Chapter 7

Structure Test

白盒测试

- 结构性测试的方法：*路径测试、数据流测试*等。
- 结构性测试力求提高*测试覆盖率*。
- 结构性测试主要用于软件验证。
  - “我们在正确地构造一个系统吗？”
- 功能性测试是一种确认技术。
  - “我们在构造一个正确的系统吗？”

黑盒测试与白盒测试的比较

- 黑盒测试：从用户观点出发，按规格说明书要求的输入数据与输出数据的对应关系设计测试用例。因此它 是*根据程序外部特征进行测试。*
- 白盒测试：*根据程序内部逻辑结构进行测试*。
- 这两类测试方法是从完全不同的起点出发，并且是两个完全对立的出发点。这两类方法各有侧重，在测试的实践中都是有效和实用的。*在进行单元测试时大都采用白盒测试，而在系统测试中采用黑盒测试*。

有了“黑盒”测试，为什么还要“白盒”测试？

- 黑盒测试是从用户的观点出发，根据程序外部特性进行的测试。如果外部特性本身有问题或规格说明的规定有误，用黑盒测试方法是发现不了的。
- 黑盒测试只能观察软件的外部表现，即使软件的输入输出都是正确的，却并不能说明软件就是正确的。因为程序有可能用错误的运算方式得出正确的结果，这种情况只有白盒测试才能发现真正的原因。
- 白盒测试能发现程序里的隐患，像内存泄漏、误差累计问题。在这方面，黑盒测试存在严重的不足。

7.1 路径测试

7.2 数据流测试

7.3 测试的效率

程序图

- 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 的语句片段之后立即执行。

三角形程序的程序流图

```
1. Program triangle2
2. Dim a,b,c as Integer
3. Dim IsATriangle As Boolean
4. OutPut("Enter 3 integer 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)
9. If (a < b + c) AND (b < a + c) AND (c < a + b)
10.    Then IsATriangle = True
11.    Else IsATriangle = False
12.EndIf
13.If IsATriangle
14.    Then If (a = b) AND (b = c)
15.          Then Output("等边三角形")
16.          Else If (a<>b) AND (a <> c) AND (b <> c)
17.                Then Output("一般三角形")
18.                 Else Output("等腰三角形")
19.          EndIf
20.     EndIf
21.Else Output("非三角形")
22.EndIf
23.End triangle2
```

三角形程序的程序流图

程序伪代码 （略）

(5分钟程序阅读)

程序流图：

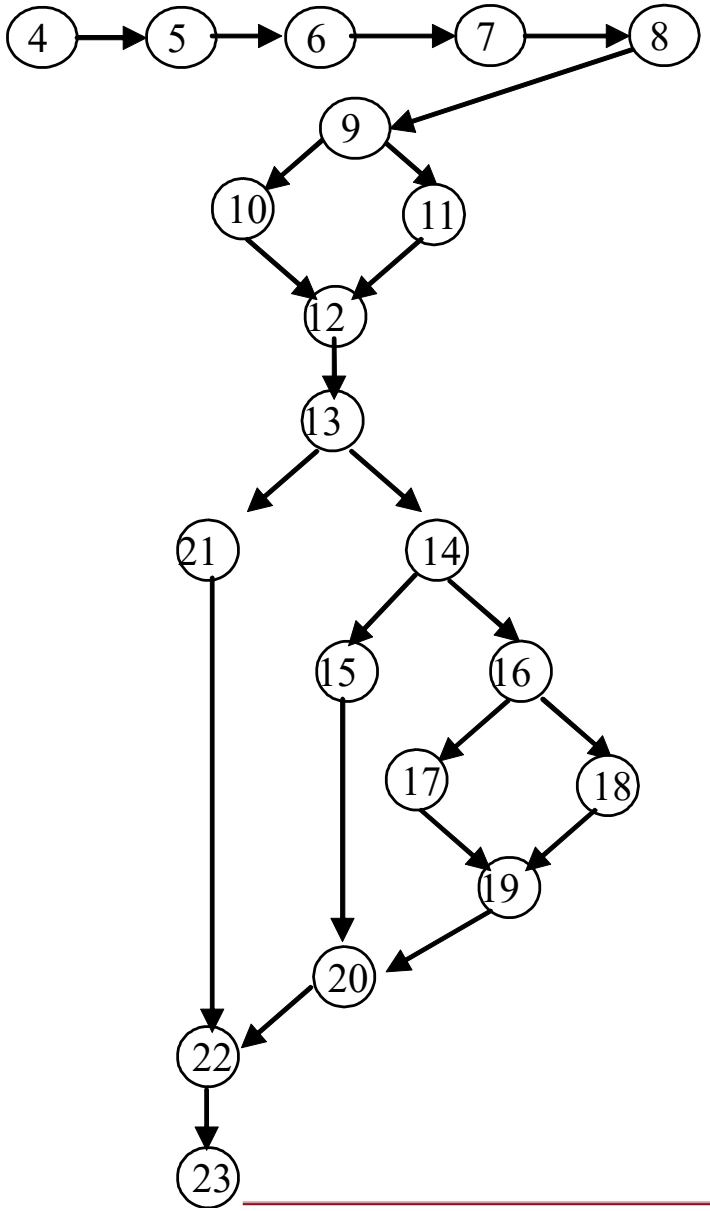
4~8是顺序执行;

9~12是分支语句;

13~22是分支的嵌套语句;

没有循环!

所以是有向无循环图.



