

软件质量保证与测试

Software Quality Assurance and Testing

第 4 章 白盒测试

4.1 白盒测试概述

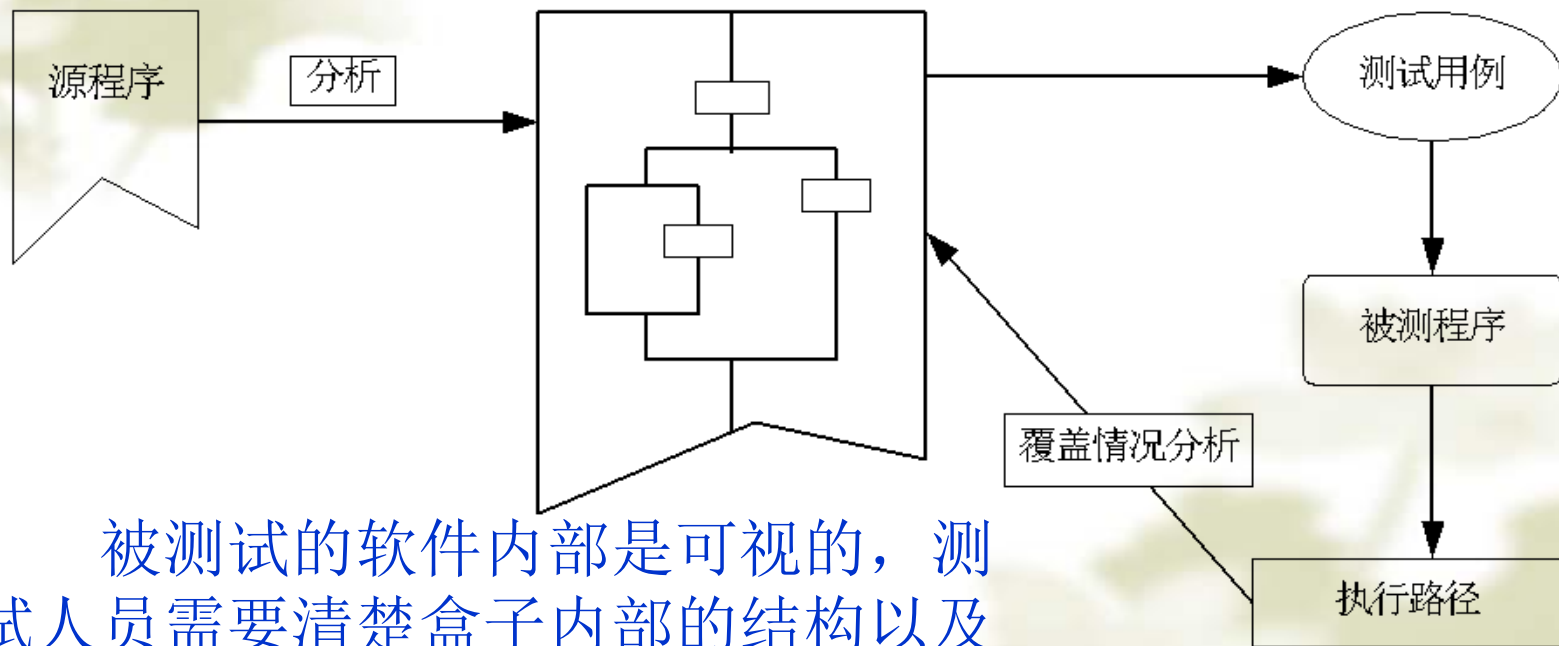


金陵科技学院

白盒测试

白盒测试是一种软件测试方法，测试对象基本上是源程序，它要求已知程序内部的逻辑结构、工作过程，检查验证每种内部操作是否符合设计规格，所有内部成分是否符合标准和要求。

