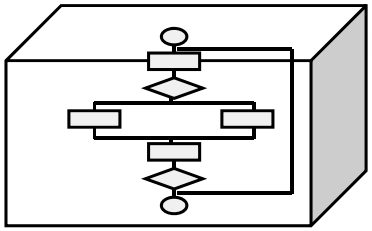
软件质量保证与软件测试
--路径测试

结构性测试方法

- 特征：
  - 基于被测程序的源代码
  - 支持严格定义、数学分析和精确度量



程序图

- 定义
  给定采用命令式程序设计语言编写的一段程序，其程序图是一种有向图，图中的节点表示语句片段，边表示控制流
- 如果i和j是程序图中的节点，从节点i到节点j存在一条边，当且仅当对应节点j的语句片段可以在对应节点i的语句片段以后立即执行

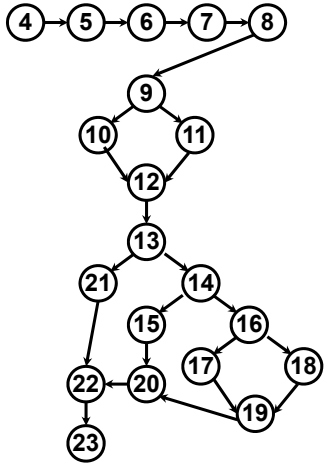
三角形程序的代码（1）

```
1. Program triagle2   'Structured programming version
2. Dim a, b, c  as integer
3. Dim IsATriangle  as  Boolean
   'Step1: Get Input
4. Output ("Enter 3 integers which are sides of a triangle")
5. Input (a，b，c)
6. Output ("Side A is "，a)
7. Output ("Side B is "，b)
8. Output ("Side C is "，c)
   'Step 2: Is A Triangle?
9. If  (a < (b + c)) AND (b < (a + c)) AND (c < (a + b))
10.    Then IsATriangle = True
11.    Else IsATriangle = False
```

三角形程序的代码（2）

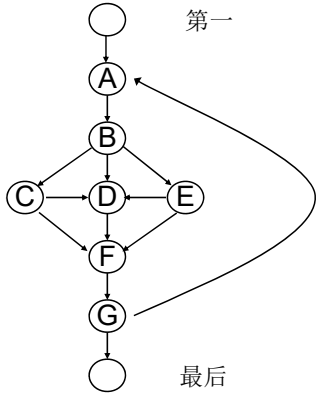
```
12. End If
    'Step 3: Determine Triangle Type
13. If  IsATriangle
14.   Then If  (a = b) AND (b = c)
15.        Then Output ("Equilateral")
16.        Else   If  (a ≠ b) AND (a ≠ c) AND (b ≠ c)
17.                 Then Output ("Scalene")
18.                 Else Output ("Isosceles")
19.             End If
20.          End If
21.   Else Output ("Not a Triangle")
22. End If
23. End trangle2
```

三角形程序的程序图



- 行号引用语句和语句片段
- 这里使用一种判断要素:
  - 有时将一个片段作为单独节点是很方便的
  - 有时又需要将其与语句的另一部分合并
- 还需要判断是否将节点与非可执行语句关联起来
  - 例如变量和类型说明语句
  - 这里不做这样的关联

一个简单程序图



- 在这段程序中，循环范围内从节点B到节点F有五条路径
- 如果循环最多有18次重复，则存在4.77万亿(5的18次方)不同的程序执行路径
- 完全测试程序路径不现实也不可能

主要内容

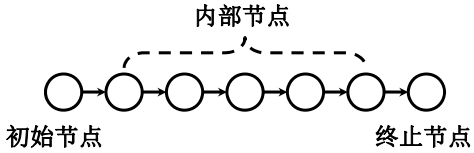
- DD-路径
- 测试覆盖指标
- 基路径测试
- 指导方针和观察

DD-路径

- 结构性测试最著名的形式以叫做决策到决策路径(DD-路径)的结构为基础
- 这个名称指语句的一种序列
  - 从决策语句的“出路”开始，到下一个决策语句的“入路”结束
  - 在这种序列中没有内部分支，因此对应的节点像排列起来的一行多米诺骨牌，当第一张牌推倒后，序列中的其他牌也会倒下