7.1.1 DD-路径图（决策到决策）

程序流图是一种有向图，其中的节点表示语句，边表示控制流。

对于给定的程序，可以使用多种不同的程序流图，所有这些程序流图，都可以简化为唯一的DD-路径图。

还是以三角形程序为例进行说明。

DD-路径定义

DD-路径是程序图中的一条链，使得：

情况1：由一个节点组成，入度=0。

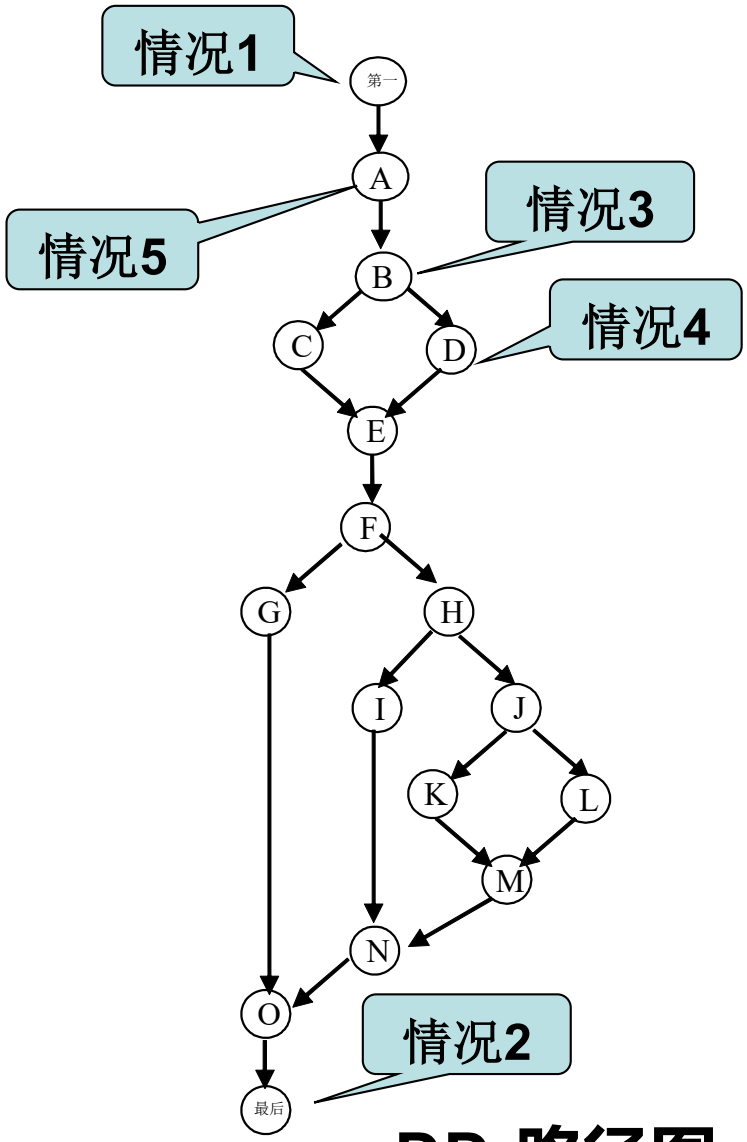
情况2：由一个节点组成，出度=0。

情况3：由一个节点组成，入度≥ 2或出度≥ 2。

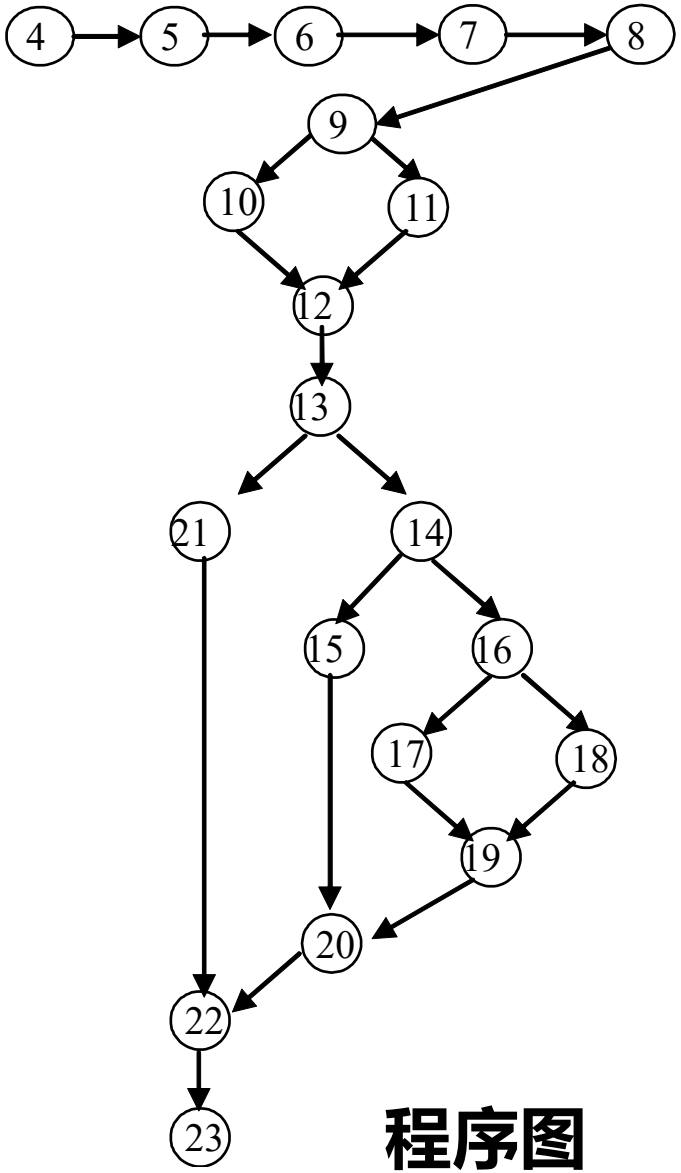
情况4：由一个节点组成，入度=1并且出度=1。

情况5：长度≥1的最大链(单入单出的最大序列)。

换言之：DD-路径是程序图中的最小独立路径，它不能被包括在其它DD-路径之中。


情况1
第一
A
情况5
情况3
B
C
D
情况4
E
F
G
H
I
J
K
L
M
N
O
情况2
最后

DD-路径图


4
5
6
7
8
9
10
11
12
13
21
14
15
16
17
18
19
20
22
23

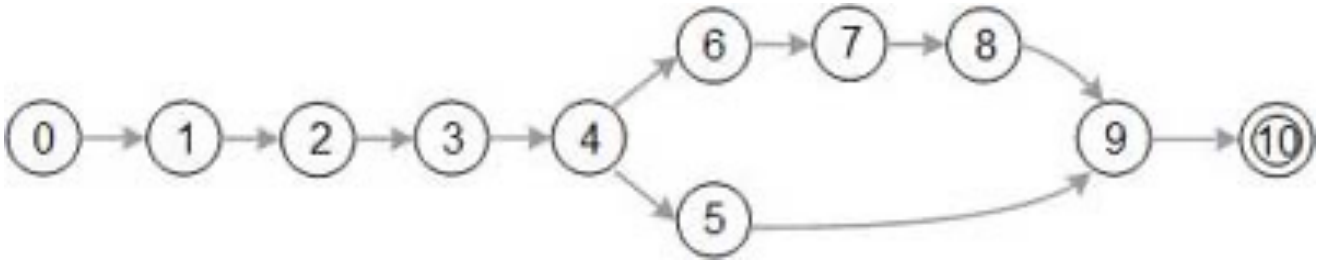
