SOFTWARE TESTING

Chapter 4

White Box Testing

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

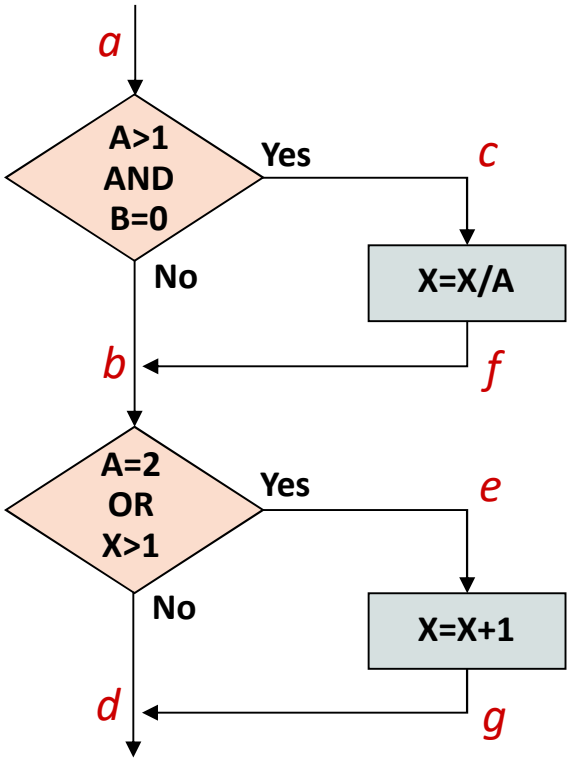
中山大學
SUN YAT-SEN UNIVERSITY

OUTLINE

路径测试

- 设计足够多的测试用例，覆盖被测试对象中的所有可能执行路径。
- 例4：程序流程图如右图所示，设计路径测试用例。

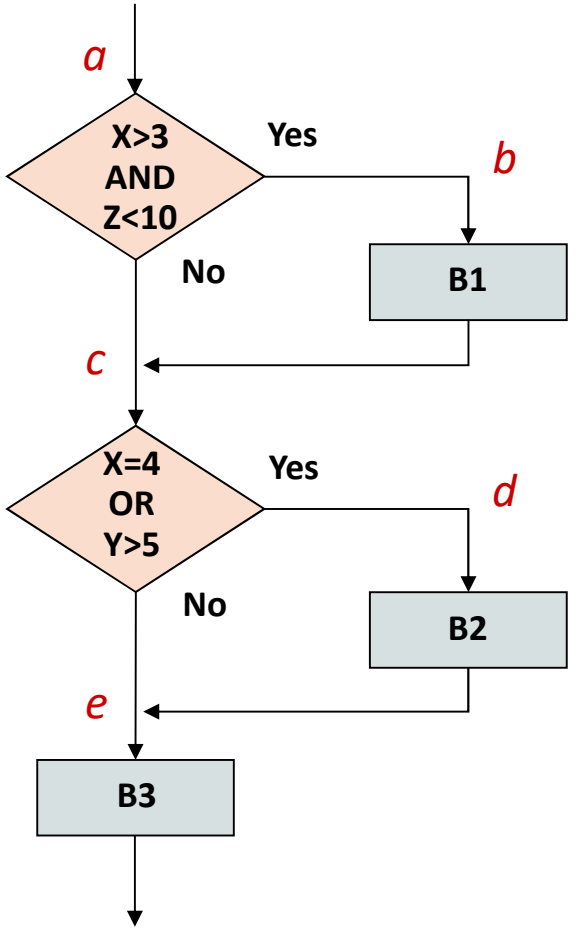
测试用例	通过路径标志
A=2, B=0, X=3	*acfbegd*
A=1, B=0, X=1	*abd*
A=2, B=1, X=1	*abegd*
A=3, B=0, X=1	*acfbd*



路径测试

- 例5：程序流程图如右图所示。
  - 条件定义
    - T1: X>3, F1: X≤3
    - T2: Z<10, F2: Z≥10
    - T3: X=4, F3: X<>4
    - T4: Y>5, F4: Y≤5

测试用例	通过路径	覆盖条件
x=4, y=6, z=5	*abcde*	T1, T2, T3, T4
x=4, y=5, z=15	*acde*	T1, F2, T3, F4
x=2, y=5, z=15	*ace*	F1, F2, F3, T4
x=5, y=5, z=5	*abce*	T1, T2, F3, F4



基本路径测试

- 在实际的应用中，一个不太复杂的程序，其执行路径数目也可能是一个庞大的数字，要想在测试中覆盖所有路径是不现实的；在不能做到覆盖程序所有路径的前提下，如果能够对程序的每一个独立路径进行测试，那么可以认为程序中的每个语句都已经检验过或覆盖到。
- 基本路径测试在程序控制流图的基础上，通过分析 *McCabe* 环路复杂性，导出基本可执行路径集合，据此设计测试用例。设计出的测试用例确保在测试中程序的每一个可执行语句至少执行一次。

基本路径测试

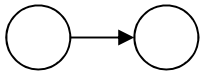
- 前提条件
  - 测试人员了解被测试软件，明确其逻辑结构。
- 测试过程
  - 针对程序逻辑结构设计加载测试用例，驱动程序执行，对程序路径进行测试。
- 测试结果
  - 分析实际的测试结果与预期的结果是否一致。

基本路径测试的设计步骤

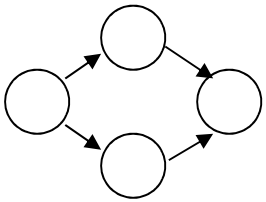
- 根据过程设计 (通常是程序流程图 Program Flow Chart) 画出程序的控制流图 Control Flow Graph。
- 计算程序控制流图的 *McCabe* 环路复杂度。
  - 环路复杂度是 *McCabe* 复杂性度量之一，从其中导出的程序基本路径集合中的独立路径条数，是确定程序中每个可执行语句至少执行一次所必须的最少测试用例数目。
- 确定一个线性独立路径 (数量由环路复杂度确定) 的基本集合 (即基本路径集合)。
- 准备测试用例：设计用例数据输入，确保基本路径集合中的每一条路径都得到执行。
- 应用测试用例，结合预期结果进行结果分析。

程序控制流图

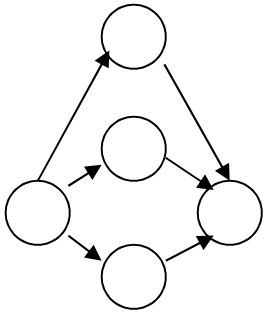
- 程序控制流图 (CFG, Control Flow Graph, 简称控制流图) 不同于程序流程图 (Program Flow Chart)。
- 控制流图使用下面的结构符号描述逻辑控制流，每一种程序结构化元素有一个相应的流图结构。



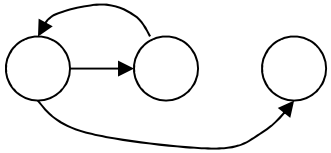
顺序结构



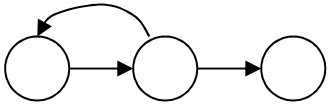
if 结构



case 结构



while 结构



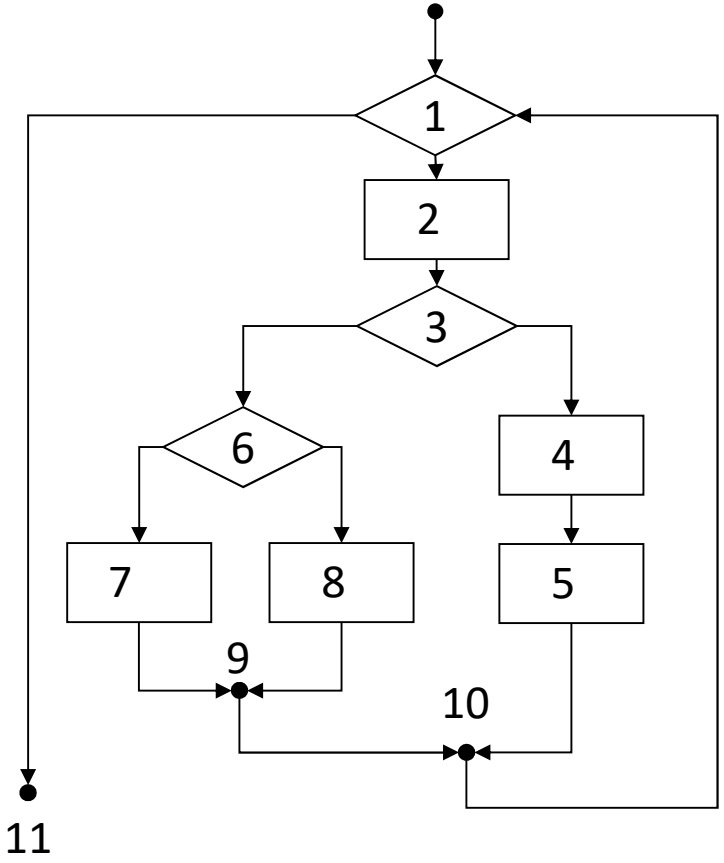
until 结构

程序控制流图

- 控制流图是一个有向图对象，只有2种基本符号：
  - 图中的一个圆圈称为流图的结点，代表一条或多条语句。
  - 图中的箭头称为边或连接，代表控制的流向。
- 任何过程设计都要被翻译成控制流图。
  - 在将程序流程图转化成程序控制流图时，在选择或多分支结构中，分支的汇聚处应有一个汇聚结点。