DD-路径与有向图的关系

- DD-路径可通过有向图中的节点路径定义，可以叫做路径链
- 其中链是一条起始和终止节点不同的路径
- 并且每个节点都满足内度=1和外度=1



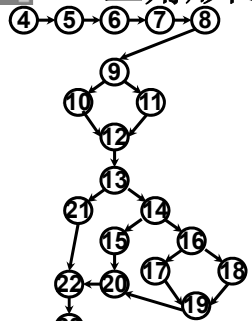
- 退化链：有一种长度为0 的链，即链有一个节点和0条边

DD-路径的定义

定义：

- DD-路径是程序图中的一条链，分为如下类型：
  - 情况1：由一个节点组成，内度=0
    - 结构化程序的程序图的唯一源节点
  - 情况2：由一个节点组成，外度=0
    - 结构化程序的程序图的唯一汇节点
  - 情况3：由一个节点组成，内度≥ 2或外度≥2
    - 复杂节点，保证节点不会包含在多个DD-路径中
  - 情况4：由一个节点组成，内度=1并且外度=1
    - 用于“短分支”， 也用于遵守一个判断是一个DD-路径原则
  - 情况5：长度≥1的最大链
    - “正常情况”，其中DD-路径是单入口、单出口的节点序列(链)，"最大"，用于确定正常(非平凡)链的最终节点

三角形问题的DD-路径类型



- 节点5到8是情况5的DD-路径。节点8是DD-路径的最后节点，因为它是遵循链的2-连接性质的最后节点
- 如果超过节点8包含节点9，就会违反链的内度=外度=1的准则
- 如果在节点7处停止，就会违反“最大”准则

程序图节点	DD-路径名称	定义情况
4	第一	1
5-8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	最后	2

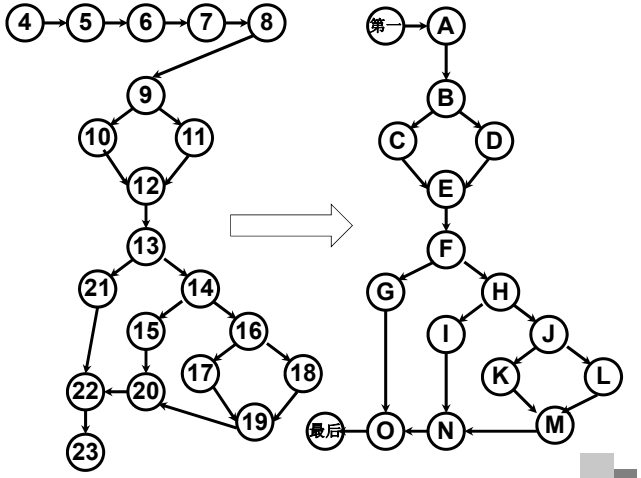
DD-路径图

■定义：

给定采用命令式语言编写的一段程序，其DD-路径图是有向图。其中，节点表示其程序图的DD-路径，边表示连续DD-路径之间的控制流

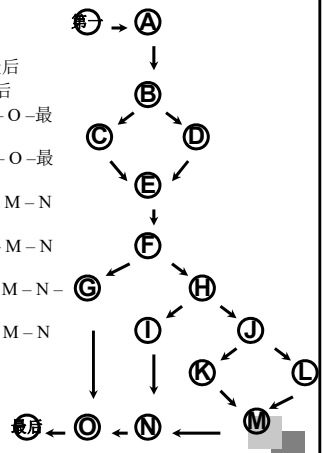
- 实际上DD-路径图是一种压缩图，在这种压缩图中，2-连接组件被压缩为对应情况5的DD-路径的单个节点
- 需要单节点DD-路径（情况1~4）是为了遵循一条语句（或语句片段）恰是一条DD-路径的约定
- 如果没有这个约定，则会得到很差的DD-路径，有些语句片段可能会出现在多个DD-路径中

三角形程序的DD-路径图



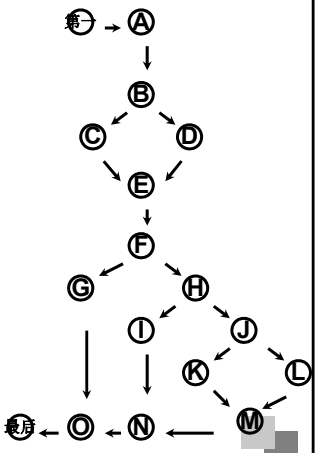
拓扑路径

- 从图可以得到8条拓扑上可行的路径
  - 1.第一 – A – B – C – E – F – G – O – 最后
  - 2.第一– A – B – D – E – F – G – O –最后
  - 3.第一 – A – B – C – E – F – H – I – N – O –最后
  - 4.第一 – A – B – D – E – F – H – I – N – O –最后
  - 5.第一– A – B – C – E – F – H – J – K – M – N – O –最后
  - 6.第一– A – B – D – E – F – H – J – K – M – N – O –最后
  - 7.第一– A – B – C – E – F – H – J – L – M – N – O –最后
  - 8.第一– A – B – D – E – F – H – J – L – M – N – O –最后