程序图

DD路径定义:对给定用命令语言编写的一段程序,其DD路径是有向图,其中,节点表示程序图的DD路径,边表示连续DD路径之间的控制流。

换言之，DD路径是一种压缩图，其中的一个节点（DD路径），对应程序中的语句片段。

对100行以内的程序生成DD路径是可行的，如果超过这个规模，一般需要有分析工具的支持。

课题练习



7.1.2 测试覆盖指标

- 测试的主要评测方法包括*覆盖和质量*。
- 测试覆盖是对测试完全程度的评测。测试覆盖是由测试需求和测试用例的覆盖或已执行代码的覆盖表示的。
- 覆盖指标提供了“测试的完全程度如何？”这一问题的答案。最常用的覆盖评测是基于需求的测试覆盖和基于代码的测试覆盖。

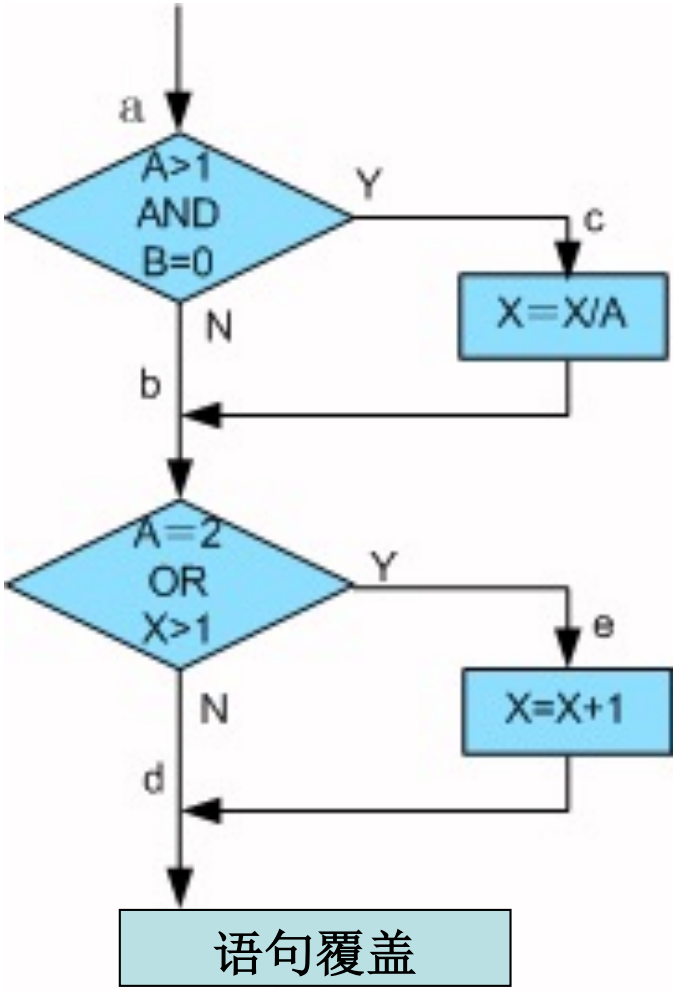
7.1.2 测试覆盖指标

表9-2

指标	覆盖描述
C_0	所有语句（语句覆盖）
C_1	所有DD-路径（分支覆盖）
C_1p	所有判断的每种分支（条件判断）
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_{start}	路径具有“统计重要性”的部分
C_∞	所有可能的执行路径

语句覆盖(C_0)