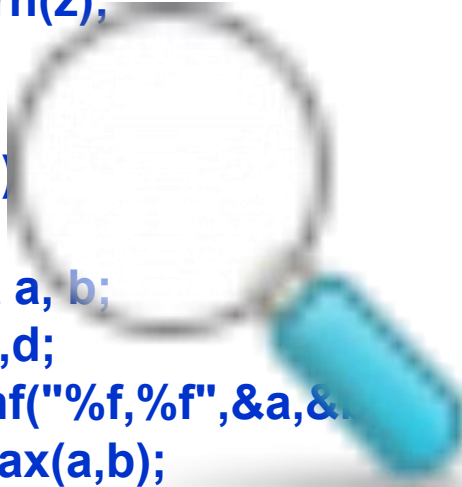
白盒测试



被测试的软件内部是可视的，测试人员需要清楚盒子内部的结构以及程序流程是如何执行的。

白盒测试

```
#include<stdio.h>
max(float x,float y)
{
    float z;
    z=x>y?x:y;
    return(z);
}
main( )
{
    float a, b;
    int c,d;
    scanf("%f,%f",&a,&b);
    c=max(a,b);
    printf("Max is %d\n",c);
}
```



软件的白盒测试是对软件及其执行过程做细致的检查，对软件的执行过程进行覆盖测试，检查程序中的每条通路是否符合预定要求，能正确工作，并可通过在程序不同位置设立检查点，来检查程序的状态，以确定实际运行状态与预期状态是否一致。

白盒测试

白盒测试既有静态测试也有动态测试。

静态白盒测试是指在不执行软件的情况下，对软件进行检查和分析，从而发现问题，找出缺陷的过程。

代码检查、静态结构分析、静态质量度量这些都是静态白盒测试方法。通过静态白盒测试，要尽可能检查发现代码中的逻辑错误，让代码达到逻辑正确性、高效性、清晰性、规范性、一致性等要求。



白盒测试

动态白盒测试是指先针对程序的内部逻辑结构设计测试用例。然后运行程序，输入测试用例，检验程序执行过程及最终结果是否符合预期要求，并查找问题和缺陷的过程。

逻辑覆盖、基本路径覆盖、域测试、符号测试和程序变异这些是动态白盒测试方法。

白盒测试

静态方法

代码检查
静态结构分析
静态质量度量
.....

动态方法

逻辑覆盖
基本路径覆盖
符号测号
程序变序变异
.....

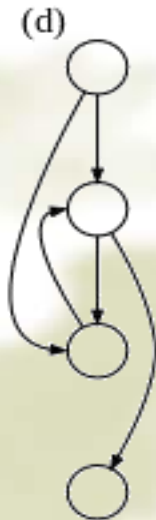
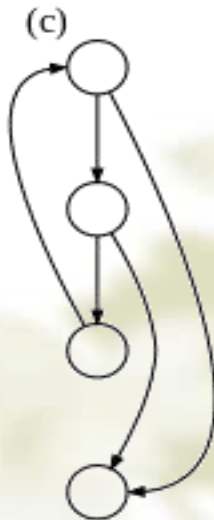
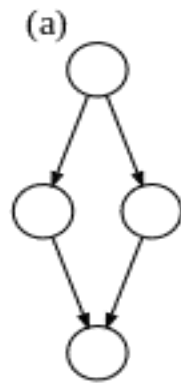
白盒测试

动态白盒测试的基本原则如下：

1. 保证一个模块中的所有独立路径至少被测试一次。
2. 对所有逻辑值均需测试 true 和 false。
3. 在上下边界及可操作范围内运行所有循环。
4. 检查内部数据结构以确保其有效性。

控制流图

控制流图（Control flow graph, 简称CFG）也叫控制流程图，它用图的方式来描述程序的控制流程，是对一个过程或程序的抽象表达。



控制流图

控制流图是一种有向图，可以形式化为：

$$G = (N, E, N_entry, N_exit)$$

N是节点集，程序中的每个语句都对应图中的一个节点，有时一组顺序执行、不存在分支的语句也可以合并为用一个节点表示。

边集 $E = \{ \langle n1, n2 \rangle \mid n1, n2 \in N, \text{ 且 } n1 \text{ 执行后, 可能立即执行 } n2 \}$ 。

控制流图

N_{entry} 和 N_{exit} 分别为程序的入口和出口节点，且G只具有唯一的入口节点 N_{entry} 和唯一的出口节点 N_{exit} 。

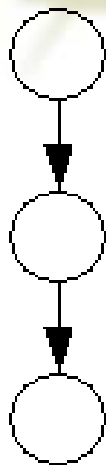
G 中的每个节点至多只能有两个直接后继。对于有两个直接后继的节点v, 其出边分别具有属性“T”或“F”，并且在G中的任意节点n, 均存在一条从 N_{entry} 经 n 到达 N_{exit} 的路径。

