测试用例，对程序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。*

1 白盒测试概述

——白盒测试的目标

- *通过检查软件内部的逻辑结构，对软件中的逻辑路径进行覆盖测试；*
- *在程序不同地方设立检查点，检查程序的状态，以确定实际运行状态与预期状态是否一致。*
- ***软件人员使用白盒测试方法，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。

1 白盒测试概述

——白盒测试的实施步骤

- ***测试计划阶段:***
  - 根据需求说明书，制定测试进度。
- ***测试设计阶段:***
  - 依据程序设计说明书，按照一定规范化的方法进行软件结构划分和设计测试用例。
- ***测试执行阶段:***
  - 输入测试用例，得到测试结果。
- ***测试总结阶段:***
  - 对比测试的结果和代码的预期结果，分析错误原因，找到并解决错误。

第十一章 白盒测试

- 1 白盒测试概述
- 2 白盒测试方法
  - —逻辑覆盖法
  - —循环测试
  - —基本路径测试法
  - —程序插装
  - —程序变异
- 小结

2 白盒测试方法

- 代码检查法
- 静态结构分析法
- 静态质量度量法
- 逻辑覆盖法
- 循环测试
- 基本路径测试法
- 程序插装
- 程序变异

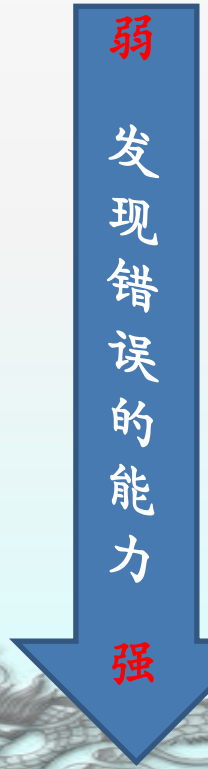
2 白盒测试方法

——逻辑覆盖法

逻辑覆盖包括六种覆盖标准：

语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖

- 语句覆盖：每条语句至少执行一次
- 判定覆盖：每个判定的每个分支至少执行一次
- 条件覆盖：每个判定的每个条件应取到各种可能的值
- 判定/条件覆盖：同时满足判定覆盖和条件覆盖
- 条件组合覆盖：每个判定中各条件的每一种组合至少出现一次
- 路径覆盖：程序中每一条可能的路径至少执行一次

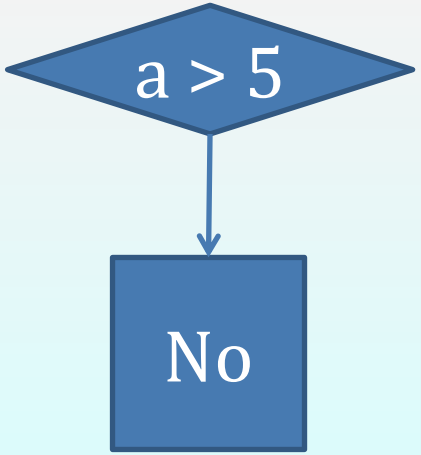


2 白盒测试方法

——逻辑覆盖法

语句覆盖

- ***语句覆盖测试准则***
  - 在测试时首先设计若干个测试用例，然后运行被测程序，使程序中的每个可执行语句至少执行一次。这里所谓“若干个”，自然是越少越好。



执行看得到的语句

A > 5

2 白盒测试方法

——逻辑覆盖法

语句覆盖

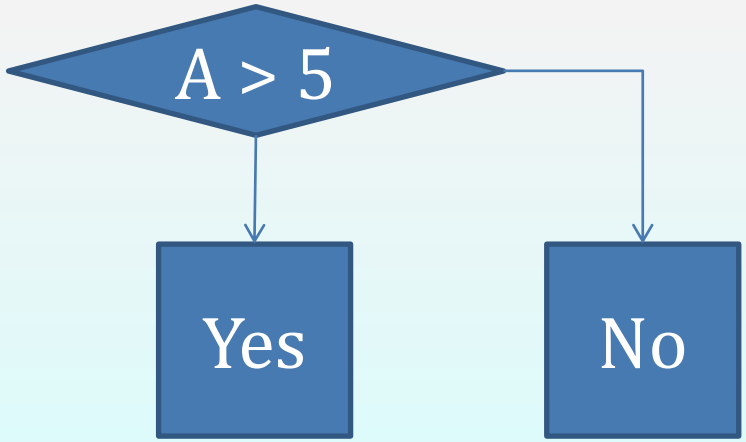
- ***语句覆盖测试准则***
  - 在测试时首先设计若干个测试用例，然后运行被测程序，使程序中的每个可执行语句至少执行一次。这里所谓“若干个”，自然是越少越好。
- ***语句覆盖测试的优点和缺点***
  - **优点：** 很直观地从代码中得到测试用例，无需细分每条判定表达式。
  - **缺点：** 对于隐藏的条件和可能到达的隐式分支是无法测试的。它只在乎运行一次，而不考虑其他情况

2 白盒测试方法

——逻辑覆盖法

判定覆盖

- ***判定覆盖测试准则***
  - 设计若干测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断的真假值均曾被满足。