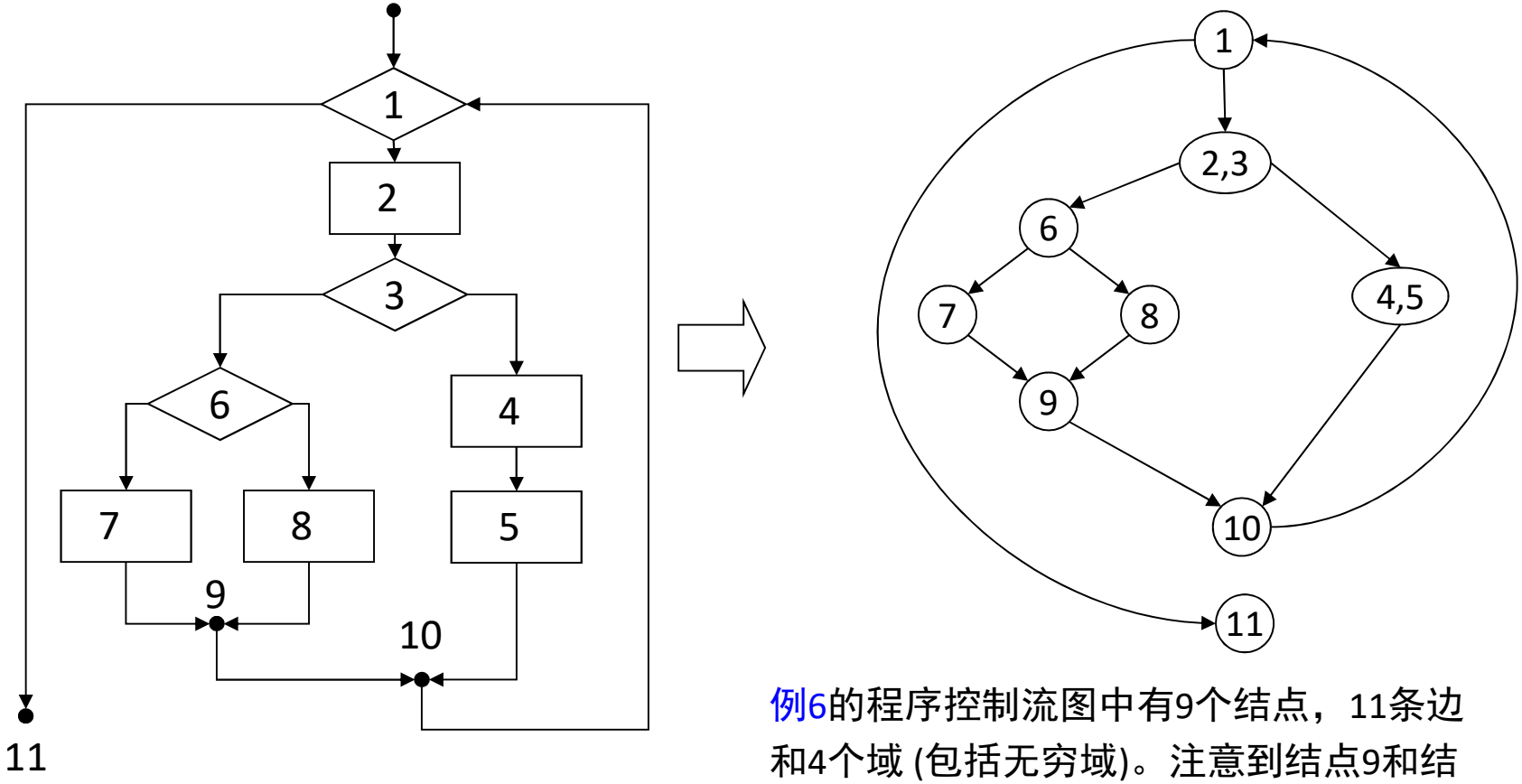
程序控制流图

- 例6：将下面的程序流程图转换成为等价的程序控制流图。



程序控制流图

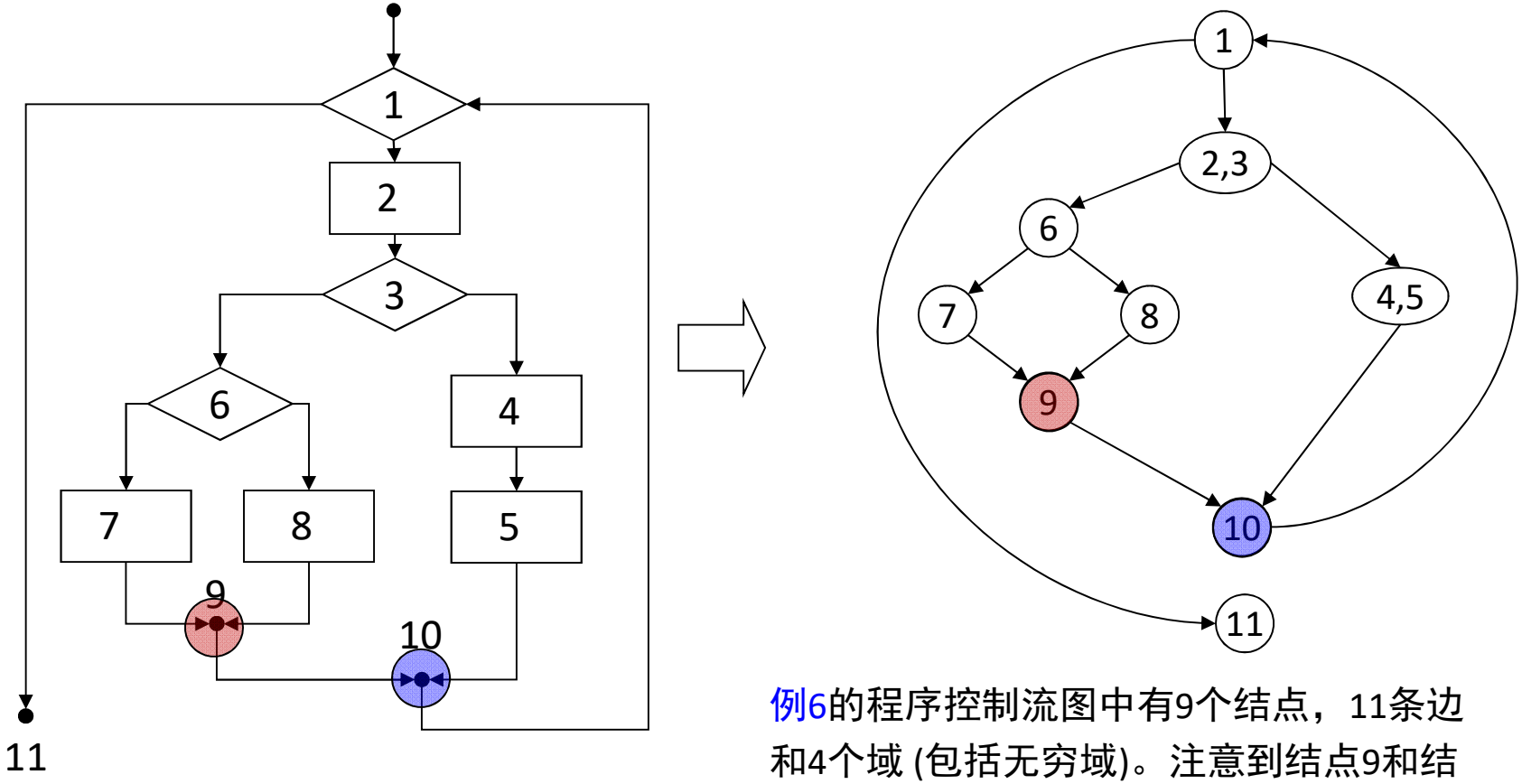
- 例6：将下面的程序流程图转换成为等价的程序控制流图。



例6的程序控制流图中有9个结点，11条边和4个域 (包括无穷域)。注意到结点9和结点10是两个汇聚结点。

程序控制流图

- 例6：将下面的程序流程图转换成为等价的程序控制流图。



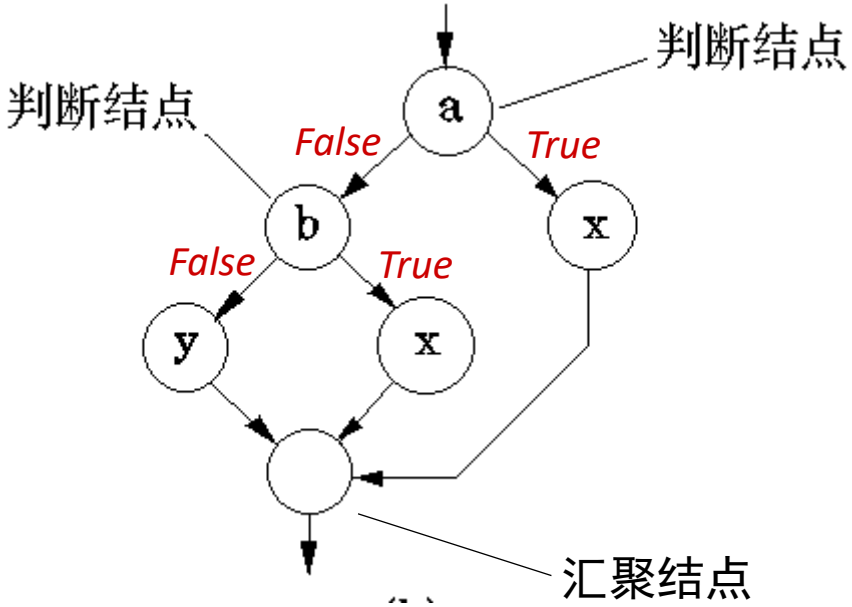
例6的程序控制流图中有9个结点，11条边和4个域 (包括无穷域)。注意到结点9和结点10是两个汇聚结点。

程序控制流图

- 如果判断中的条件表达式是由一个或多个逻辑运算符连接 (如OR, AND, NAND, NOR) 的复合条件表达式，则需要等价转换成一系列只含有单条件判定的嵌套的条件表达式。
  - 例7：下列语句

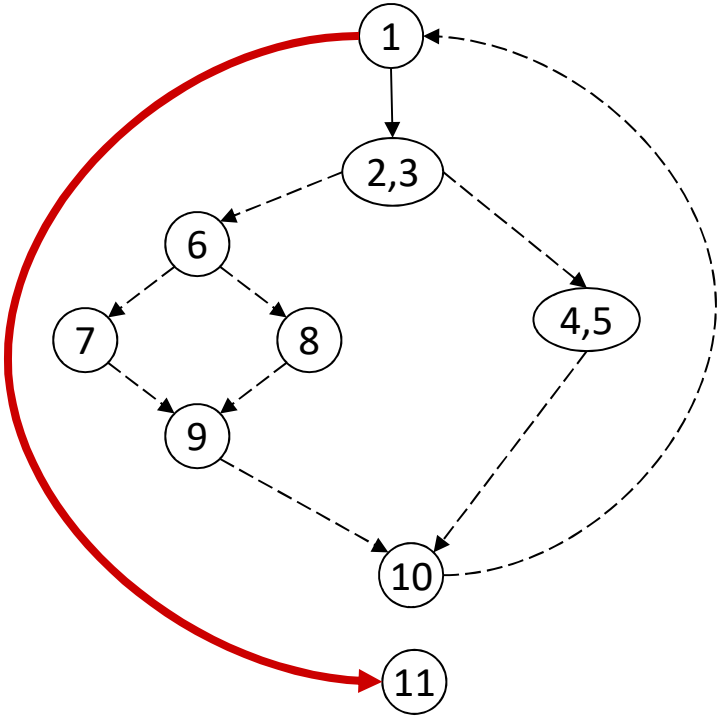
```
if (a or b)
    x;
else
    y;
```

  - 对应的逻辑如右图：

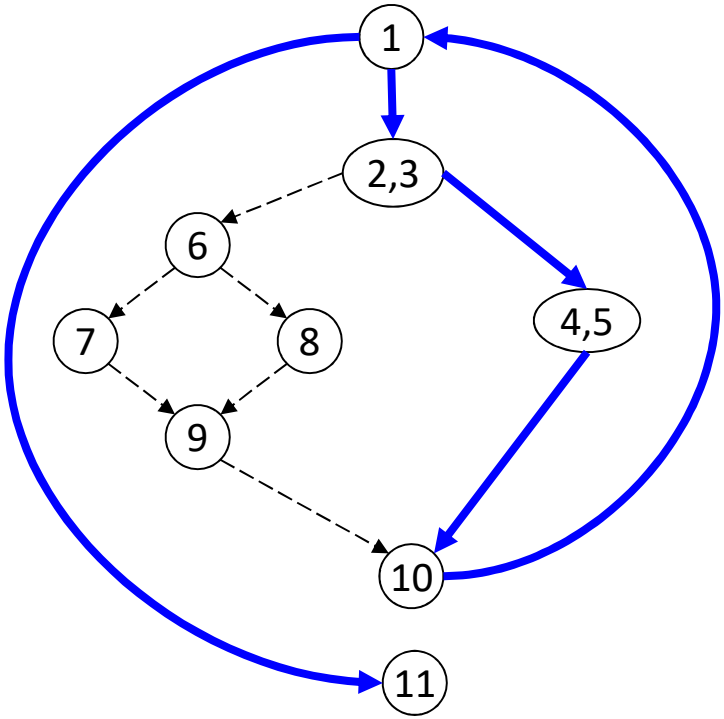