控制流图

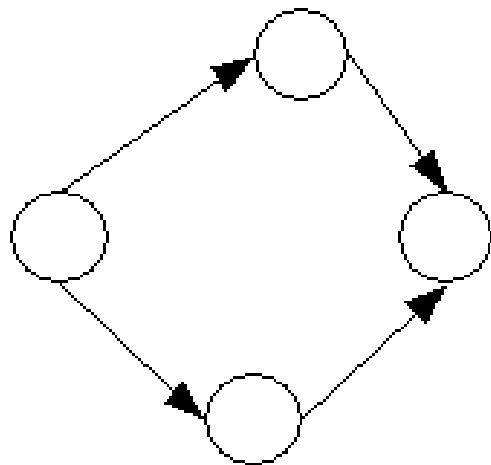
在控制流图中，用节点来代表操作、条件判断及汇合点，用弧或者叫控制流线来表示执行的先后顺序关系。

圆圈称为控制流图的一个节点，它表示一个或多个无分支的语句；有向箭头称为弧或者叫控制流线，表示执行的先后顺序关系。

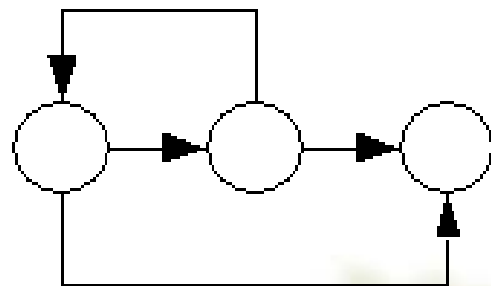
控制流图



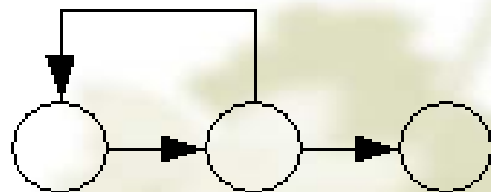
顺序结构



IF 选择结构



WHILE 循环结构



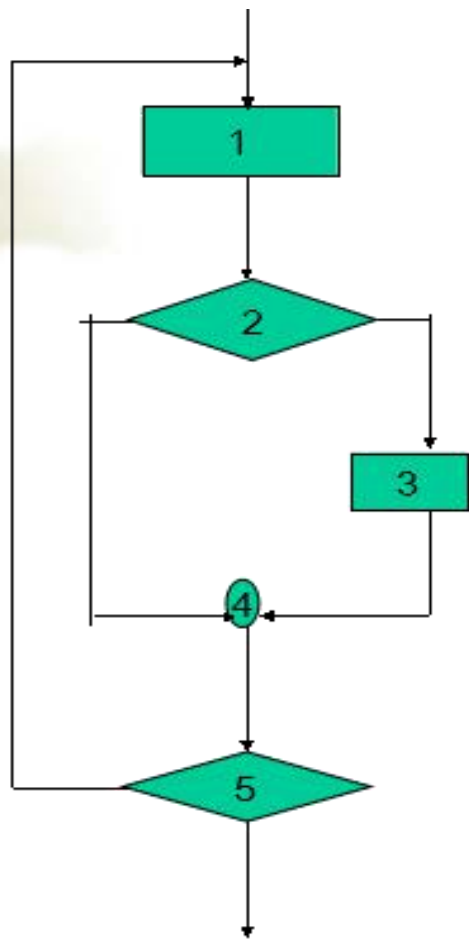
UNTIL 循环结构

控制流图

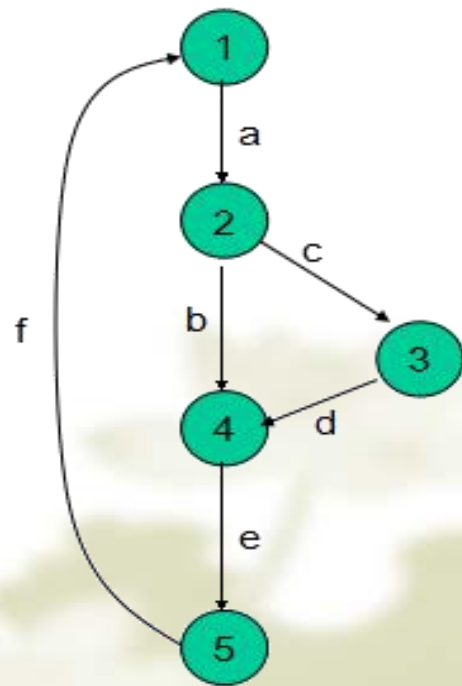
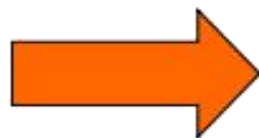
可以根据程序来得出其控制流图，也可以由程序流程图来转换得到控制流图。