- 使程序中每一可执行语句至少执行一次；



为使程序中每个语句至少执行一次，只需设计一个能通过路径**ace**的例子就可以了，例如选择输入数据为：

A=2，B=0，X=3

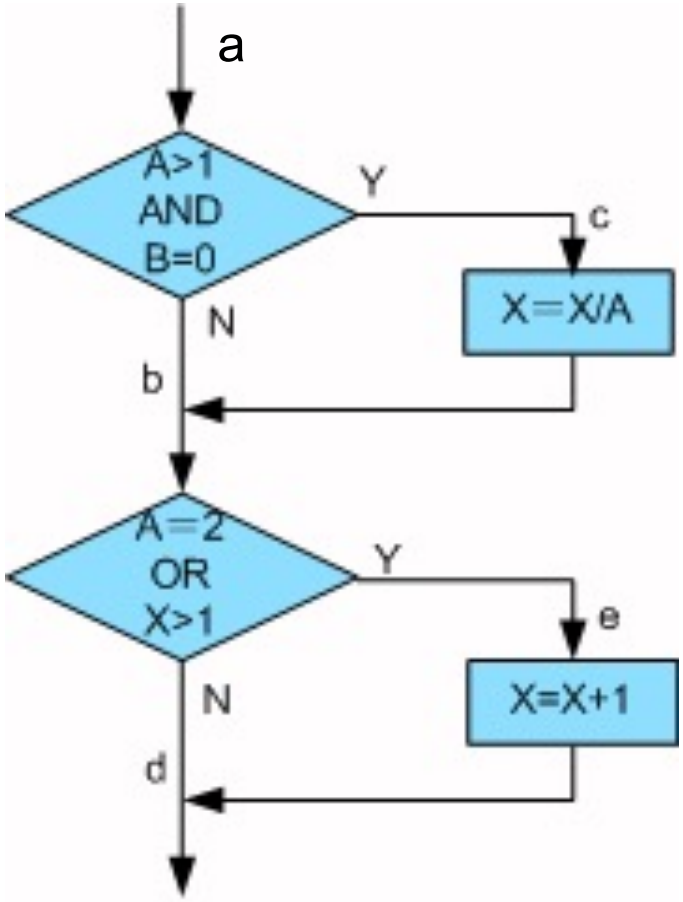
就可达到“语句覆盖”标准。

DD-路径测试

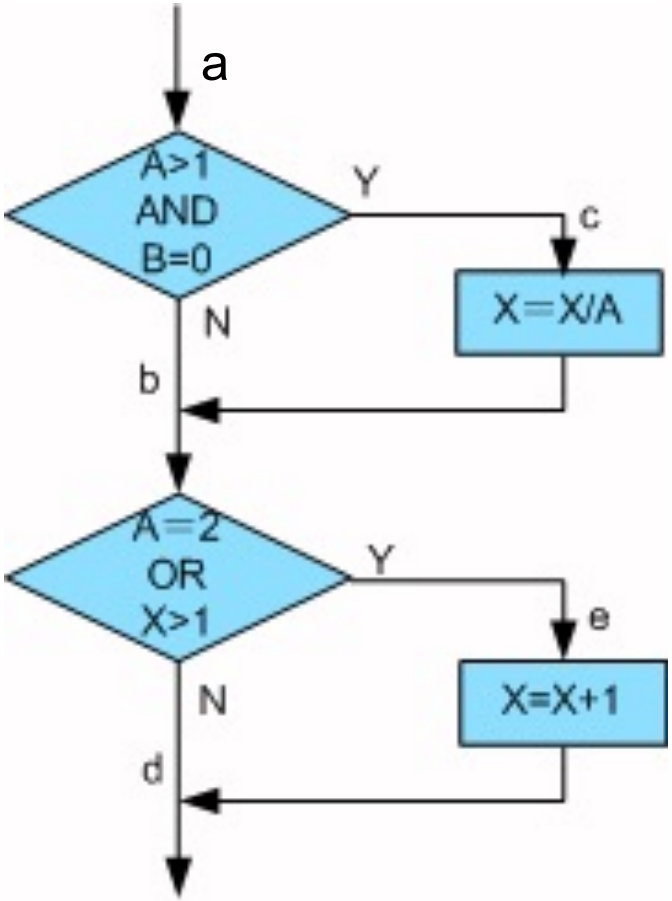
- **分支**覆盖：使程序中的每个逻辑判断的取真取假分支至少经历一次； **(C_1)**
- 条件覆盖：所有判断的每种分支（ **C_1p)**

分支覆盖(C_1)

A=3，B=0，X=1 (沿路径acd执行)
A=2，B=1，X=3 (沿路径abe执行)



分支覆盖



分支覆盖

条件覆盖（C_1p)