每个判定真假各一次

A > 5　yes

A <= 5 no

2 白盒测试方法

——逻辑覆盖法

判定覆盖

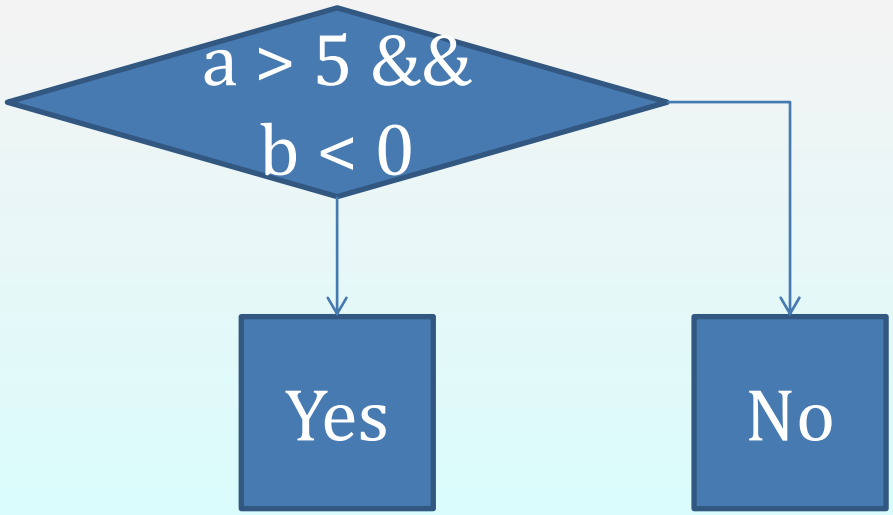
- ***判定覆盖测试准则***
  - 设计若干测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断的真假值均曾被满足。
- ***判定覆盖测试的优点和缺点***
  - **优点：** 分支覆盖是比语句覆盖更强的测试能力，比语句覆盖要多几乎一倍的测试路径。它无需细分每个判定就可以得到测试用例。
  - **缺点：** 往往大部分的判定语句是由多个逻辑条件组合而成，若仅仅判断其最终结果，而忽略每个条件的取值必然会遗漏部分的测试路径。

2 白盒测试方法

——逻辑覆盖法

条件覆盖

- ***条件覆盖测试准则***
  - 条件覆盖要求设计足够多的测试用例，使得判定中的每个条件获得各种可能的结果，即每个条件至少有一次为真值，有一次为假值。



每个条件的可能取值一次

a > 5 && b >= 0 no

a <= 5 && b < 0 no

2 白盒测试方法

——逻辑覆盖法

条件覆盖

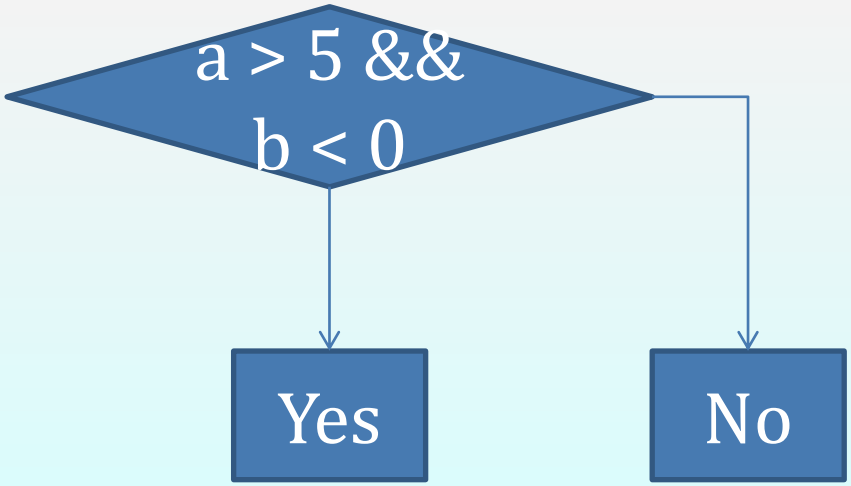
- ***条件覆盖测试准则***
  - 条件覆盖要求设计足够多的测试用例，使得判定中的每个条件获得各种可能的结果，即每个条件至少有一次为真值，有一次为假值。
- ***条件覆盖测试的优点和缺点***
  - 优点：显然条件覆盖比判定覆盖，增加了对符合判定情况的测试，增加了测试路径。
  - 缺点：要达到条件覆盖，需要足够多的测试用例，但条件覆盖并不能保证判定覆盖。条件覆盖只能保证每个条件至少有一次为真，而不考虑所有的判定结果。