程序控制流图

- 独立路径：至少沿一条新的边移动的路径。
  - 例6程序控制流图中的4条独立路径。对路径1-4的遍历使得程序中的所有语句至少被执行了一次。



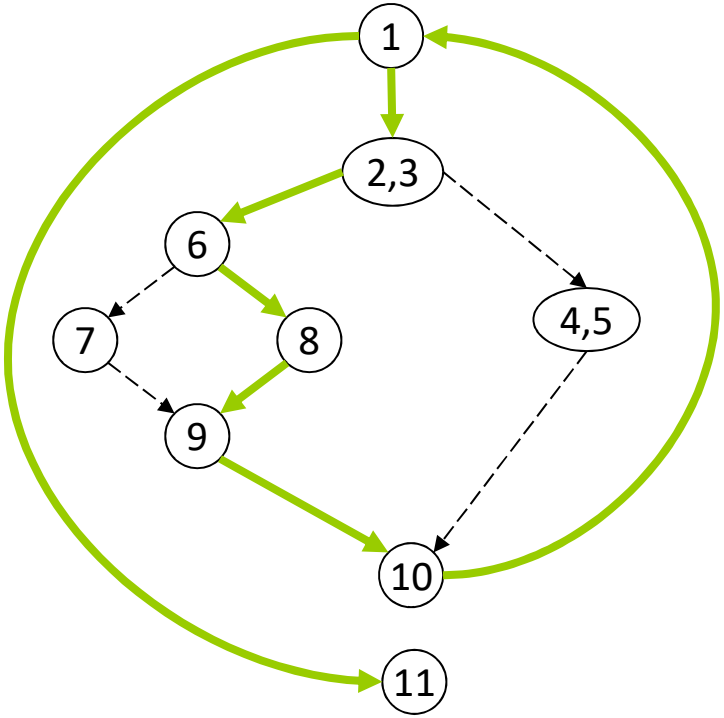
路径1：1-11



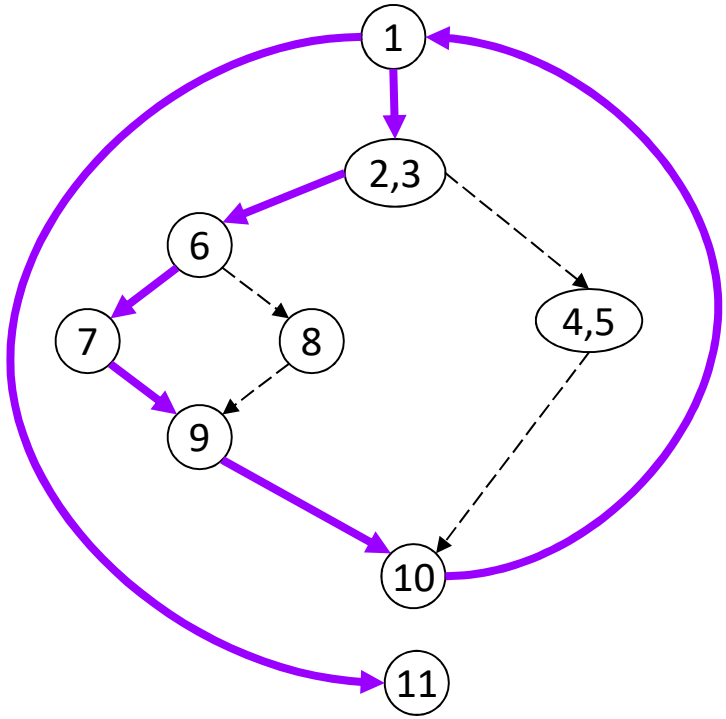
路径2：1-2,3-4,5-10-1-11

程序控制流图

- 独立路径：至少沿一条新的边移动的路径。
  - 例6程序控制流图中的4条独立路径。对路径1-4的遍历使得程序中的所有语句至少被执行了一次。



路径3：1-2,3-6-8-9-10-1-11



路径4：1-2,3-6-7-9-10-1-11

- 基本路径测试的具体步骤
  - 第一步：画出控制流图
    - 程序流程图用来描述程序控制结构。将程序流程图映射到一个相应的程序控制流图。控制流图中的圆称为流图的结点，代表一个或多个语句。流程图一个处理方框序列或一个菱形判决框都可被映射为流图的一个结点 (简化起见假设程序流程图的菱形判决框中不包含复合条件)。流图中的箭头称为边或连接，代表控制流向。流图中的一条边必须终止于一个结点。由边和结点限定的封闭范围称为区域。计算区域时应包括外部域。