可行路径

- 如果分析程序源码实现，发现只有4条可行路径
  - 2.第一– A – B – D – E – F – G – O –最后
  - 3.第一 – A – B – C – E – F – H – I – N – O –最后
  - 5.第一– A – B – C – E – F – H – J – K – M – N – O –最后
  - 7.第一– A – B – C – E – F – H – J – L – M – N – O –最后



练习1

对下列代码，给出程序图、DD-路径、 DD-路径图

```
1.typedef struct {
    double x,y;
} Point;


2.int convex(Point *p,int n) {
3.int i,j,k;
4.int flag = 0;
5.double z;
6.if (n < 3)
7.   return(0);
```

```
8.for (i=0;i<n;i++) {
9.    j = (i + 1) % n;
10.   k = (i + 2) % n;
11.    z = (p[j].x - p[i].x) * (p[k].y - p[j].y);
12.    z -= (p[j].y - p[i].y) * (p[k].x - p[j].x);
13.   if (z < 0)
14.       flag |= 1;
15.   else if (z > 0)
 16.       flag |= 2;
17.}
18.    if (flag == 3)
 19.       return －1; //CONCAVE

20.if (flag != 0)
21.   return 1; //CONVEX
22.else
23.    return 0;

24.}
```

练习2

对下列代码，给出程序图、DD-路径、 DD-路径图

```
1.FindMean (FILE ScoreFile)
2.{  float SumOfScores = 0.0;
3.     int NumberOfScores = 0;
4.     float Mean=0.0; float Score;
5.     Read(ScoreFile, Score);
6.     while (! EOF(ScoreFile) {
7.       if (Score  > 0.0 ) {
8.         SumOfScores = SumOfScores + Score;
9.         NumberOfScores++;
10.      }
11.      Read(ScoreFile, Score);
12.    }
 /* Compute the mean and print the result */
13.    if (NumberOfScores > 0) {
14.      Mean = SumOfScores / NumberOfScores;
15.      printf(" The mean score is %f\n", Mean);
16.    } else {
17.       printf ("No scores found in file\n");
18.    }
19.}
```

主要内容

- ■DD-路径
- ■测试覆盖指标
- ■基路径测试
- ■指导方针和观察

测试覆盖指标

■测试覆盖指标，是度量一组测试用例覆盖(或执行)某个程序程度的工具

指标	覆盖描述
C_0	所有语句
C_1	所有DD-路径（判断分支、判断/判定覆盖）
C_1P	所有判断的每种分支
C_2	C_1覆盖+循环覆盖
C_d	C_1覆盖+DD-路径的所有依赖对偶
C_{MCC}	多条件覆盖
C_ik	包含最多k次循环的所有程序路径（通常k=2）
C_{stat}	路径具有“统计重要性”的部分
C_∞	所有可能的执行路径

当通过一组测试用例满足DD-路径覆盖要求时，可以发现全部缺陷中的大约85%

结构性测试回顾——路径测试

- 测试覆盖指标（基于程序图）

指标	覆盖描述	
C_0	所有语句	语句覆盖（点覆盖）
C_1	所有DD-路径（判断分支）	
C_{1p}	所有判断的每种分支	判定覆盖（边覆盖）
Cd	C_1覆盖 + DD-路径的所有依赖对偶	
C_2	C_1覆盖 + 循环覆盖	循环覆盖
C_{MCC}	多条件覆盖	条件覆盖
C_{ik}	包含最多k次循环的所有程序路径（通常k=2）	
C_{stat}	路径具有“统计重要性”的部分	
C_∞	所有可能的执行路径	路径测试