2 白盒测试方法

——逻辑覆盖法

判定/条件覆盖

- *判定/条件覆盖测试准则*
  - 设计足够多的测试用例，使得判定中每个条件的所有可能结果至少出现一次，每个判定本身所有可能结果也至少出现一次。



每个判定真假各一次
每个判定中的条件各取一次
a > 5 && b < 0 yes
a <= 5 && b >= 0 no

2 白盒测试方法

——逻辑覆盖法

判定/条件覆盖

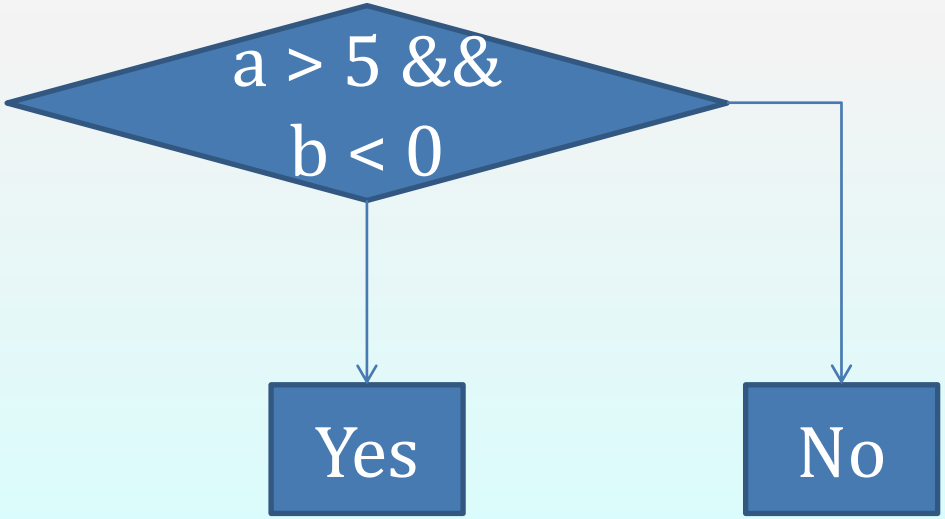
- ***判定/条件覆盖测试准则***
  - 设计足够多的测试用例，使得判定中每个条件的所有可能结果至少出现一次，每个判定本身所有可能结果也至少出现一次。
- ***判定/条件覆盖测试的优点和缺点***
  - **优点：** 满足判定覆盖准则和条件覆盖准则，弥补了二者的不足。
  - **缺点：** 未考虑条件的组合情况

2 白盒测试方法

——逻辑覆盖法

条件组合覆盖

- ***条件组合覆盖测试准则***
  - 要求设计足够多的测试用例，使得每个判定中条件结果的所有可能组合至少出现一次。



判定中所有可能的条件组合

a > 5 和 a <= 5 有两种

b < 0 和 b >= 0 有两种

共有2 * 2 = 4种

a > 5 && b < 0

a > 5 && b >= 0

a <= 5 && b < 0

a <= 5 && b >= 0

2 白盒测试方法

——逻辑覆盖法

条件组合覆盖

- ***条件组合覆盖测试准则***
  - 要求设计足够多的测试用例，使得每个判定中条件结果的所有可能组合至少出现一次。
- ***条件组合覆盖测试的优点和缺点***
  - **优点：** 满足判定覆盖、条件覆盖和判定/条件覆盖准则。使得判定中每个条件的所有可能结果至少出现一次，每个判定本身的所有可能结果也至少出现一次。并且每个条件都显示能单独影响判定结果。
  - **缺点：** 线性地增加了测试用例的数量