需要注意的是，在将程序流程图简化成控制流图时，在选择或多分支结构中，分支的汇聚处应有一个汇聚结点；如果一个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结构。

控制流图

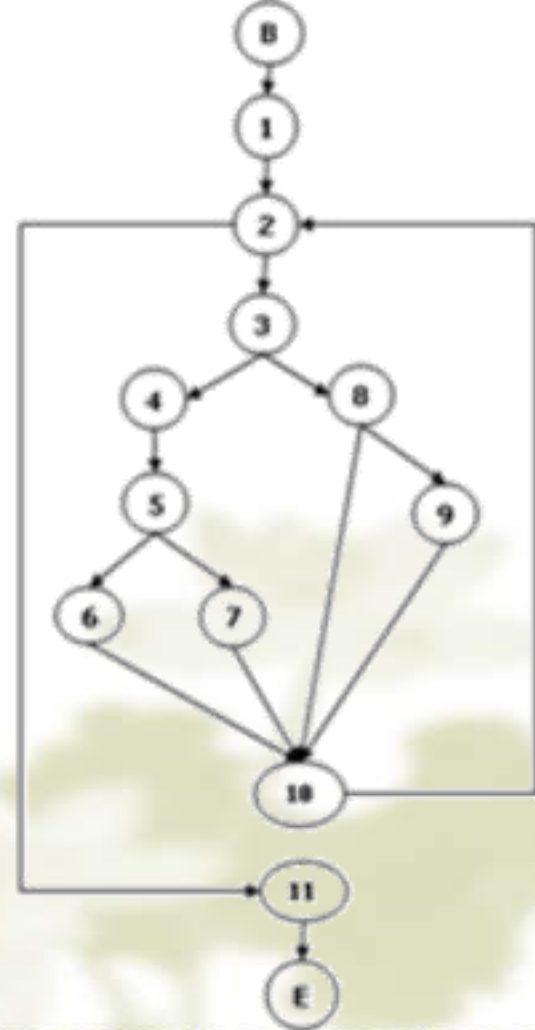
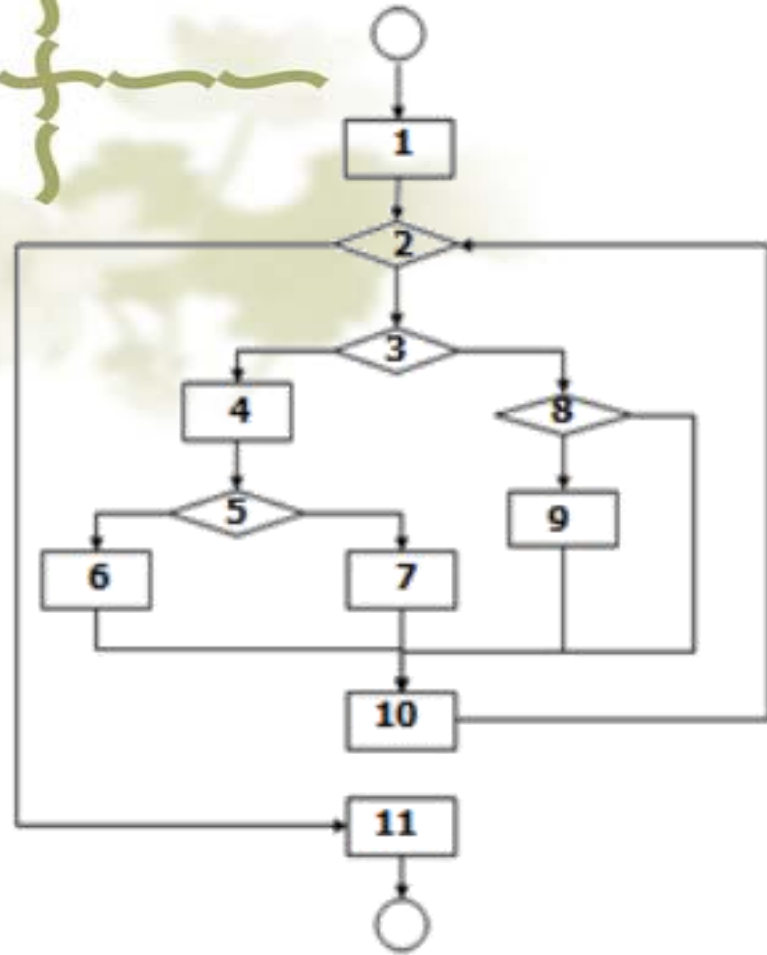