基于指标的测试

- 上表中的测试指标告诉我们要测试什么，但没有说怎么测试
- 下面进一步研究根据表中的指标执行源代码的技术

- 语句与判断测试
- DD-路径测试
- DD-路径的依赖对偶
- 多条件覆盖
- 循环覆盖

语句与判断测试

■这些覆盖指标要求找出一组测试用例，使得当执行时，程序图的所有节点都至少走过一次

■由于我们的表示方式允许语句片段作为独立的节点，因此语句和判断层次(C0和C1) 可合并为一个问题

■例如，在三角形问题程序图中，节点9、10、11和12是一个完整的if- then-else语句。

■如果要求节点对应完整语句，则可以只执行判断的一个选项并满足语句覆盖准则

■由于允许语句片段，因此可以很自然地将这种语句分解为三个节点。这样做导致判断分支覆盖

DD-路径测试

■等价于判断分支覆盖指标

■对于C_1指标，要求执行每个判断分支，则应**遍历DD-路径图中的每条边**

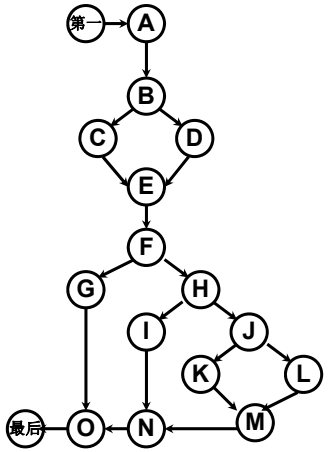
■对于C_{1P} 覆盖，则需要覆盖每个分支的所有情况，如果为条件语句，则应覆盖真、假分支；若为CASE语句，则应覆盖每个子句

■对于较长的DD-路径，一般代表复杂计算，应用多个功能性测试可能比较适合，尤其是边界值和特殊值测试

DD-路径的依赖对偶

- C_d涉及第10章将要讨论的问题，即数据流测试
- DD-路径对偶之间的最常见的依赖关系是定义/引用关系，其中变量在一个DD-路径中被定义(接受值)，在另一个DD-路径中被引用
- 简单DD-路径覆盖可能不会遍历这些依赖关系，因此更深的缺陷类不会被发现

DD-路径的依赖对偶



如：

- B和D就是DD-路径的依赖对偶
- C和L也是DD-路径的依赖对偶

多条件覆盖

- 不是直接遍历判断是到其真或假分支，而应该研究可能出现分支的不同方式
- 这里可以看到一种有意思的折衷：语句复杂性和路径复杂性
- 多条件覆盖可保证上述复杂性不会被DD-路径覆盖所掩盖
- 如：三个简单条件的复合条件**If(A>5 and B!=10 and C<100)** 会有八行（2的3次方），产生八个测试用例

(A>5,B!=10,C<100)、 (A>5,B!=10,C>=100)、

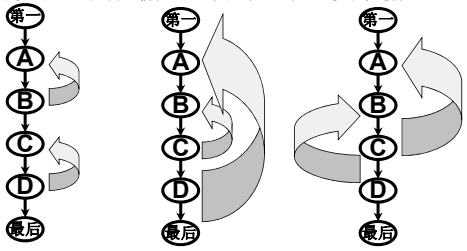
(A>5,B==10,C<100)、 (A>5,B==10,C>100)

(A<5,B!=10,C<100)、 (A<5,B!=10,C>=100)、

(A<5,B==10,C<100)、 (A<5,B==10,C>100)

循环覆盖

- 循环是源代码中非常容易出错的部分，**循环的分类：**
  - 串联循环是不相交的简单循环序列
  - 嵌套循环是一个循环包含在另一个循环中的循环
  - 复杂循环，如果跳转到某个循环内(或跳转出)，而这个分支位于其他循环的内部，就是复杂循环



串联循环　　嵌套循环　　复杂循环

循环覆盖

- 每个循环都包含一个判断，并且需要测试判断的两个分支:
  - 一个遍历循环
  - 另一个退出(或不进入)循环
- 采用经过修改的边界值方法，循环指数按最小值、一般值和最大值给出

循环覆盖

- 循环测试的路径选择
- 1.简单循环 （边界值方法）
- 1）零次循环：从循环入口到出口
- 2）一次循环：检查循环的初始值
- 3）二次循环：检查初始值+1
- 4）m次循环：检查多次循环
- 5）最大次数循环-1
- 6）最大次数循环
- 7）最大次数+1

循环覆盖

- 循环测试的路径选择
- 2.嵌套循环
- 1）对最内层循环做简单循环的全部测试，其他层的循环变量置为最小值
- 2）逐步外推，对其外面的一层循环进行简单循环测试，内层取典型值，当前循环的所有外层取最小值
- 3）反复进行，直到所有各层循环测试完毕
- 4）对全部各层循环同时取最小循环次数、同时取最大循环次数进行测试