2 白盒测试方法

——逻辑覆盖法

路径覆盖

- ***路径覆盖测试准则***
  - 设计足够的测试用例，覆盖程序中所有可能的路径。

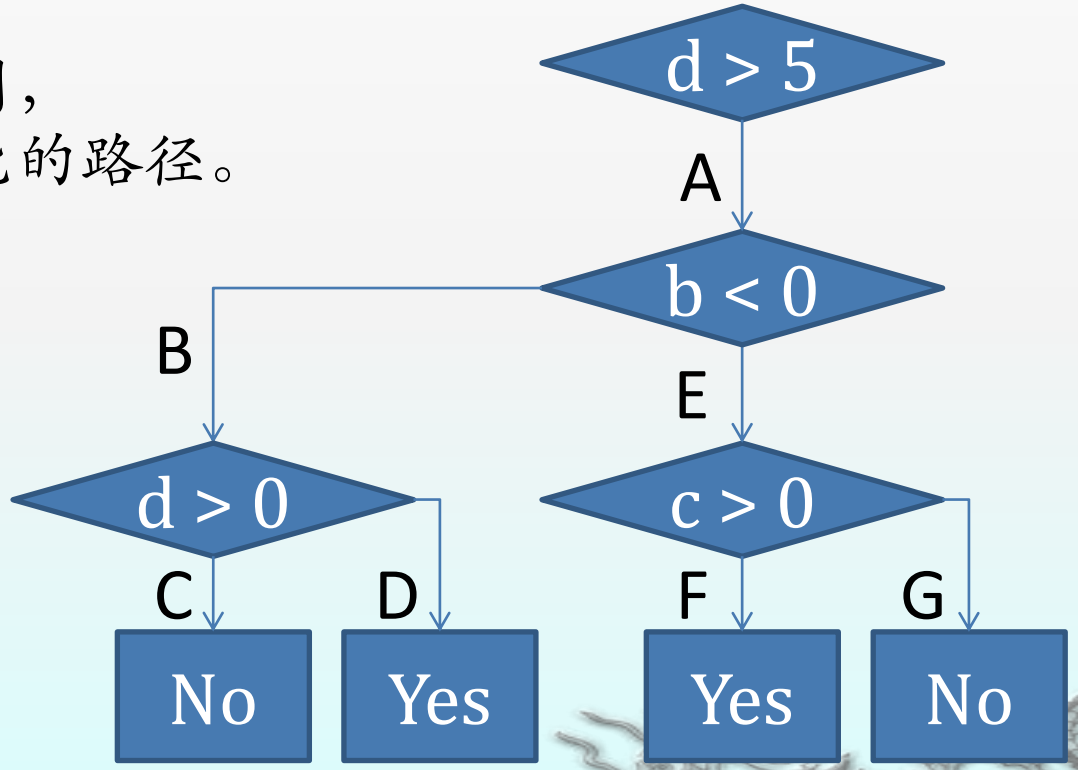
走完所有路径

A -> B -> C

A -> B -> D

A -> E -> F

A -> E -> G



2 白盒测试方法

——逻辑覆盖法

路径覆盖

- ***路径覆盖测试准则***
  - 设计足够的测试用例，覆盖程序中所有可能的路径。
- ***路径覆盖测试的优点和缺点***
  - **优点：** 可以对程序进行彻底的测试，比前面五种的覆盖面都广。
  - **缺点：** 由于需要对所有可能的路径进行测试，需要设计大量、复杂的测试用例，使得工作量呈指数级增长。而在有些情况下，一些执行路径是不可能被执行的。如：If （!A） B++;
    If （!A） D--;

2 白盒测试方法

——逻辑覆盖法

几种常用逻辑覆盖的比较

- ***语句覆盖：***
  - 在测试时，首先设计若干个测试用例，然后运行被测程序，使程序中的每个语句至少执行一次；
  - 语句覆盖的测试可能给人们一种心理的满足，以为每个语句都经历过，似乎可以放心了；
  - 实际上，语句覆盖在测试被测试程序中，除去对检查不可执行语句有一定作用外，并没有排除被测程序包含错误的风险；
  - 必须看到，被测程序并非语句的无序堆积，语句之间的确存在着许多有机的联系。

2 白盒测试方法

——逻辑覆盖法

几种常用逻辑覆盖的比较

- *判定覆盖：*
  - 设计若干测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断的真假值均曾被满足；
  - 同样，只作到判定覆盖仍无法确定判断内部条件的错误。
- *条件覆盖：*
  - 设计若干测试用例，执行被测程序以后，要使每个判断中每个条件的可能取值至少满足一次；
  - 但覆盖了条件的测试用例不一定覆盖了分支。

2 白盒测试方法

——逻辑覆盖法

几种常用逻辑覆盖的比较

- ***判定—条件覆盖：***
  - 判定—条件覆盖要求设计足够的测试用例，使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次；
  - 不过忽略了路径覆盖的问题，而路径能否全面覆盖在软件测试中是个重要问题，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 白盒测试方法

——逻辑覆盖法

几种常用逻辑覆盖的比较

- ***路径覆盖：***
  - 设计足够多的测试用例，要求覆盖程序中所有可能的路径；
  - 许多情况路径数是个庞大的数字，要全部覆盖是无法实现的；
  - 即使都覆盖到了，仍然不能保证被测程序的正确性。

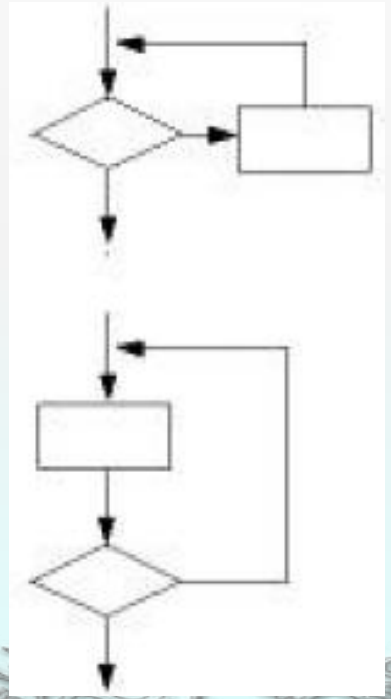
2 白盒测试方法

——循环测试

循环语句分为以下4种：

简单循环、嵌套循环、连锁循环和非结构循环

- ***简单循环：***
  - 测试用例设计需要考虑以下方面：
    - *零次循环：从循环入口直接跳到循环出口。*
    - *一次循环：查找循环初始值方面的错误。*
    - *二次循环：检查在多次循环时才能暴露的错误。*
    - *m次循环：此时的m＜n，也是检查在多次循环时才能暴露的错误；*
    - *循环n-1次、n次、n+1次。*
  - 可以把循环次数处于范围边界附近和处于范围内部的情况都测试到