程序框图

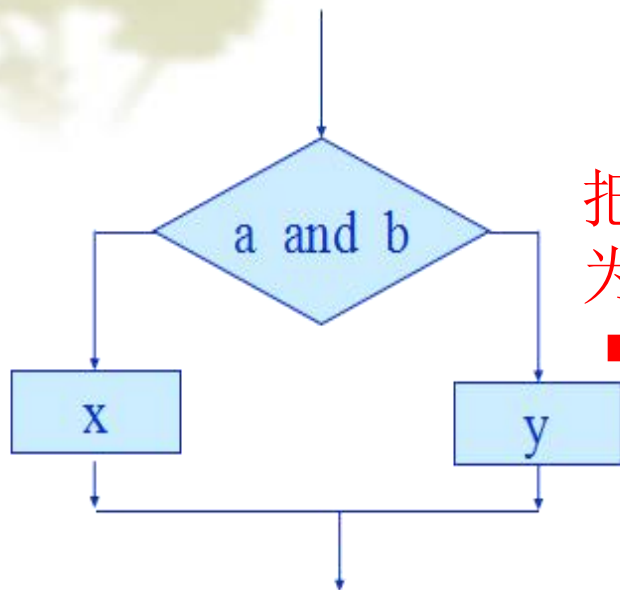


控制流图

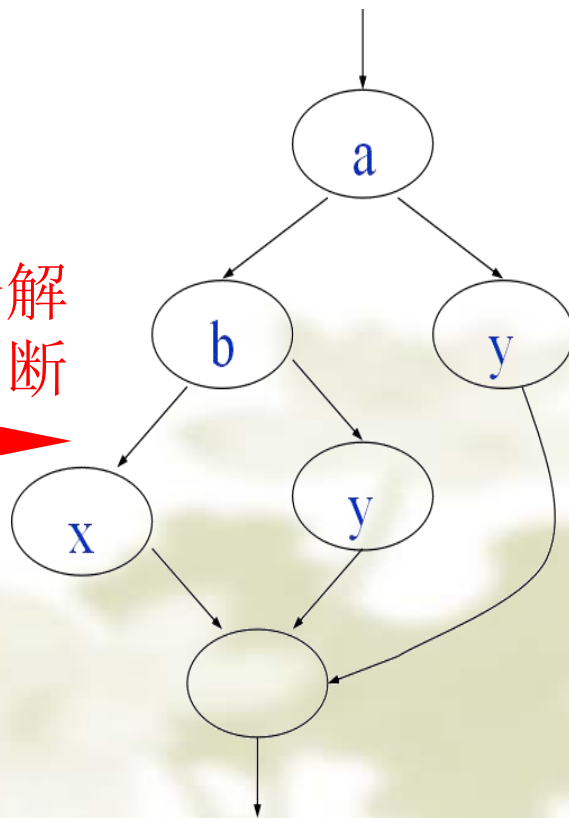
控制流图



控制流图



把多条件判断分解
为多个单条件判断



程序的复杂度

我们来思考一个问题，应当如何度量一个程序的复杂度？

是否程序的大小就能准确反映程序的复杂程度呢？一个1000行的程序就一定比一个100行的程序复杂吗？答案是否定的。

这就好比100道100以内加减法题并不比做一道二元积分题复杂是一样的道理。

例如，一个由1000行顺序执行的赋值语句、输出语句组成的程序，并不比一个100行的排序算法程序复杂。

程序的复杂度