测试覆盖分析器

- 覆盖分析器是一类测试工具，可用于自动化测试支持
- 测试人员可以在经过覆盖分析器“处理”的程序上执行一组测试用例
- 分析器再使用处理代码所生成的信息生成覆盖报告

主要内容

- DD-路径
- 测试覆盖指标
- 基路径测试
- 指导方针和观察

向量空间理论基础

- 向量空间也称为线性空间
- 设***V*** 为*n* 维向量的集合，若集合***V*** 非空，且集合***V*** 对于*n* 维向量的加法及数乘2 种运算封闭，即(1)若α ∈***V***,β ∈***V***，则α +β ∈***V***；(2)若α ∈***V***,λ ∈ *R* ，则λα ∈***V***，则称集合***V*** 为*R* 上的向量空间
- 在这个向量空间 ***V*** 上，若有*r* 个向量α_1，α_2, …, α_r ∈***V*** ，且满足：
- (1) α_1，α_2, …, α_r线性无关;
- (2) *V* 中任一向量都可由α_1，α_2, …, α_r线性表示，则称向量组α_1，α_2, …, α_r为向量空间***V*** 的一个基

基路径测试

- 空间中的一切都可以用基表示，并且如果一个基元素被删除，则这种覆盖特性也会丢失
- 对测试的潜在意义是，如果可以把程序看做是一种向量空间，则这种空间的基就是要测试的非常有意义的元素集合
- 如果基没有问题，则可以希望能够用基表述的一切都是没有问题的

基路径测试

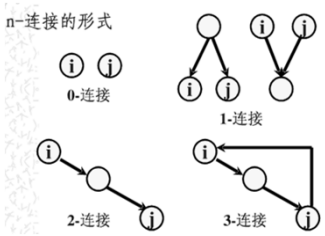
- 基路径：对应于测试来说，所有的程序路径认为是一个集合，那么在这些路径当中必然会存在一个最小路径的集合，我们称之为基路径；只要这几个路径是正确的，那么可以认为整个程序的所有路径都是正确的
- 基路径测试：通过某种算法来确定基路径，然后研究功能性测试的测试用例是否完全覆盖了这些基路径，如果完全覆盖，则代表测试完毕

McCabe的测试观点

■强连接图的圈数量就是图中线性独立环路的数量

■环路：类似于链，不出现内部循环或判断，但是初始节点是终止节点

■环路是一组3-连接节点



McCabe的控制图

- 右图为某个程序的程序图（或DD-路径图）
- 节点B和C是由两个出口的循环，而且从B到E的边是跳入节点D、E、F中的if-else-then语句的分支
- 这段程序有单入口（A）和单出口（G）

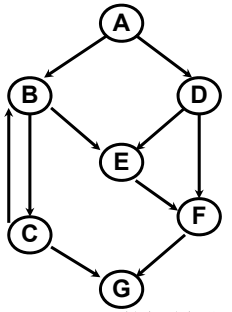
■线性独立路径数是：

V(G)=e-n+2p=10-7+2*1=5

■e:边数

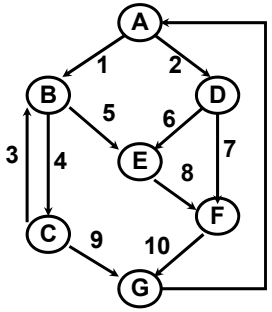
■n:节点数

■p:连接区域数



McCabe的控制图

McCabe的强连通图

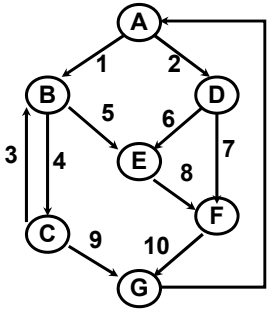


■ 通过从汇节点到源节点添加一条边，永远都可以创建强连接图

■线性独立环路的数量：V(G)=e-n+p

=11-7+1=5

McCabe的基路径方法



■由于圈复杂度为5，所以有5条独立路径

■用节点序列表示的路径：

p1：A，B，C，G

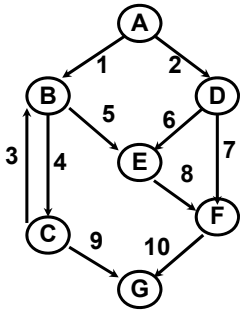
p2：A，B，C，B，C，G

p3：A，B，E，F，G

p4：A，D，E，F，G

p5：A，D，F，G

McCabe的路径线性组合



■路径加法：一条路径后接另一条路径

■路径乘法：路径的重复

■例如：

- ■路径A、B、C、B、E、F、G是基和p2+p3-p1
- ■路径A、B、C、B、C、B、C、G是线性合并2p2-p1

■通加法与乘法，可得到程序路径的向量空间

■注意：而减法则只有数学上去除边的含义，而缺少实际意义

McCabe的路径/边的关联矩阵

- 表中的条目是按照遍历的路径和所经历的边确定的
- 路径P1经过边1、4、9，路径P2经过边序列1、4、3、4、9
- 由于边4被路径P2经过了两次，2就是边4列的条目