例1的程序有四个条件：

A＞1、 B=0、A=2、X＞1

为了达到“条件覆盖”标准，需要执行足够的测试用例使得在a点有：

A＞1、A≤1、B=0、B≠0

等各种结果出现，以及在b点有：

A=2、A≠2、X＞1、X≤1

等各种结果出现。

- 现在只需设计以下两个测试用例就可满足这一标准：

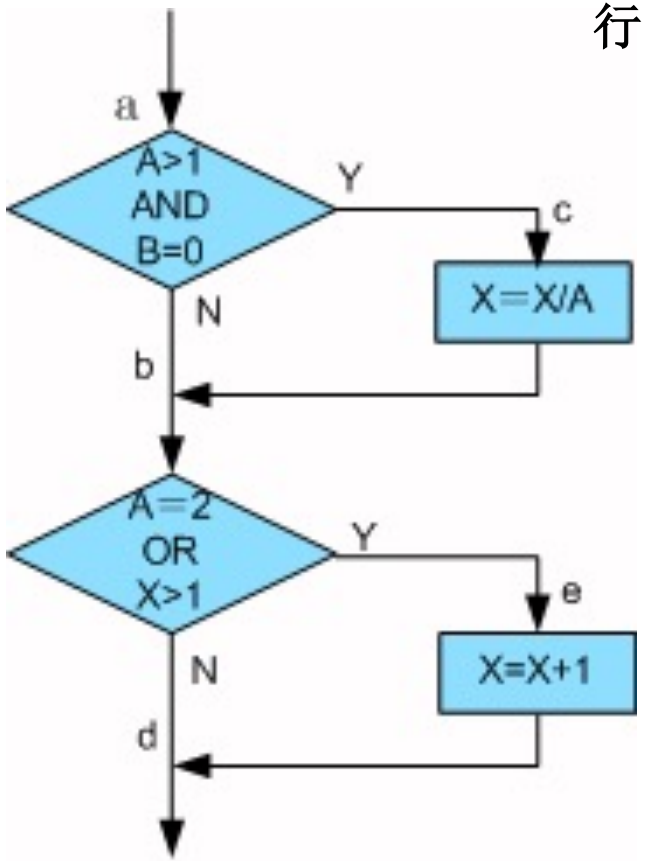
① A=2，B=0，X=4 (沿路径ace执行)；

② A=1，B=1，X=1 (沿路径abd执行)。

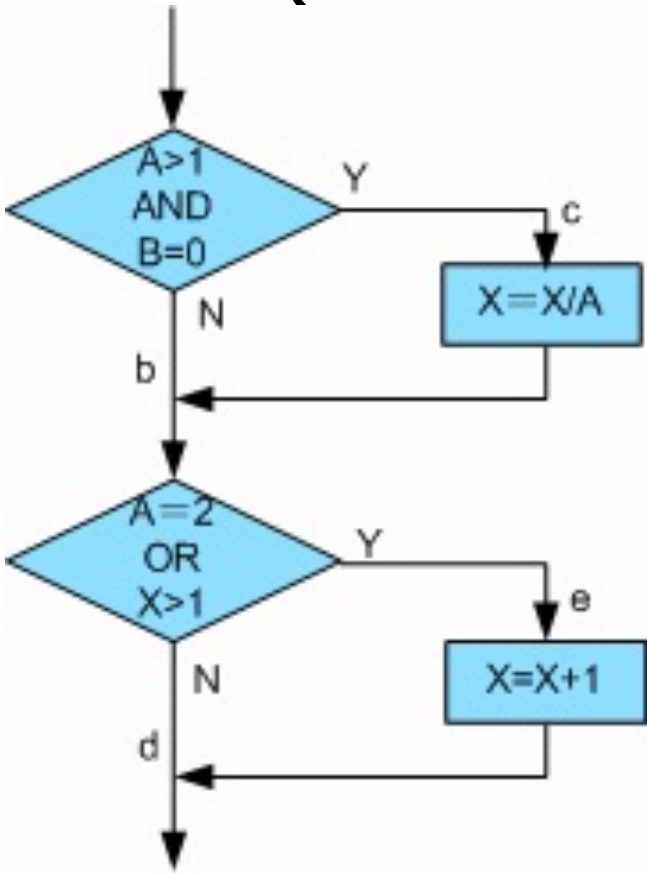
条件覆盖

A=2，B=0，X=4（沿路径ace执行）

A=1，B=1，X=1（沿路径abd执行）



条件覆盖

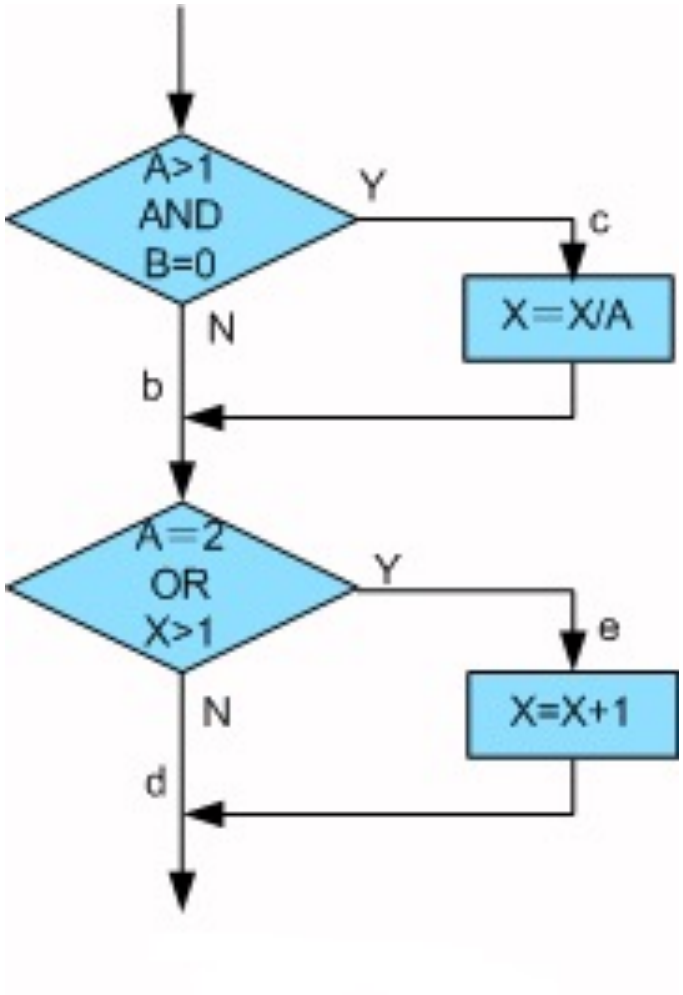


条件覆盖

多条件覆盖C_{mcc}

- 使得每个判断表达式中条件的各种可能组合都至少出现一次；(条件组合覆盖)

多条件覆盖(C_{mcc})



再看例1的程序，我们需要选择适当的例子，使得下面 8种条件组合都能够出现：

1）A>1, B=0　　2）A>1, B≠0

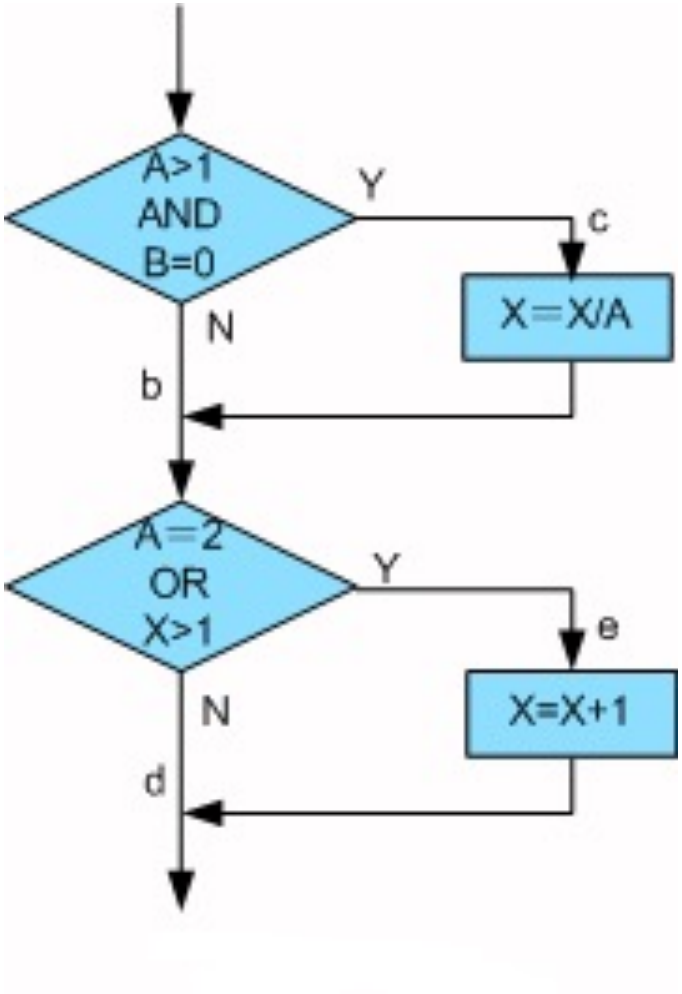
3）A≤1, B=0　　4）A≤1, B≠0

5）A=2, X>1　　6）A=2, X≤1

7）A≠2, X>1　　8）A≠2, X≤1

5）、 6）、 7）、8）四种情况是第二个 IF语句的条件组合，而X的值在该语句之前是要经过计算的，所以还必须根据程序的逻辑推算出在程序的入口点X的输入值应是什么。

多条件覆盖(C_{mcc})