- 基本路径测试的具体步骤
  - 第一步：画出控制流图
    - 例8：有 C 函数源代码如下，画出其程序控制流图。

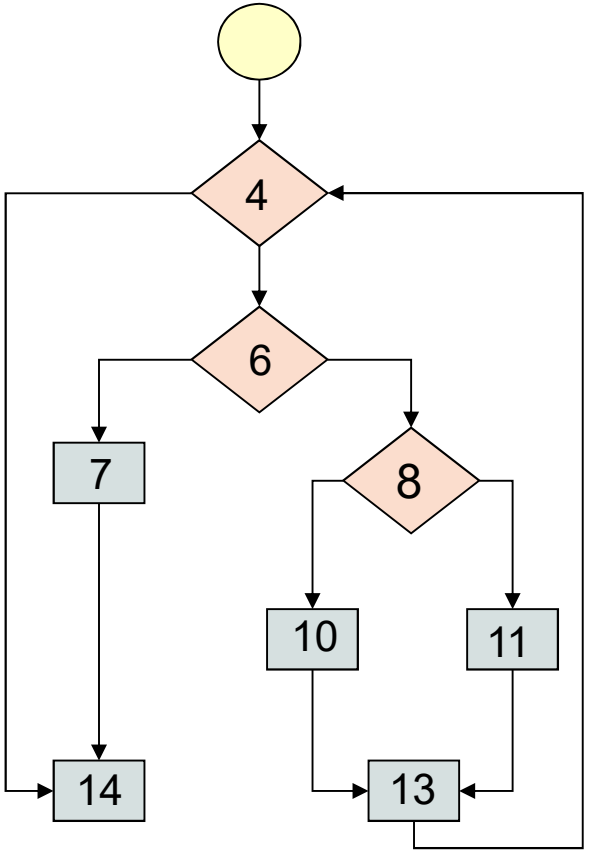
```
      void  Sort( int iRecordNum, int iType )
1.    {
2.       int x = 0;
3.       int y = 0;
4.       while ( iRecordNum - - >= 0 )
5.       {
6.          if ( 0 == iType )
7.             { x = y + 2; break; }
8.          else
9.             if ( 1 == iType )
10.               x = y + 10;
11.  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

思考：例8 的 C 源代码有没有不符合静态规范的地方？

基本路径测试的具体步骤

- 第一步：画出控制流图
  - 例8 的程序流程图如右图。