路径的线性组合

所经过的路径/边	1	2	3	4	5	6	7	8	9	10
p1：A，B，C，G	1	0	0	1	0	0	0	0	1	0
p2：A，B，C，B，C，G	1	0	1	2	0	0	0	0	1	0
p3：A，B，E，F，G	1	0	0	0	1	0	0	1	0	1
p4：A，D，E，F，G	0	1	0	0	0	1	0	1	0	1
p5：A，D，F，G	0	1	0	0	0	0	1	0	0	1
Ex1: A，B，C，B，E，F，G	1	0	1	1	1	0	0	1	0	1
Ex2: A，B，C，B，C，B，C，G	1	0	2	3	0	0	0	0	1	0

- 通过观察这个关联矩阵的前五行可检查路径P1-P5的依赖关系
- 红色粗体显示的条目表示只出现在一条路径中的边，因此路径P2-P5必须是独立的

McCabe的算法—基线方法

■McCabe的算法，用于确定基路径集合

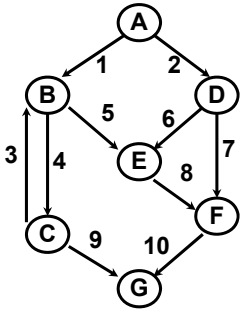
■步骤1：首先从DD-路径图的首节点出发，到末节点结束，寻找一条最长的路径记录下来，称为基线路径

■步骤2：然后对于基线路径中出现分支结构的节点转移到另外一条路径由此产生新路径

■步骤3：重复步骤2的过程直到没有新路径加入，即所有判断节点都“翻转”一次

■步骤4：分析所产生的所有路径，排除那些没有可能的路径，剩余的就是基路径

McCabe的基路径算法举例



- 步骤1、首先从DD-路径图的首节点出发，到末节点结束，寻找一条最长的路径记录下来

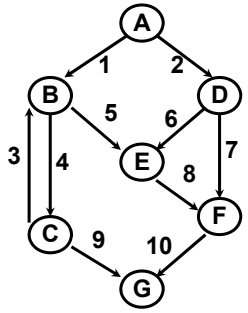
1. A、B、C、B、E、F、G的路径为基线

- 步骤2、然后对于出现分支结构的节点转移到另外一条路径由此产生新路径

这条路径上的第一个判断节点（外度>=2）是节点A，因此对于下一个基路径，要经过边2，而不是边1；

2. A、D、E、F、G

McCabe的基路径算法举例



- 对于下一条路径，可以选第二条路径，取节点D的另一个判断分支，得到路径：

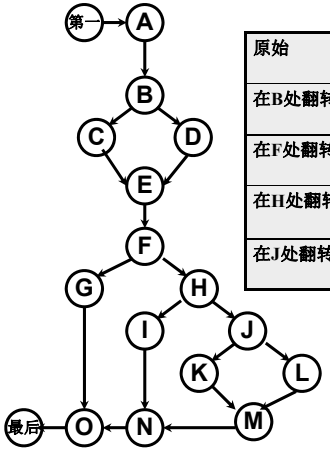
3. A、D、F、G

- 对B、C进行翻转，得到如下2条路径：

4. A、B、E、F、G
5. A、B、C、G

- 这组基路径与上表的基路径集合不同：这没有问题，因为不要求唯一基

三角形问题应用McCabe的基路径方法



原始	p1:A-B-C-E-F-H-J-K-M-N-O-最后	不等边三角形
在B处翻转p1	p2:A-B-D-E-F-H-J-K-M-N-O-最后	不可行
在F处翻转p1	p3:A-B-C-E-F-G-O-最后	不可行
在H处翻转p1	p4:A-B-C-E-F-H-I-N-O-最后	等边三角形
在J处翻转p1	p5:A-B-C-E-F-H-J-L-M-N-O-最后	等腰三角形