下面设计的四个例子可以使上述 8种条件组合至少出现一次：

① A=2，B=0，X=4

使 1)、5)两种情况出现；

② A=2，B=1，X=1

使 2)、6)两种情况出现；

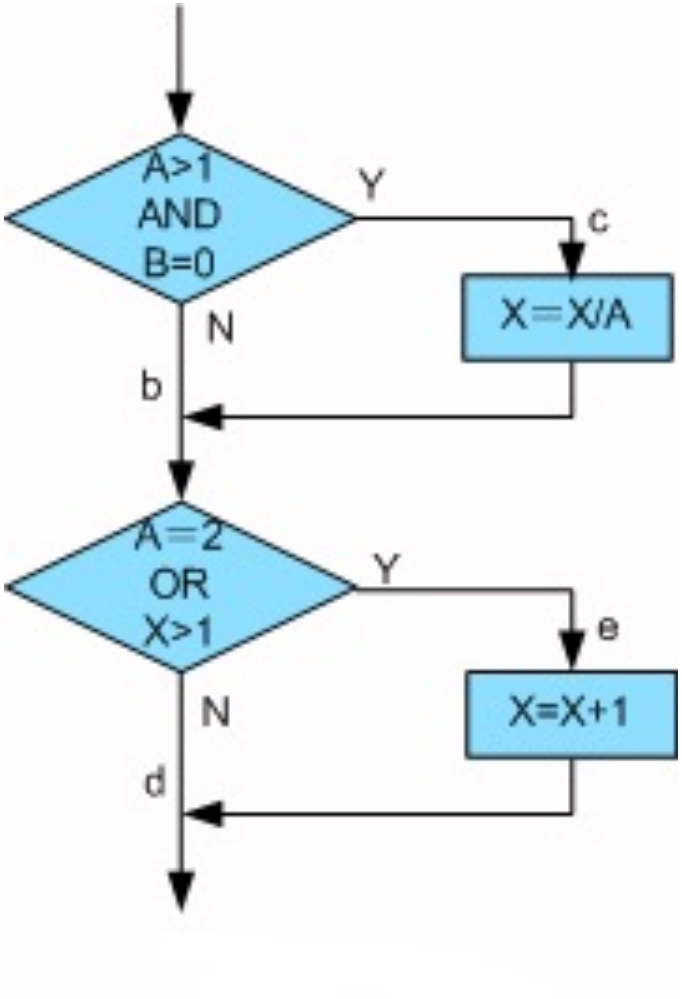
③ A=1，B=0，X=2

使 3)、7)两种情况出现；

④ A=1，B=1，X=1

使 4)、8)两种情况出现。

多条件覆盖(C_{mcc})



上面四个例子虽然满足条件组合覆盖，但并不能覆盖程序中的每一条路径，例如路径**acd**就没有执行，因此，条件组合覆盖标准仍然是不彻底。

分支/条件覆盖

- 针对上面的问题引出了另一种覆盖标准—“**分支 / 条件覆盖**”，它的含义是：*执行足够的测试用例，使得**分支**中每个条件取到各种可能的值，并使每个**分支**取到各种可能的结果。*

  —对例1的程序，前面的两个例子

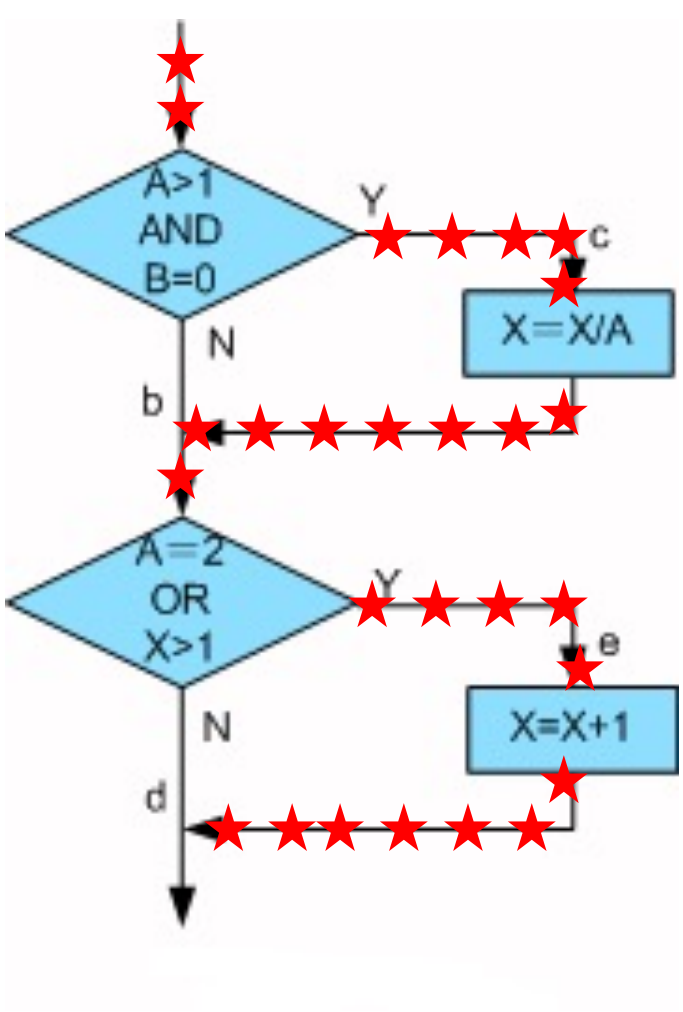
  ① A=2，B=0，X=4 (沿ace路径)

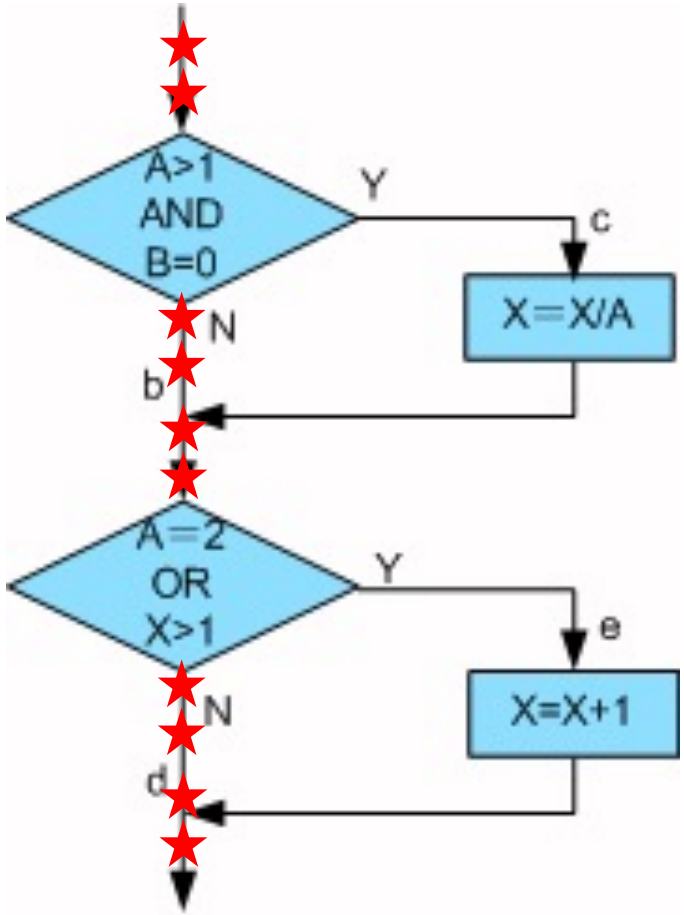
  ② A=1，B=1，X=1 (沿abd路径)