简单的用程序的大小来度量程序的复杂度是片面和不准确的，而环路复杂度是程序复杂度度量的方法之一。

程序中的控制路径越复杂，环路越多，则环路复杂度越高，环路复杂度用来定量度量程序的逻辑复杂度。

根据程序的控制流图，可以计算程序的环路复杂度。

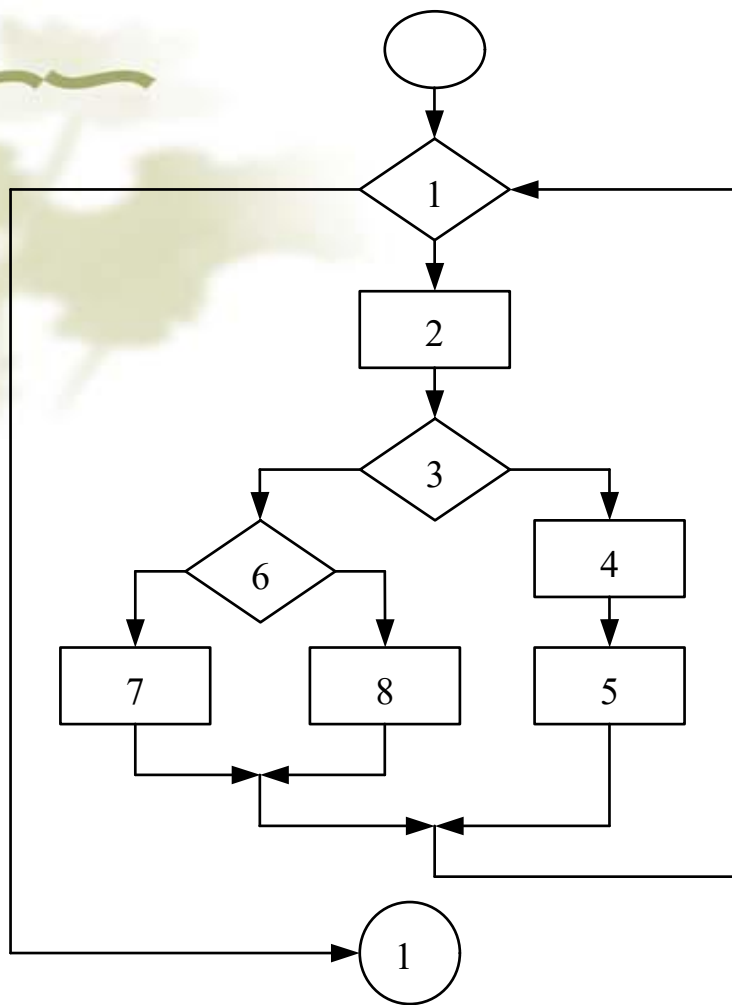
程序的环路复杂度

在画出控制流图的基础上，程序环路复杂度的计算方法如下：

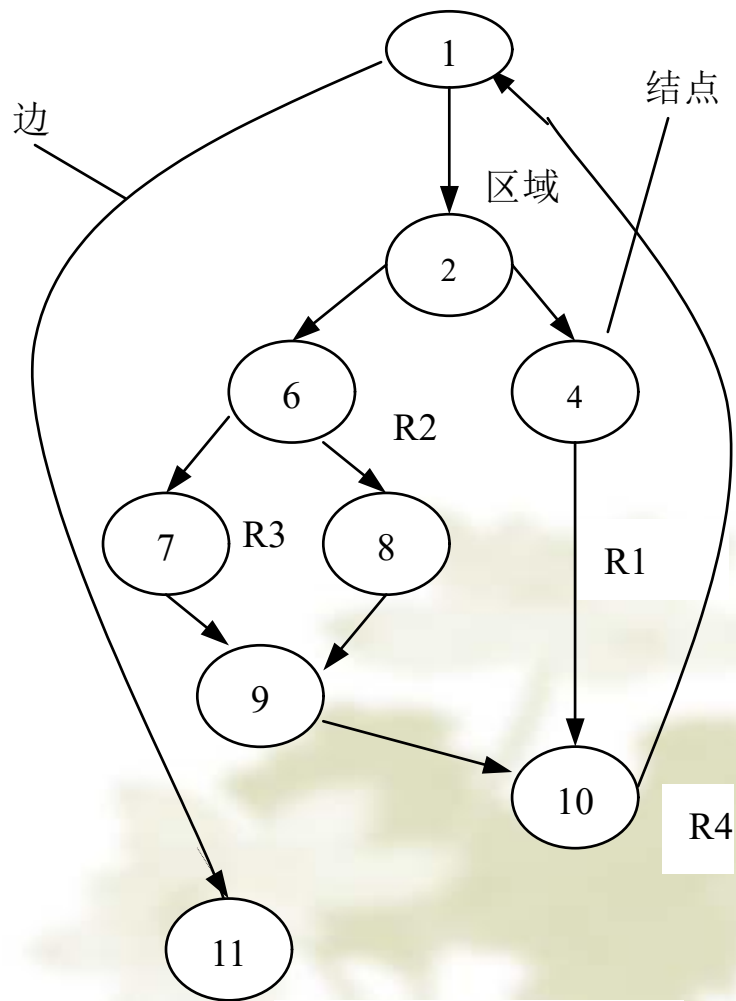
- ① 将环路复杂度定义为控制流图中的区域数。
- ② 设 E 为控制流图的边数， N 为图的结点数，则定义环路的复杂性为 $V(G)=E-N+2$ 。
- ③ 若设 P 为控制流图中的判定结点数，则有 $V(G)=P+1$ 。