三角形问题应用McCabe方法的问题

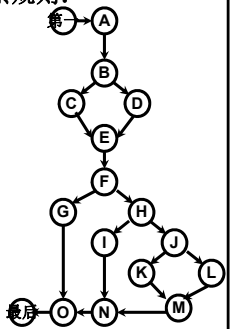
- 路径P2不可行，因为通过节点D意味着这些边不构成三角形，因此节点F的判断结果一定是节点G
- 在P3中，通过节点C意味着这些边确实会构成三角形，因此节点G不会经过
- 这种依赖关系与独立基路径的隐含假设发生冲突
- McCabe的过程成功地标识了在拓扑结构上独立的基路径，但是如果存在矛盾的语义依赖关系，拓扑结构上可行的路径在逻辑上有可能不可行
- 解决方案1：要求永远翻转语义可行路径中的判断结果
- 解决方案2：找出逻辑依赖性的原因

三角形问题应用McCabe的方法解决方法

如果仔细考虑到这个问题，可以找出两条规则：

- 如果经过节点C，则必须经过节点H
- 如果经过节点D，则必须经过节点G

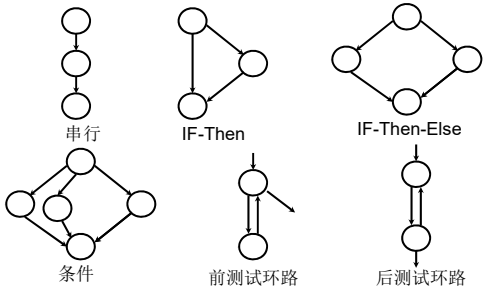
p1:A-B-C-E-F-H-J-K-M-N-O-最后	不等边三角形
p6:A-B-D-E-F-G-O-最后	非三角形
p4:A-B-C-E-F-H-I-N-O-最后	等边三角形
p5:A-B-C-E-F-H-J-L-M-N-O-最后	等腰三角形



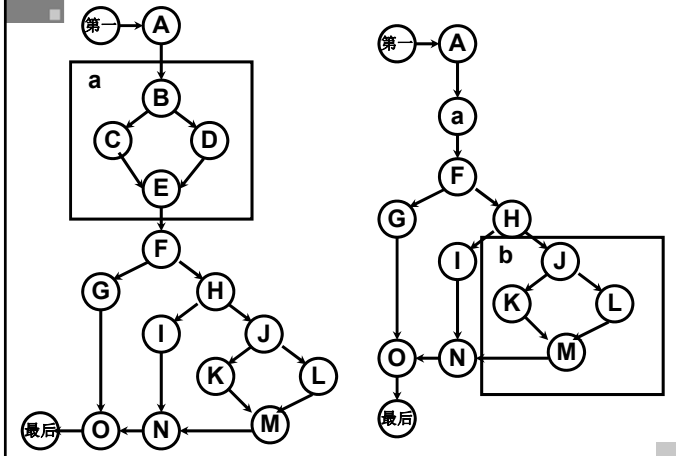
基路径覆盖可保证能够经过所有决策分支，与DD-路径覆盖相同

基本复杂度

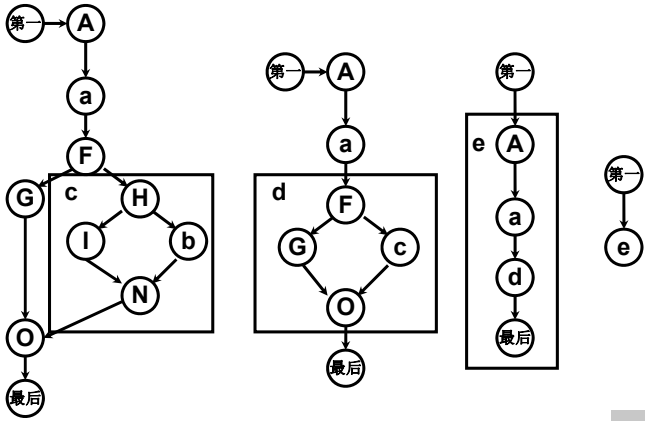
- 基本思想：寻找一种结构化程序设计构造图，将其压缩成单一的节点，重复这种处理，直到不能再找出其他结构化程序设计构造为止
- 如果程序具有很好的结构性，则总是可以压缩为只有一条路径的图，即圈复杂度为1的压缩图