  是满足这一标准的。

① A=2，B=0，X=4 (沿ace路径)

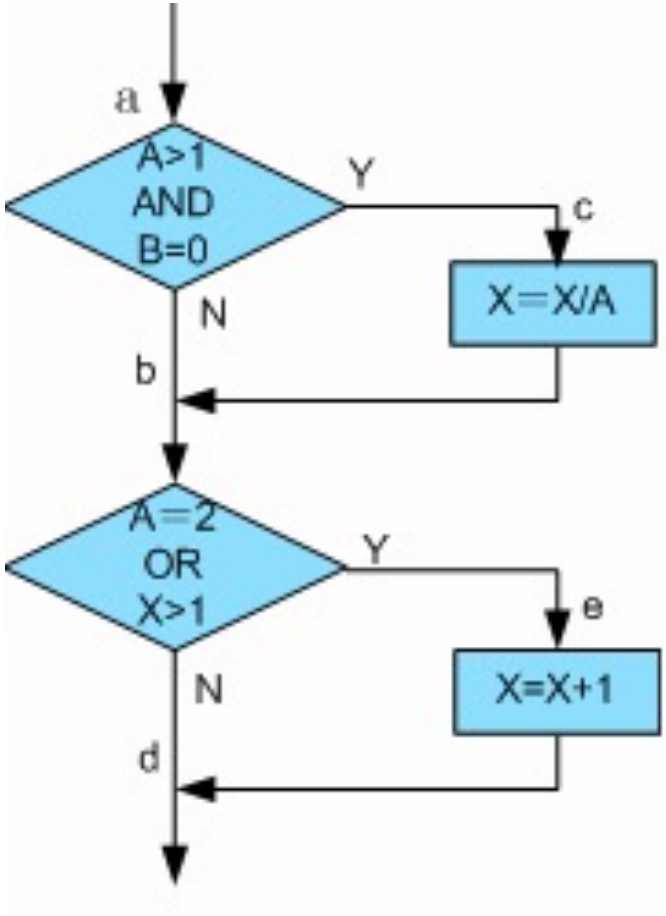
② A=1，B=1，X=1 (沿abd路径)





路径测试(C_∞)