```
     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; break; }
8.         else
9.            if ( 1 == iType )
10.              x = y + 10;
11.           else
12.              x = y + 20;
13.     }
14.  }
```

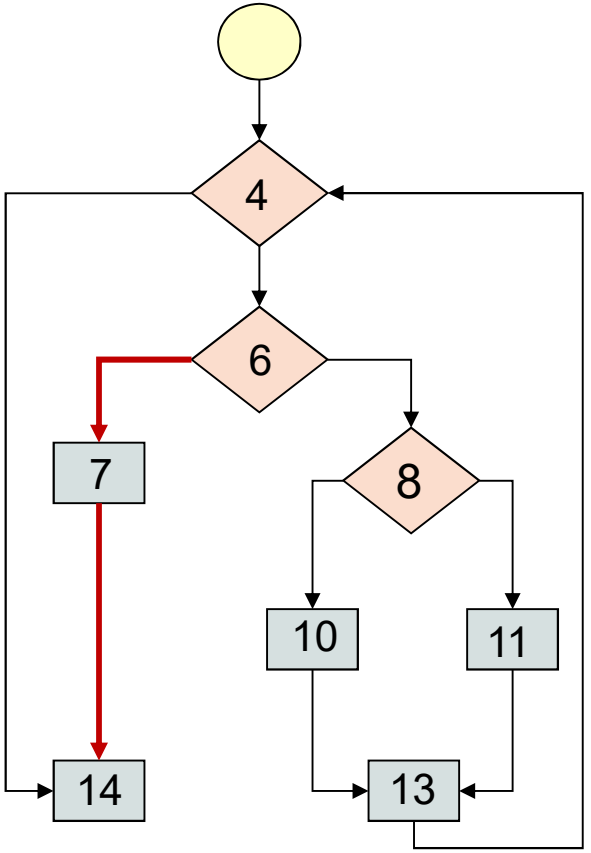


例8 的程序流程图

基本路径测试的具体步骤

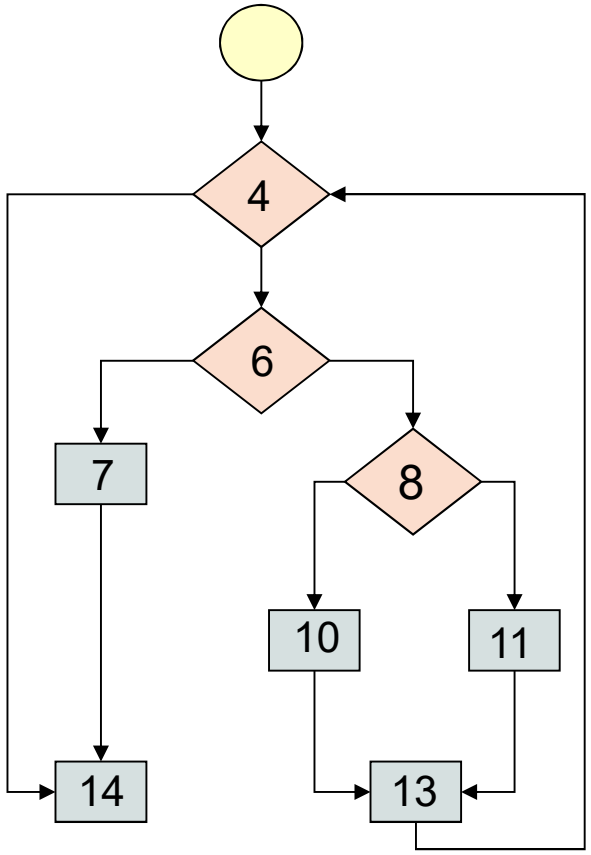
- 第一步：画出控制流图
  - 例8 的程序流程图如右图。

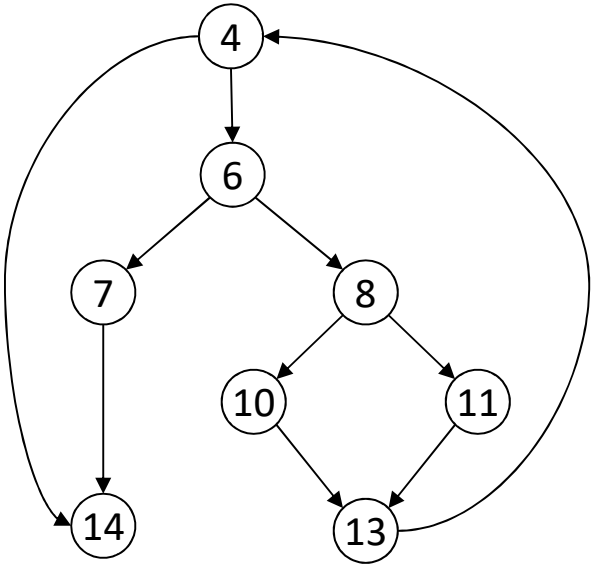
```
    void  Sort( int iRecordNum, int iType )
1.  {
2.      int x = 0;
3.      int y = 0;
4.      while ( iRecordNum - - >= 0 )
5.      {
6.          if ( 0 == iType )
7.              { x = y + 2; break; }  //直接跳出循环
8.          else
9.              if ( 1 == iType )
10.                 x = y + 10;
11.             else
12.                 x = y + 20;
13.     }
14. }
```



例8 的程序流程图

- 基本路径测试的具体步骤
  - 第一步：画出控制流图
    - 例8 的程序控制流图如右图。





例8 的程序控制流图

- 基本路径测试的具体步骤
  - 第二步：计算 *McCabe* 环路复杂度
    - *McCabe* 环路复杂度为程序逻辑复杂性提供定量测度。该度量用于计算程序的基本独立路径数目，也即是确保所有语句至少执行一次的起码测试数量。
    - 有以下三种方法计算环路复杂度：
      - 给定流图 G 的环路复杂度 V(G)，定义为

        V(G) = m – n + 2.

        m 是流图中边的数量，n 是流图中结点的数量；
      - 平面流图中区域的数量对应于环路复杂度；
      - 给定流图 G 的环路复杂度 V(G)，定义为

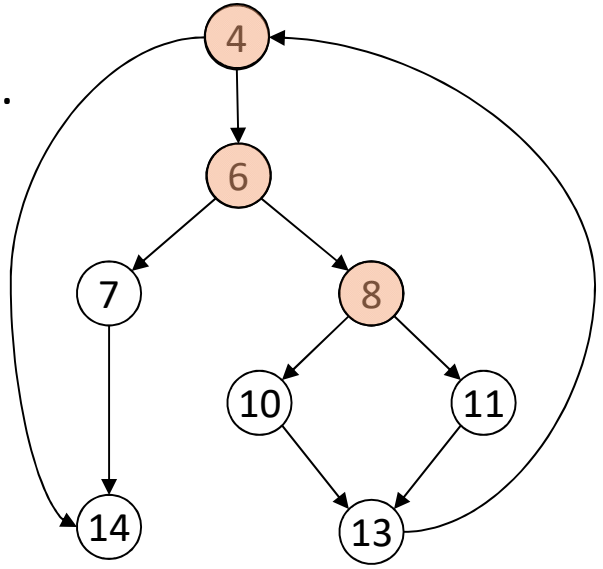
        V(G) = d +1.

        d 是流图 G 中单判定结点的数量。

- 基本路径测试的具体步骤
  - 第二步：计算环路复杂度
    - 例9：计算例8 的 *McCabe* 环路复杂度。
    - 解：例8 的程序控制流图的环路复杂度计算如下：
      - 流图的边数 m=10，点数 n=8，故

        V(G) = 10 – 8 + 2 = 4.
      - 平面流图中有4个区域，故 V(G) = 4.
      - 流图的单判定结点数 d=3，故

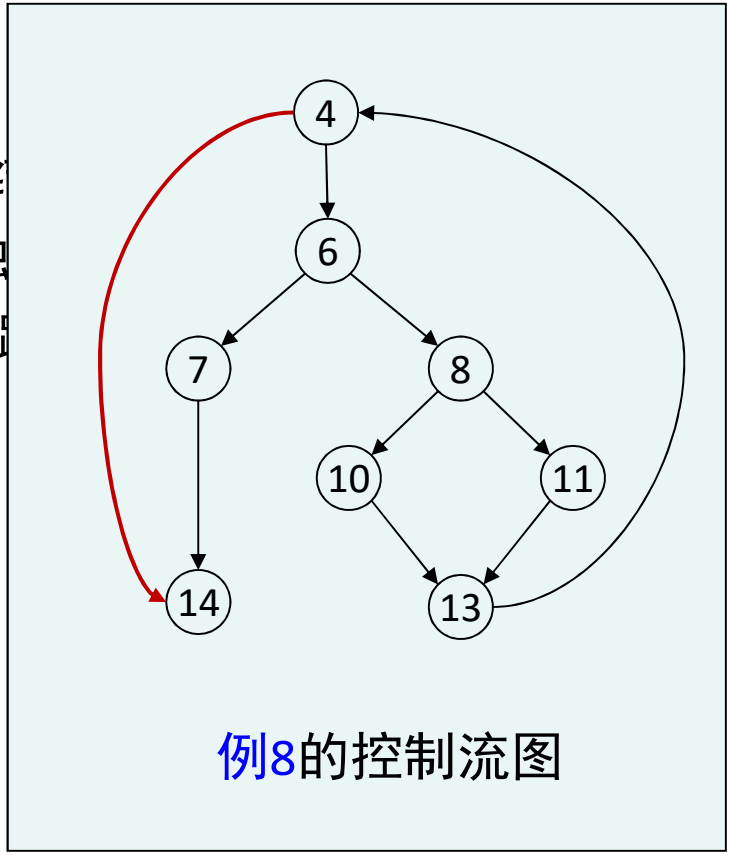
        V(G) = 3 + 1 = 4.



例8 的程序控制流图

- 基本路径测试的具体步骤
  - 第三步：准备测试用例