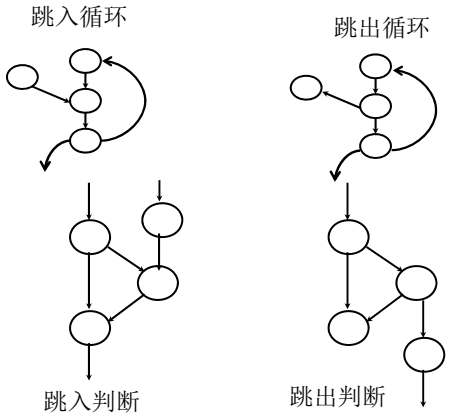
三角形问题结构化程序设计构造压缩



三角形问题结构化程序设计构造压缩



与结构化程序设计的冲突

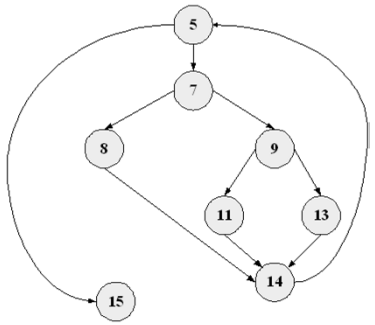


非结构化程序的测试

- 每一个“非结构化”程序设计包含三个不同的路径，而相应的结构化程序设计包含两个路径
- **冲突增加圈复杂度**
- **测试人员的基本原则是：具有高圈复杂度的程序需要更充分的测试**
- **采用圈复杂度指标的机构，大多数都确定了某种最大可接受复杂度指导方针，一般都选择V(G)=10**
- **如果单元具有更高的复杂度该怎么办？有两种可能：要么简化单元、要么计划更充分的测试**
  - **如果单元结构良好，则基本复杂度为1，因此可以很容易简化**
  - **如果单元的基本复杂度超过了指导方针规定，则最好的选择常常是解决非结构化问题**

练习－给出基路径

```
1 public void Sort(int iRecordNum,int iType)
2 {
3   int x=0;
4   int y=0;
5   while (iRecordNum--)
6   {
7     if (iType == 0)
8       x = y+2;
9     else
10      if (iType == 1)
11        x = y+10;
12      else
13        x = y+20;
14  }
15 }
```



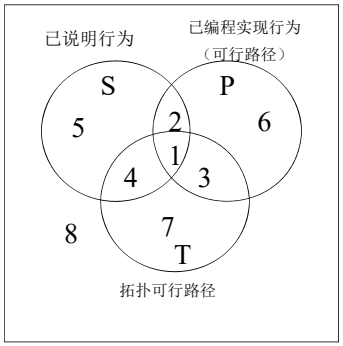
主要内容

- DD-路径
- 测试覆盖指标
- 基路径测试
- 指导方针和观察

指导方针与观察

- **基路径测试给出了必须进行的测试的下限**
- **如果发现同一条程序路径被多个功能性测试性用例遍历，就可以怀疑这种冗余不会发现新的缺陷**
- **如果没有达到一定的DD-路径覆盖，则可以知道在功能性测试用例中存在漏洞**
- **利用源代码的性质标识合适的覆盖指标，然后再使用这些指标交叉检查功能性测试用例**

可行的和在拓扑结构上可能的路径



- 已描述行为(集合S)、已编程行为(集合P)和程序中在拓扑结构上可行的路径(集合T)
- 通常区域1是最需要的，因为它包含由可行路径实现的已描述行为
- 根据定义，所有可行路径在拓扑结构上都是可行的，因此集合P区域2和6必须为空
- 区域3包含对应未描述行为的可行路径。需要检查这种额外功能：如果这些功能有用，则应该修改规格说明，否则应该删除这些可行路径
- 区域4和7包含不可行路径。其中区域4是有疑问的。区域4指未被实现的已描述行为，即拓扑结构可能但是不可行的程序路径。这种区域通常可能对应代码错误，需要通过修改使路径可行
- 区域5仍然对应没有实现的已描述行为。基于路径的测试永远也不会发现这种区域
- 区域7很奇怪：未定义、不可行，但是从拓扑结构看是可能的路径。严格地说，这里不会出现问题，因为不会执行不可行路径。如果对应的代码被维护人员不正确地修改(也可能是没有充分理解该代码的程序员)，这些路径可能变为可行路径，与区域3一样

总结

- DD-路径是什么？
- 基路径测试的思想是什么？
- McCabe基路径测试方法是什么？

作业（可选）

- 查阅相关文献，写一篇关于基路径测试的研究报告
- 要求：正文字数不少于5000字
- 需要提交报告、相关文献
- 需要自己进行语言组织、不允许直接抄袭