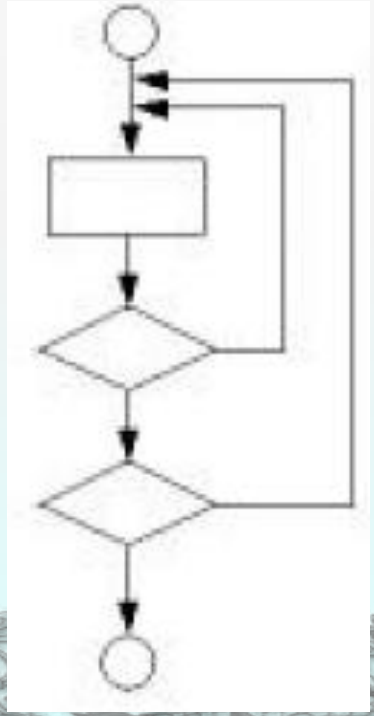
2 白盒测试方法

——循环测试

循环语句分为以下4种：

简单循环、嵌套循环、连锁循环和非结构循环

- ***嵌套循环：***
  - 减少测试数目的测试方法：
    - *除最内层循环外，从最内层循环开始，置所有其他层的循环为最小值；*
    - *对最内层循环做简单循环的全部测试。测试时保持所有外层循环的循环变量为最小值。另外，对越界值和非法值做类似的测试。*
    - *逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环的循环变量取最小值，所有其它嵌套内层循环的循环变量取“典型”值。*
    - *反复进行，直到所有各层循环测试完毕。*
    - *完全跳过循环；仅循环一次；*



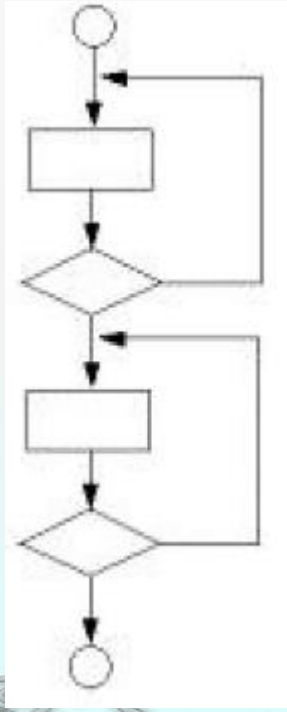
2 白盒测试方法

——循环测试

循环语句分为以下4种：

简单循环、嵌套循环、连锁循环和非结构循环

- ***连锁循环：***
  - 各个循环互相独立：
    - *可以用与简单循环相同的方法进行测试*
  - 各个循环不是互相独立：
    - *使用测试嵌套循环的办法来处理*



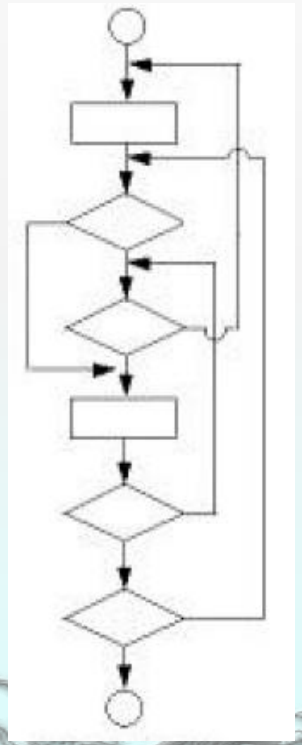
2 白盒测试方法

——循环测试

循环语句分为以下4种：

简单循环、嵌套循环、连锁循环和非结构循环

- ***非结构循环：***
  - 应该使用结构化程序设计方法重新设计测试用例



2 白盒测试方法

——基本路径测试

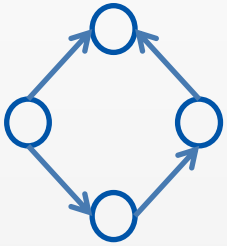
- *如果把覆盖的路径数压缩到一定限度内，例如，程序中的循环体只执行零次和一次，就成为基本路径测试。*
- *它是在**程序控制流图**的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。*
- *设计出的测试用例要保证在测试中，程序的每一个可执行语句至少要执行一次。*

2 白盒测试方法

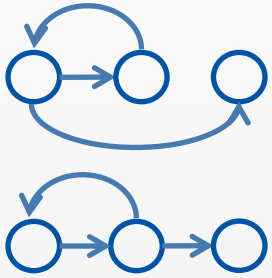
——基本路径测试

- *程序控制流图*

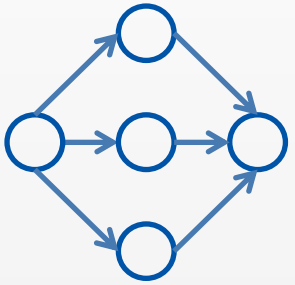
顺序结构



if选择结构



while循环结构
until循环结构



case多分支结构

- 符号○称为控制流图的一个结点，一组顺序处理框可以映射为一个单一的结点。
- 控制流图中的箭头称为边，它表示了控制流的方向；
- 在选择或多分支结构中分支的汇聚处，即使没有执行语句也应该有一个汇聚结点。
- 边和结点圈定的区域叫做区域，当对区域计数时，图形外的区域也应记为一个区域。