    - 根据 *McCabe* 环路复杂度的计算结果，图中存在4条独立的路径(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)。
    - 给出一个独立的路径集合如下：
      - 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
    - 根据上面的独立路径集合，设计输入用例数据，迫使程序分别执行上面4条路径。

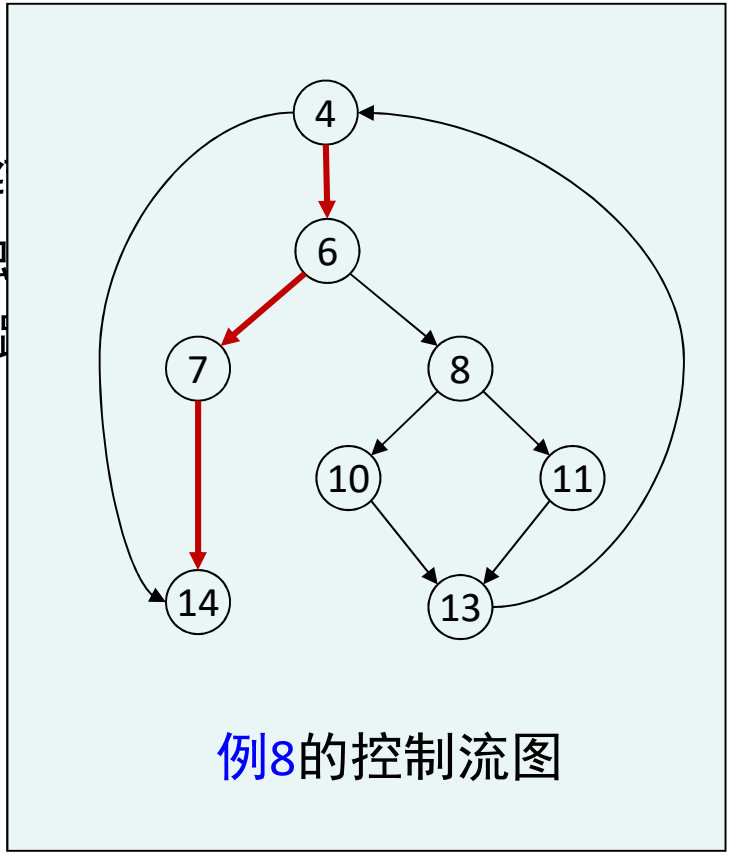
- 基本路径测试的具体步骤
  - 第三步：准备测试用例
    - 根据 *McCabe* 环路复杂度的计算
      (一条独立路径是指，和其他的独
      处理语句或一个新判断的程序通
      独立路径的条数)。
    - 给出一个独立的路径集合如下：
      - 路径1：4-14
      - 路径2：4-6-7-14
      - 路径3：4-6-8-10-13-4-14
      - 路径4：4-6-8-11-13-4-14



例8的控制流图

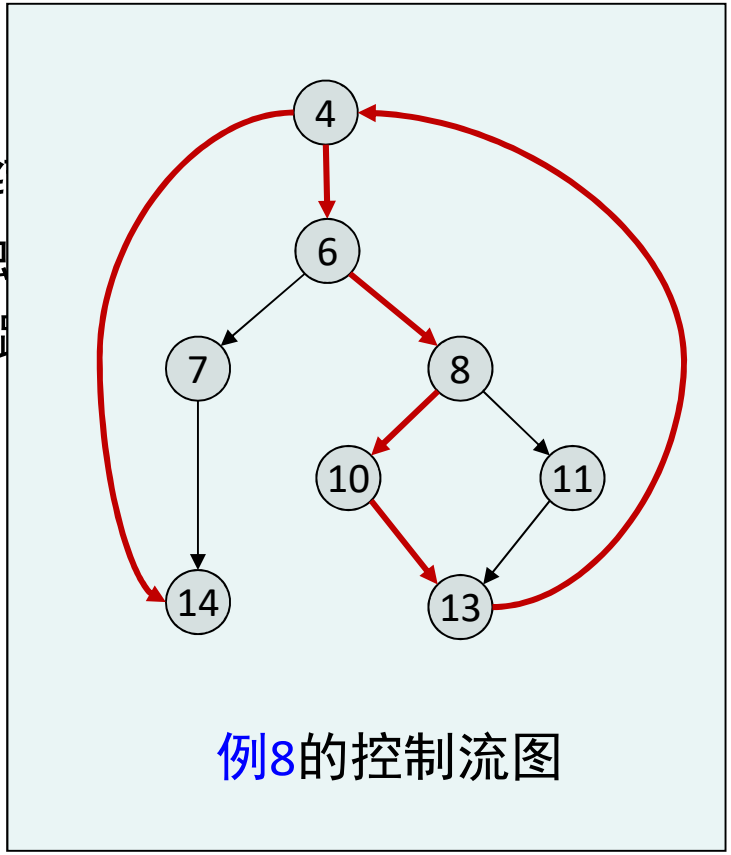
    - 根据上面的独立路径集合，设计输入用例数据，迫使程序分别执行上面4条路径。

- 基本路径测试的具体步骤
  - 第三步：准备测试用例
    - 根据 *McCabe* 环路复杂度的计算
      (一条独立路径是指，和其他的独
      处理语句或一个新判断的程序通
      独立路径的条数)。
    - 给出一个独立的路径集合如下：
      - 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
    - 根据上面的独立路径集合，设计输入用例数据，迫使程序分别执行上面4条路径。



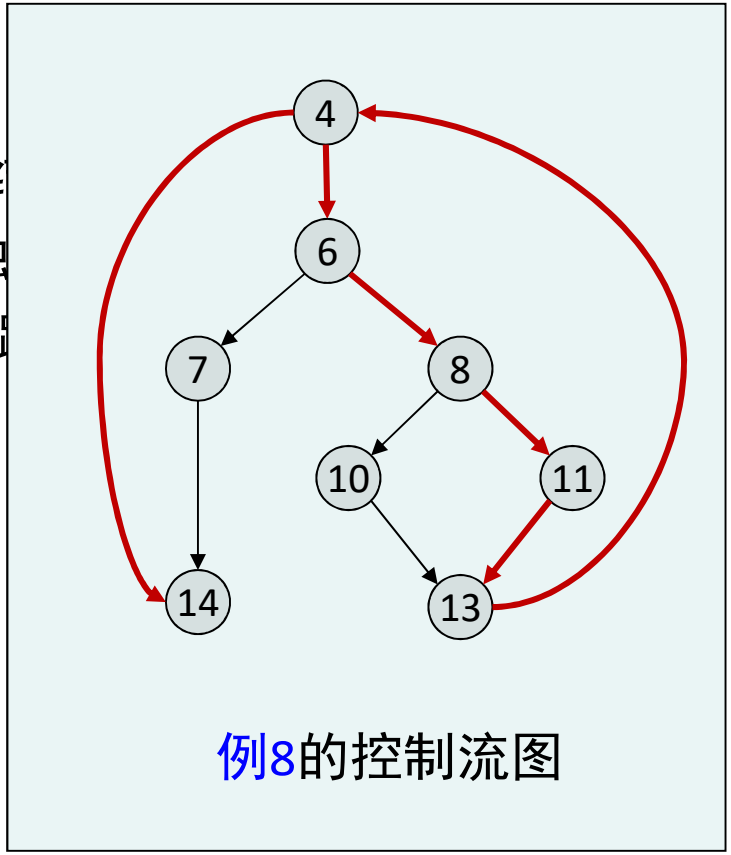
例8的控制流图

- 基本路径测试的具体步骤
  - 第三步：准备测试用例
    - 根据 *McCabe* 环路复杂度的计算
      (一条独立路径是指，和其他的独
      处理语句或一个新判断的程序通
      独立路径的条数)。
    - 给出一个独立的路径集合如下：
      - 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
    - 根据上面的独立路径集合，设计输入用例数据，迫使程序分别执行上面4条路径。



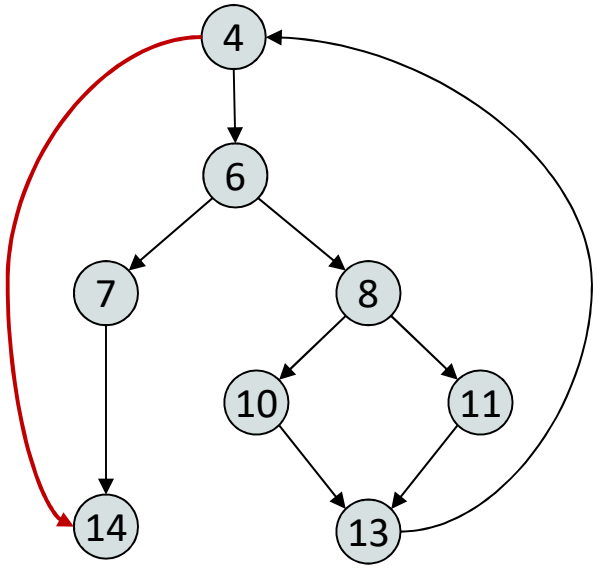
例8的控制流图

- 基本路径测试的具体步骤
  - 第三步：准备测试用例
    - 根据 *McCabe* 环路复杂度的计算
      (一条独立路径是指，和其他的独
      处理语句或一个新判断的程序通
      独立路径的条数)。
    - 给出一个独立的路径集合如下：
      - 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
    - 根据上面的独立路径集合，设计输入用例数据，迫使程序分别执行上面4条路径。



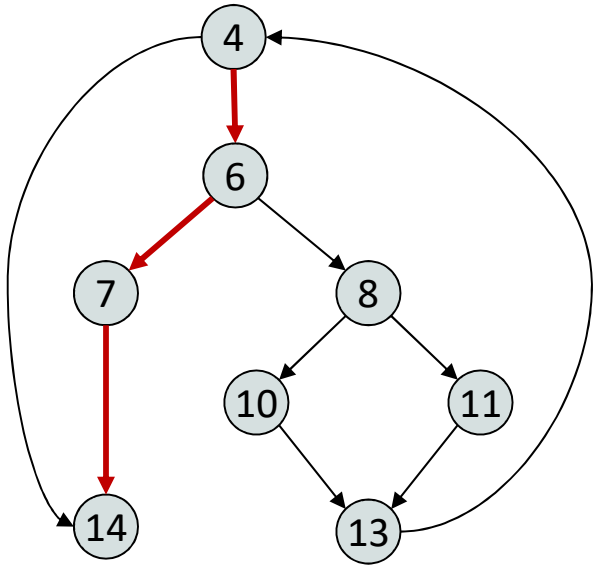
例8的控制流图

- 基本路径测试的具体步骤
  - 第四步：应用测试用例。满足上述基本路径集的测试用例是：
    - 路径1：4-14
      - 输入：iRecordNum=0，或者 iRecordNum<0 的某一个值
      - 预期结果：x=0



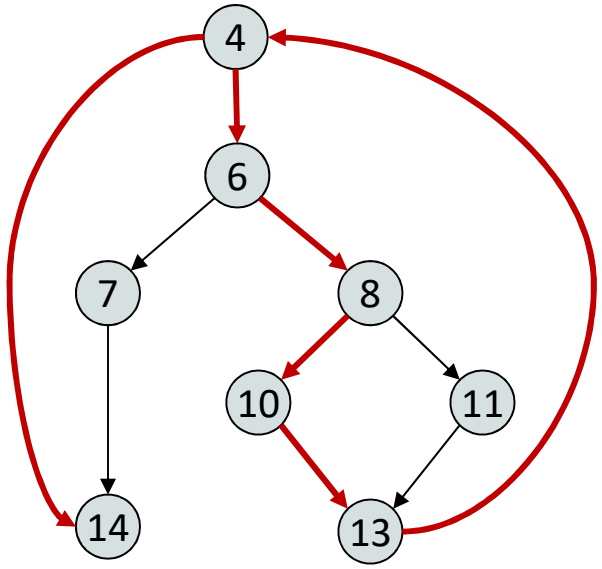