对于同一个控制流图，三种计算方法算出的结果是一样的。

程序的环路复杂度



(a) 程序流程图



(b) 控制流图

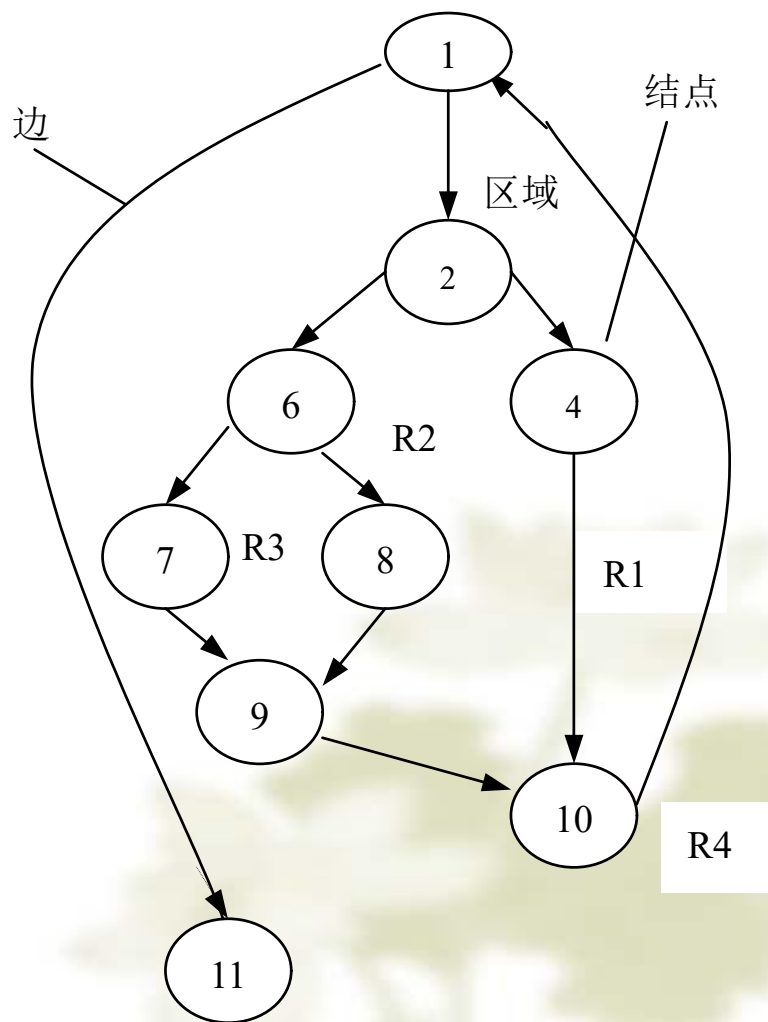
程序的环路复杂度

① 图中的区域数为4，故环路复杂度 $V(G) = 4$ 。

② 边数 $E = 11$ ，节点数 $N=9$ ，环路复杂度 $V(G) = E - N + 2 = 4$ 。

③ 图中的判定结点数 $P = 3$ ，则有 $V(G) = 3 + 1 = 4$ 。

三种计算方法算出的结果相等，右图的环路复杂度为4。



本节内容就讲到这里，谢谢，再见！



金陵科技学院