2 白盒测试方法

——基本路径测试

- ***程序控制流图***
  - 在将程序流图简化成控制流图时，应注意：
    - *如果判定中的条件表达式是**复合条件**时，即条件表达式是由一个或多个逻辑运算符(OR，AND，NAND，NOR)连接的逻辑表达式，则需要改复合条件的判定为一系列只有单个条件的嵌套的判定。*
    - *进行程序的基本路径测试时，程序的**环路复杂性**给出了程序基本路径集合中的**独立路径条数**，这是确保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。所谓独立路径，是指包括一组以前没有处理的语句或条件的一条路径。*

2 白盒测试方法

——基本路径测试

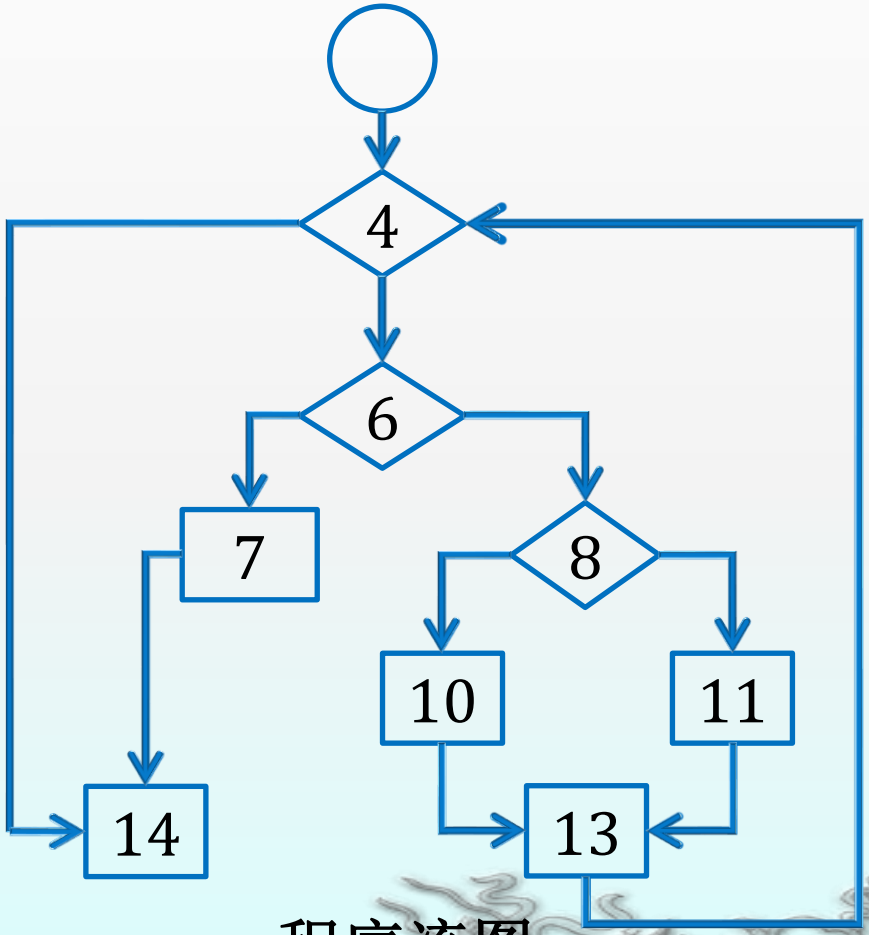
- ***程序结构的要求***
  - 程序结构有以下4点基本要求，写出的程序代码中**不应**包含：
    - *转向并不存在的标号。*
    - *没有用的语句标号。*
    - *从程序入口进入后无法到达的语句；*
    - *不能到达退出程序的语句。*
  - 在编写程序代码时稍加注意，做到这几点也是比较容易的。另外对这四种情况的检测也可以通过编译器和程序分析工具来实现。

2 白盒测试方法

——基本路径测试

例子

```
   void  Sort(int iR,int  iT)
1  {
2     int x=0;
3     int y=0;
4     while (iR-- > 0)
5        {
6           if(0= =iT)
7              { x=y+2;break;}
8           else
9              if(1= =iT)
10                x=y+10;
11             else
12                x=y+20;
13       }
14 }
```