```
    void  Sort ( int iRecordNum, int iType )
1.  {
2.      int x = 0;
3.      int y = 0;
4.      while ( iRecordNum - - >= 0 )
5.      {
6.          if ( 0 == iType )
7.          { x = y + 2; break; }
8.          else
9.              if ( 1 == iType )
10.                 x = y + 10;
11.         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

- 基本路径测试的具体步骤
  - 第四步：应用测试用例。满足上述基本路径集的测试用例是：
    - 路径2：4-6-7-14
      - 输入数据：iRecordNum=1, iType=0
      - 预期结果：x=2



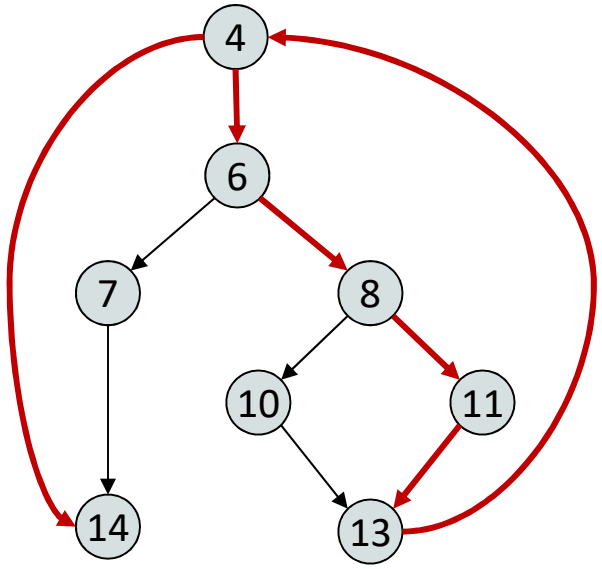

```
    void  Sort ( int iRecordNum, int iType )
1.  {
2.      int x = 0;
3.      int y = 0;
4.      while ( iRecordNum - - >= 0 )
5.      {
6.          if ( 0 == iType )
7.          { x = y + 2; break; }
8.          else
9.              if ( 1 == iType )
10.                 x = y + 10;
11.         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

- 基本路径测试的具体步骤
  - 第四步：应用测试用例。满足上述基本路径集的测试用例是：
    - 路径3：4-6-8-10-13-4-14
      - 输入数据：iRecordNum=1, iType=1
      - 预期结果：x=10




```
    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; break; }
8.          else
9.              if ( 1 == iType )
10.                 x = y + 10;
11.             else
12.                 x = y + 20;
13.         }
14. }
```

- 基本路径测试的具体步骤
  - 第四步：应用测试用例。满足上述基本路径集的测试用例是：
    - 路径4：4-6-8-11-13-4-14
      - 输入数据：iRecordNum=1, iType=2
      - 预期结果：x=20

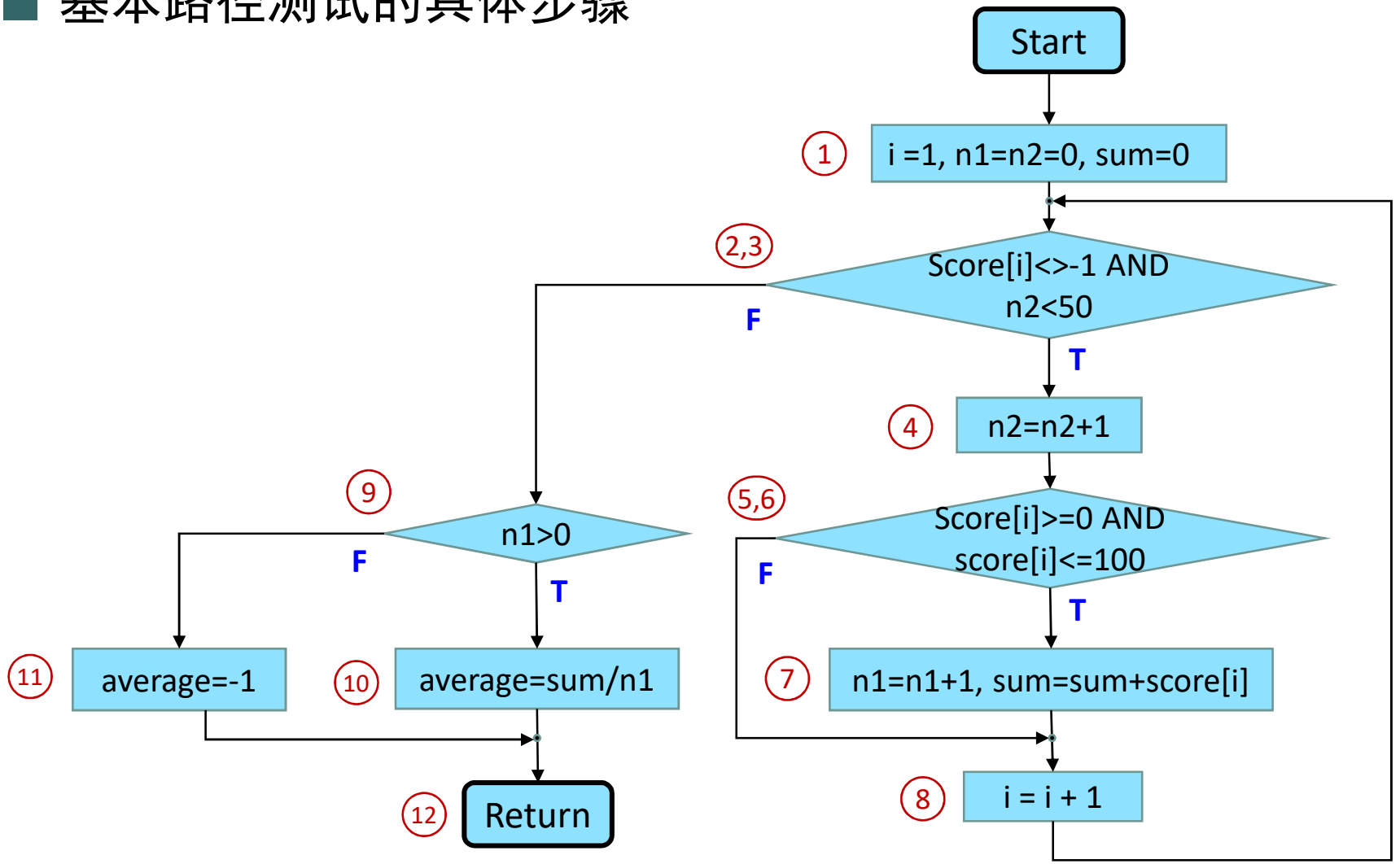



```
    void  Sort ( int iRecordNum, int iType )
1.  {
2.      int x = 0;
3.      int y = 0;
4.      while ( iRecordNum - - >= 0 )
5.      {
6.          if ( 0 == iType )
7.          { x = y + 2; break; }
8.          else
9.              if ( 1 == iType )
10.                 x = y + 10;
11.         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

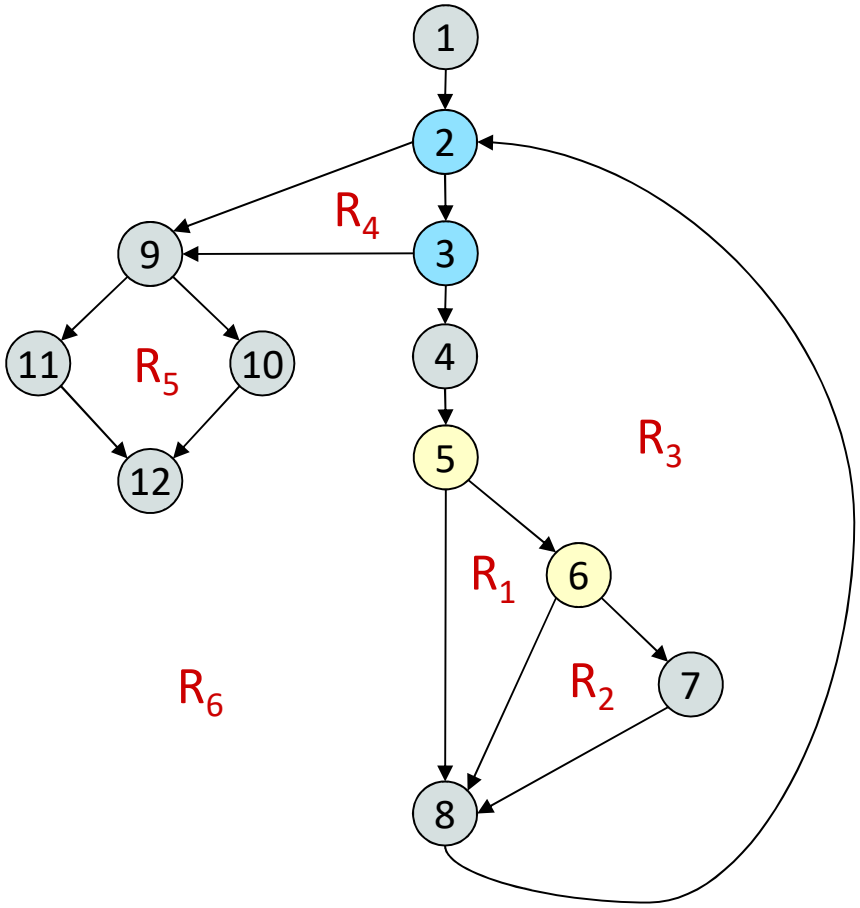
基本路径测试的具体步骤

- 例10：下例程序流程图描述了最多输入50个值 (以–1作为输入结束标志) 作为学生成绩的分数，计算其中有效的学生成绩 (0-100分) 的个数、总分数和分数平均值的一个程序。数据类型作如下定义：
  - Score[i]: 存储学生成绩的整型数组
  - n1: 有效的输入数目计数器
  - n2: 总输入数目计数器
  - sum: 有效的输入分数累计
  - average: 结束时有效的输入分数平均值

基本路径测试的具体步骤

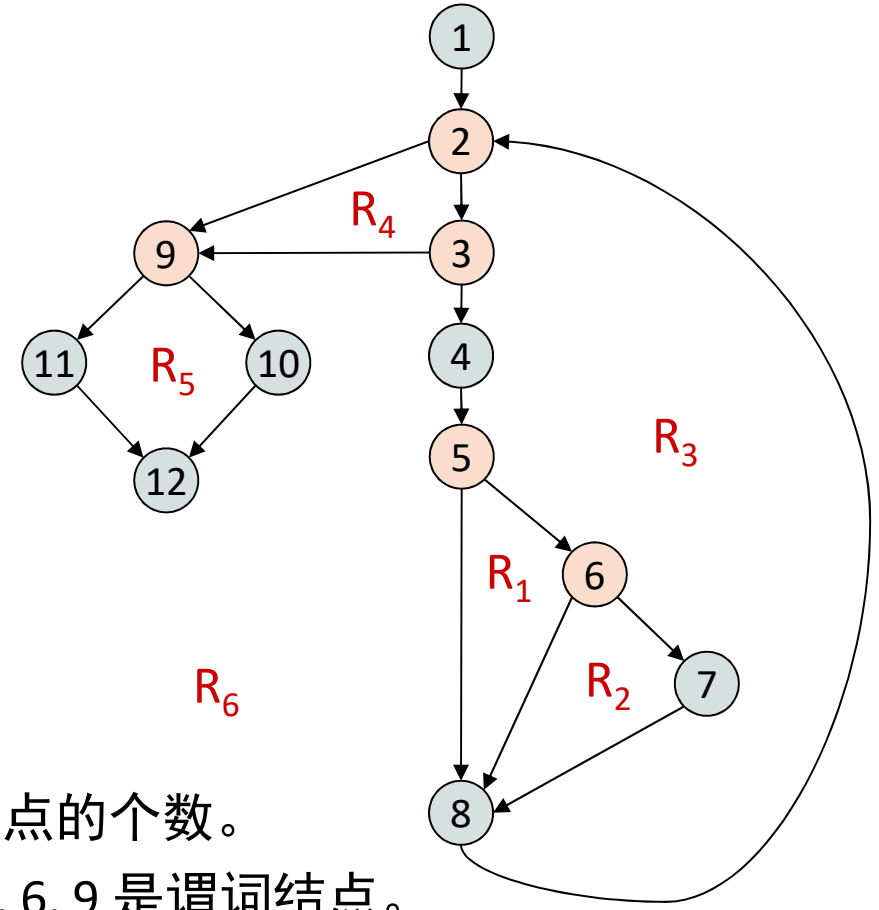


- 基本路径测试的具体步骤
  - 例10解：
    - 画出程序的控制流图



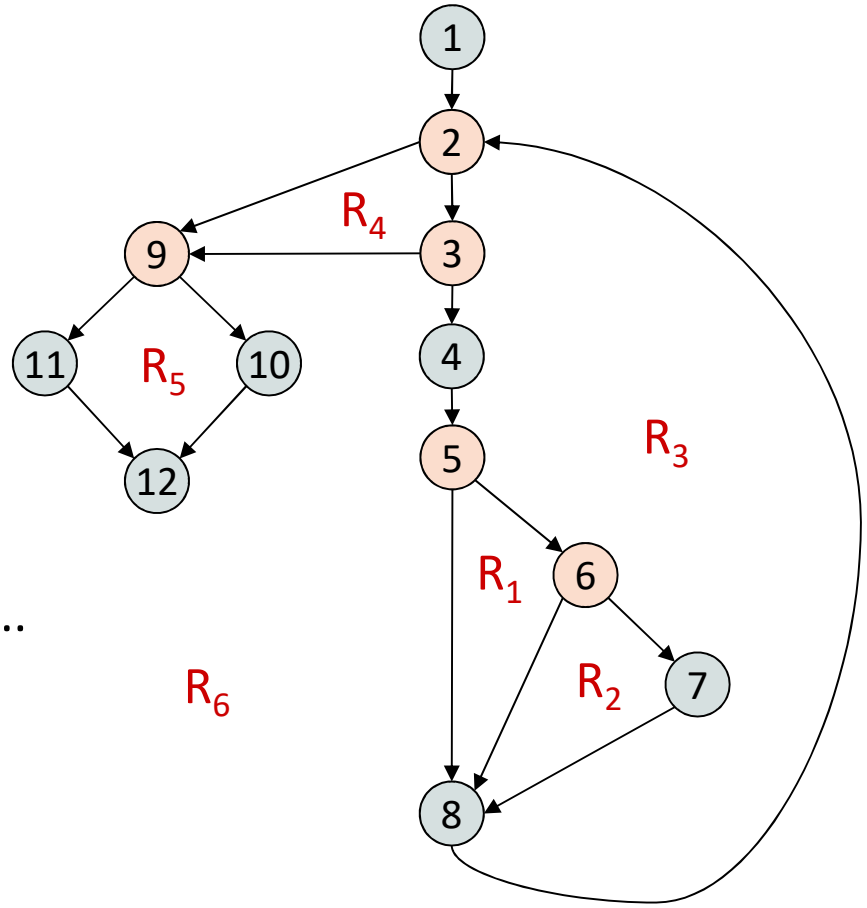
- 注意到其中的复合判定条件 "Score[i]<>-1 AND n2<50" 和 "Score[i]>=0 AND score[i]<=100" 已经被分离。

- 基本路径测试的具体步骤
  - 例10解：
    - 确定环路复杂度：
      - V(G) = m - n + 2
        = 16 - 12 + 2
        = 6
      - V(G) = 6 (区域数)
      - V(G) = d + 1
        = 5 + 1
        = 6
        d 为单条件判定谓词结点的个数。
      - 控制流图中结点 2, 3, 5, 6, 9 是谓词结点。



基本路径测试的具体步骤

- 例10解：
  - 确定6条独立的基本路径：
    - 路径1：1-2-9-10-12
    - 路径2：1-2-9-11-12
    - 路径3：1-2-3-9-10-12
    - 路径4：1-2-3-4-5-8-2 ...
    - 路径5：1-2-3-4-5-6-8-2 ...
    - 路径6：1-2-3-4-5-6-7-8-2 ...



实例：基本路径测试

- 例10解：
  - 为每一条独立路径各设计一组测试用例，以便强迫程序沿着该路径至少执行一次。

(1) 路径1 (1-2-9-10-12) 的测试用例：

- score[k]=有效分数值，当 k<i；
- score[i]=-1, $2 \le i \le 50$；
- 期望结果：根据输入的有效分数算出正确的分数个数 n1、总分 sum 和平均分 average。

(2) 路径2 (1-2-9-11-12) 的测试用例：

- score[1]=-1；
- 期望结果：average = -1，其他量保持初值。

实例：基本路径测试

- 例10解：
  - 为每一条独立路径各设计一组测试用例，以便强迫程序沿着该路径至少执行一次。

(3) 路径3 (1-2-3-9-10-12) 的测试用例：

- 输入多于50个有效分数，即试图处理51个分数，要求前51个为有效分数；
- 期望结果：n1=50，且算出正确的总分 sum 和平均分 average。

(4) 路径4 (1-2-3-4-5-8-2 …) 的测试用例：

- score[i]=有效分数，当 i<50；
- score[k]<0，k<i；
- 期望结果：根据输入的有效分数算出正确的分数个数 n1、总分 sum 和平均分 average。

实例：基本路径测试

- 例10解：
  - 为每一条独立路径各设计一组测试用例，以便强迫程序沿着该路径至少执行一次。

(5) 路径5 (1-2-3-4-5-6-8-2 ...) 的测试用例：

- score[i]=有效分数， 当 i<50；
- score[k]>100， k<i；
- 期望结果：根据输入的有效分数算出正确的分数个数 n1、总分 sum 和平均分 average。

(6) 路径6 (1-2-3-4-5-6-7-8-2 ...) 的测试用例：

- score[i]=有效分数， 当 i<50；
- 期望结果：根据输入的有效分数算出正确的分数个数 n1、总分 sum 和平均分 average。

- 小结：
  - 白盒测试方法是在测试人员全面了解程序内部逻辑结构的基础上，对程序的所有逻辑路径进行测试，是穷举路径测试。但是贯穿程序的路径数目庞大，而且即使每条路径都测试了仍然可能存在错误。
    - 穷举路径测试无法查出程序违反了设计规范；
    - 穷举路径测试无法查出程序中因遗漏路径而发生的错误；
    - 穷举路径测试可能发现不了一些与数据相关的错误。

Thank you!