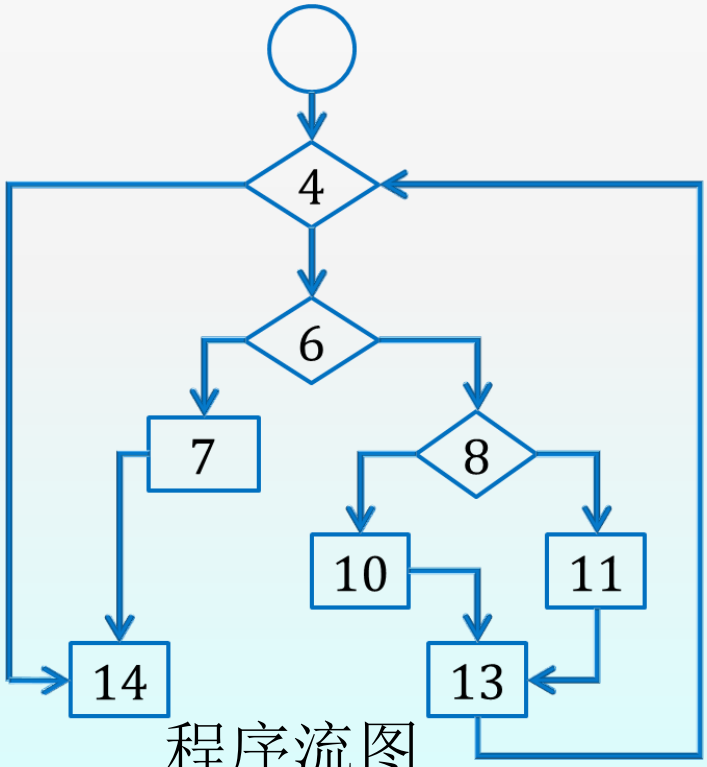


程序流图

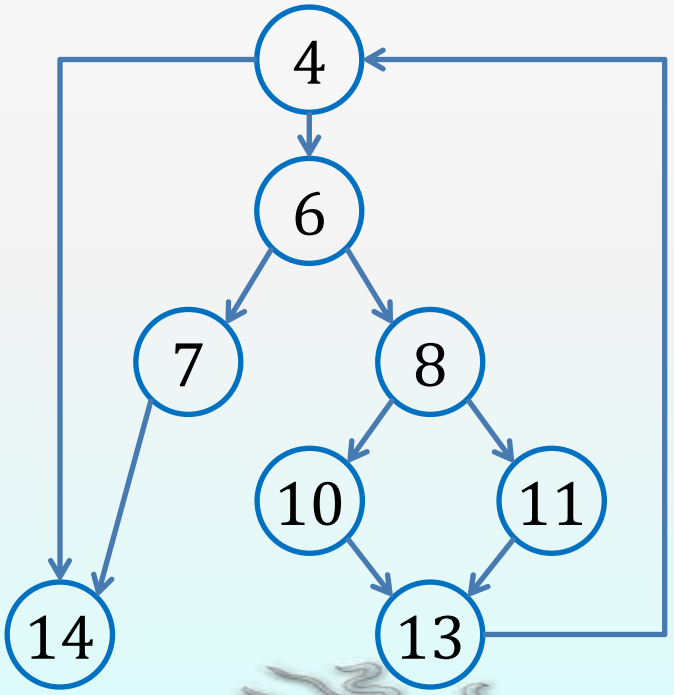
2 白盒测试方法

——基本路径测试

- ***基本路径测试步骤***
  - 第一步：画出控制流图



程序流图



控制流图

2 白盒测试方法

——基本路径测试

- ***基本路径测试步骤***
  - 第二步：计算圈复杂度
    - *一种为程序逻辑复杂性提供定量测度的软件度量。*
    - *该度量用于计算程序的基本的独立路径数目，为确保所有语句至少执行一次的测试数量的上界。*
    - *独立路径必须包含一条在定义之前不曾用到的边。*
    - *三种方法计算圈复杂度：*

① *流图中区域的数量对应于环型的复杂性;*

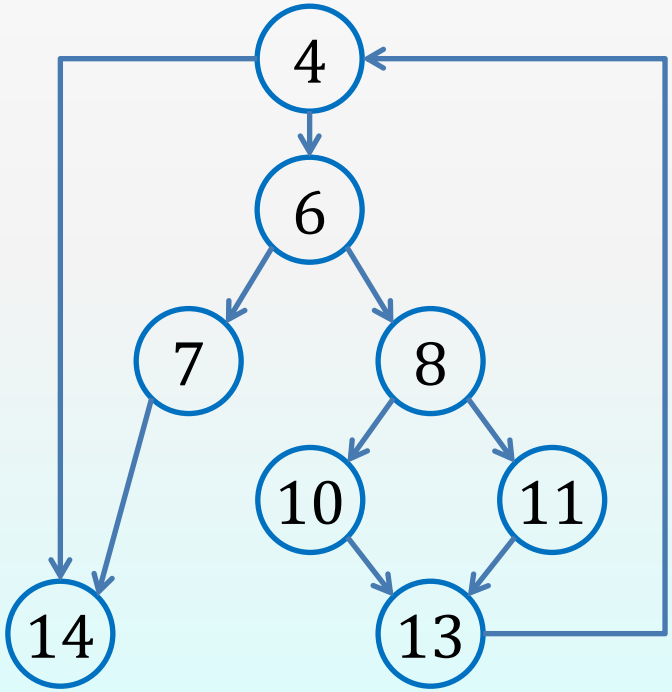
② *给定流图G的圈复杂度V(G)，定义为V(G)=E-N+2，E是流图中边的数量，N是流图中结点的数量;*

③ *给定流图G的圈复杂度V(G)，定义为V(G)=P+1，P是流图G中判定结点的数量。* 至少有两个分支出来的叫做判定节点

2 白盒测试方法

——基本路径测试

- ***基本路径测试步骤***
  - 第二步：计算圈复杂度



控制流图

对应左图中的圈复杂度，计算如下：

- 流图中有四个区域；
- V(G)=10条边-8结点+2=4；
- V(G)=3个判定结点+1=4。

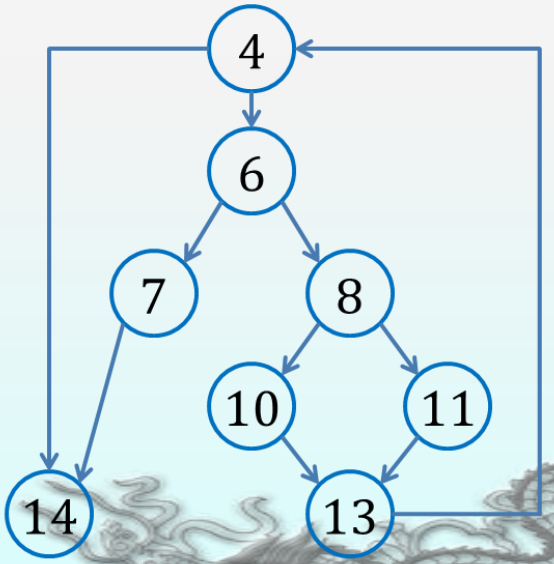
2 白盒测试方法

——基本路径测试

- ***基本路径测试步骤***
  - 第三步：导出测试用例
    - *根据上面的计算方法，可得出四个独立的路径。*
      (一条独立路径是指，和其他的独立路径相比，至少引入一个新处理语句或一个新判断的程序通路。V(G)值正好等于该程序的独立路径的条数。)
    - *确定路径*

① *路径1：4-14*
② *路径2：4-6-7-14*
③ *路径3：4-6-8-10-13-4-14*
④ *路径4：4-6-8-11-13-4-14*

    - *根据上面的独立路径，去设计输入数据，使程序分别执行到上面四条路径。*



2 白盒测试方法

——基本路径测试

- ***基本路径测试步骤***
  - 第四步：准备测试用例
    - *为了确保基本路径集中的**每**一条路径的执行，根据判断结点给出的条件，选择适当的数据以保证**某**一条路径可以被测试到*
    - *满足上面例子基本路径集的测试用例是：*

路径	输入数据	预期输出
路径1：4-14	iR<=0	X=0 Y=0
路径2：4-6-7-14	iR=1 iT=0	X=2 Y=0
路径3：4-6-8-10-13-4-14	iR=1 iT=1	X=10 Y=0
路径4：4-6-8-11-13-4-14	iR=1 iT=2	X=20 Y=0

2 白盒测试方法

——程序插装

基本的测试手段，在软件测试中有着广泛的应用

- *保持被测试程序原有逻辑完整性基础上在程序中插入一些探针，通过探针的执行并抛出程序的运行特征数据。基于这些特征数据分析可以获得程序的控制流及数据流信息，进而得到逻辑覆盖等动态信息；* eg print中间结果看对不对
- *这种方式相当于在运行程序以后，一方面检验测试结果数据，另一方面借助插入语句给出的信息了解程序的动态执行情况。*

2 白盒测试方法

——程序变异测试

通过检查测试用例集合是否可以发现特定的错误，来评判测试用例集合完备性的一种软件测试策略

- ***变异测试的假定：***
  - 如果一个测试用例集合可以检测出程序中所有的简单错误，那么这个测试用例集合就可以检测出所有错误（基于单缺陷原则）
- ***变异测试目的：***
  - 帮助测试者发现有效地测试，或者定位测试数据的弱点，或者是在执行中很少（或从不）使用的代码的弱点。

2 白盒测试方法

——程序变异测试

- ***变异测试方法：***
  - 变异测试中通过对源程序进行细小的修改来引入简单错误，生成程序的错误版本，这些错误版本被称作变异体；
  - 这样的修改过程被称作变异，而修改操作本身被称作变异运算符，每个变异运算符对应于一类简单错误。
  - 变异测试中，测试用例集合中的每一个测试用例运行在这些变异体上，如果变异体在一个测试用例中的输出与源程序输出不相同，则表示这个测试用例可以检测出该变异体所代表的简单错误。
  - 一个完备的测试用例集合应该可以检测出所有的变异体。

2 白盒测试方法

——程序变异测试

- ***变异测试方法的优缺点：***
  - **优点：** 变异测试是一个功能强大的软件测试策略，它生成的测试用例集合在理论上是完备的。
  - **缺点：** 变异测试系统相当耗费时间和资源。

小结

- 本章主要讲解了白盒测试的基本概念和技术，包括白盒测试的基本概念、分类、白盒测试中的边界值技术、语句覆盖测试、分支覆盖测试、条件覆盖测试、分支-条件覆盖测试、条件组合覆盖测试、路径覆盖测试。
- 白盒测试允许观察“盒子”内部，不像黑盒测试那样把系统理解为一个“内部不可见的盒子”，不需要明白内部结构。
- 为了完整的测试一个软件，这两种测试都是不可或缺的。一个产品在其概念分析阶段直到最后交付给用户期间往往要经过各种静态的、动态的、白盒的和黑盒的测试。