- 路径测试就是设计足够多的测试用例，覆盖被测试对象中的所有可能路径。

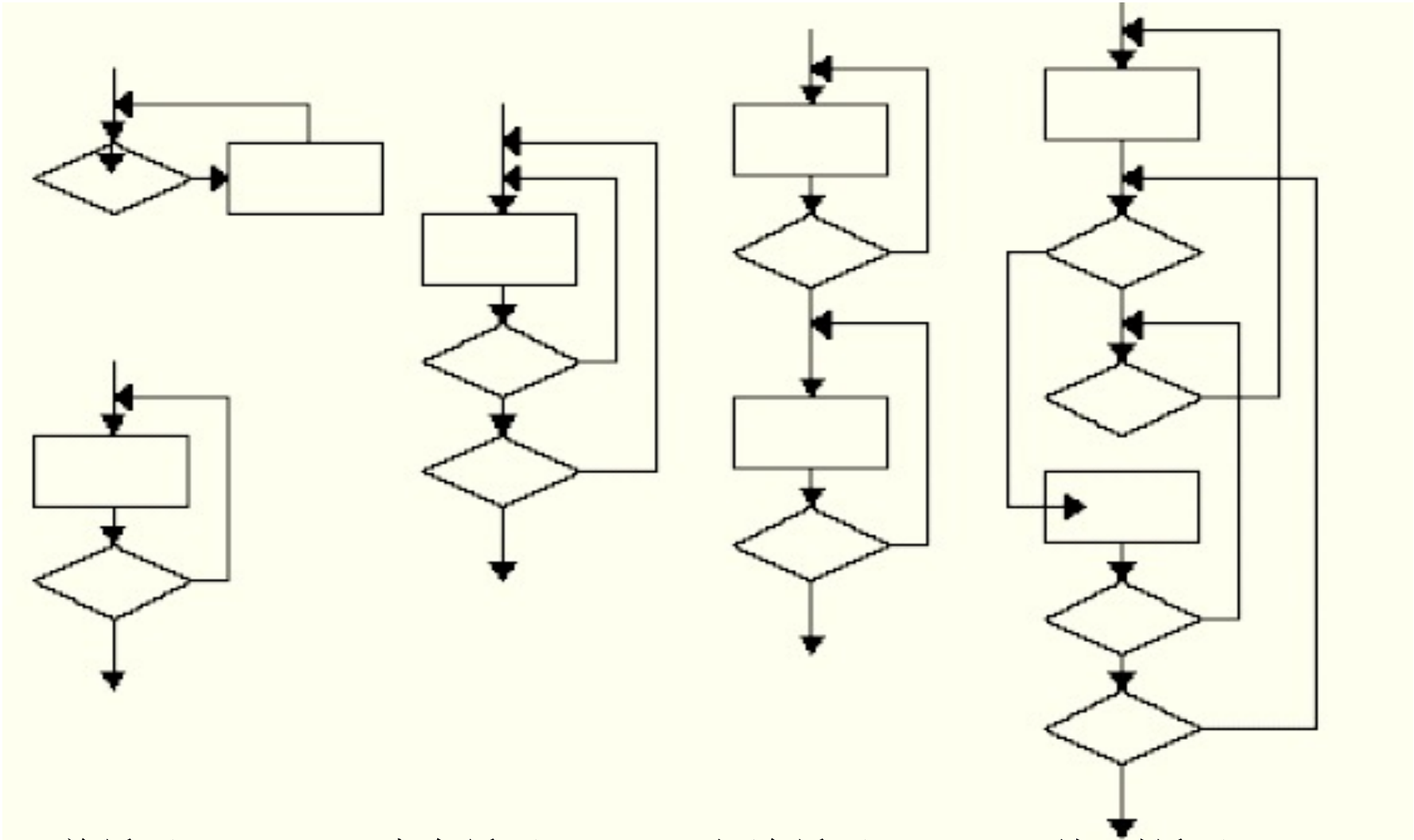


1. abd、abe
2. acd、ace

DD-路径的依赖对偶

- 数据流测试（第十章）

循环测试



单循环　　嵌套循环　　级连循环　　不规则循环

- 单循环测试

假设循环次数为N

a. 直接跳过循环

b. 循环次数为1

c. 循环次数为2

d. 循环次数为M，M<N

e. 循环次数为N-1，N，N+1

- 嵌套循环测试

a. 先测试最内部循环，其它循环次数为1

b. 测试第二层循环，其它循环次数为1，

c. 直到最外部循环完成测试

- 级连循环测试

a. 分别采用单循环测试方法进行测试

- 不规则循环测试　无法测试——重新设计！

小结

- 无论哪种测试覆盖，即使其覆盖率达到百分之百，都不能保证把所有隐藏的程序欠缺都揭露出来。
- 提高结构的测试覆盖率只能增强我们对被测软件的信心，但它绝不是万无一失的。

7.1.3 McCabe圈数

- 基路径的概念与方法

“基”：向量空间的概念，向量空间的基是相互独立的一组向量，基“覆盖”整个向量空间。使得该空间中的任何其它向量都可以用基向量表示。

基路径：程序图中相互独立的一组路径，使得该程序中的所有路径都可以用基路径表示。

圈复杂度

- 圈复杂度：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基本路径数目。
- 基本路径必须从起始点到终止点的路径。
- *包含一条其他基本路径不曾用到的边。或至少引入一个新处理语句或者新判断的程序通路。*
- *对于循环而言，基本路径应包含不执行循环和执行一次循环的路径*

McCabe圈数V（G）计算方法：

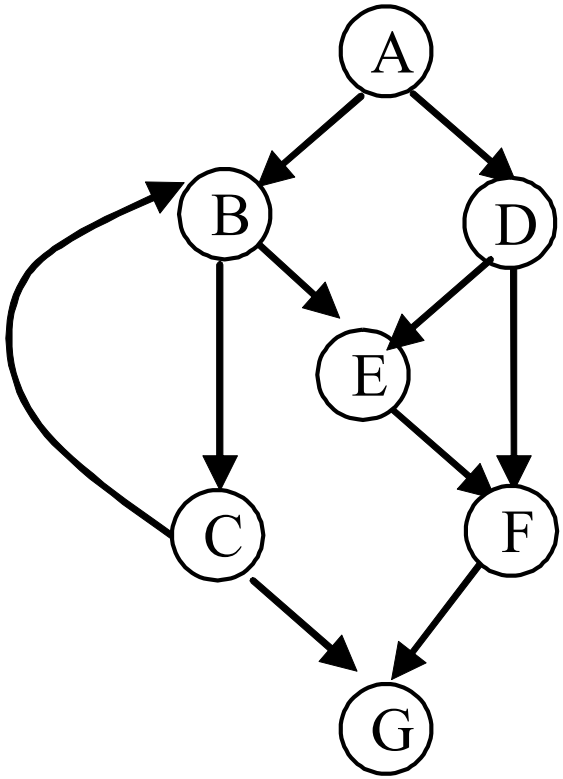
V（G）=e-n+2p

e：边数；

n：节点数

p：连接区域数

对于右图： V（G）=10-7+2p=5



为避免混乱，不再说明强连接有向图的计算公式(不满足单入口、单出口条件)。

- 5条基本路径是：

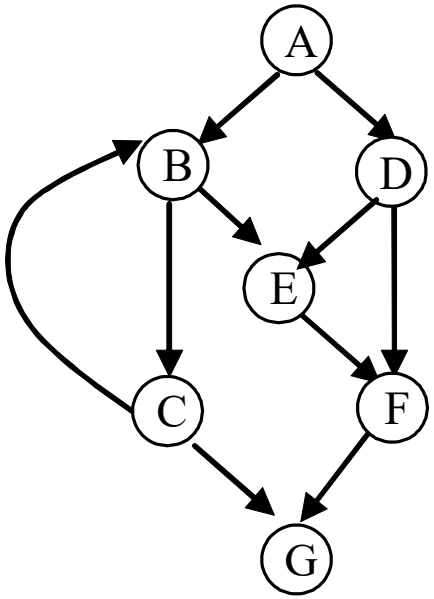
P1：A，B，C，G；

P2：A，B，C，B，C，G；

P3：A，B，E，F，G；

P4：A，D，E，F，G；

P5：A，D，F，G。



任何路径都可以看作是上述5条路径的组合形成的。

例：A、B、C、B、E、F、G是P2+P3-P1

其中，加法是一条路径后接一条路径；乘法是路径的重复，而减法则只有数学上去除边的含义，而缺少实际意义。

=A，B，C，B，C，G+ A，B，E，F，G- A，B，C，G；

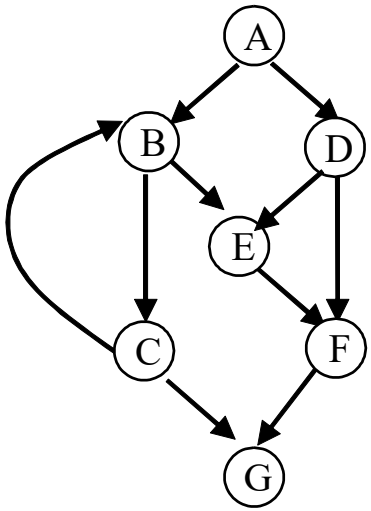
AB+BC+CB+~~BC+CG+AB~~+BE+EF+FG-~~AB-BC-CG~~

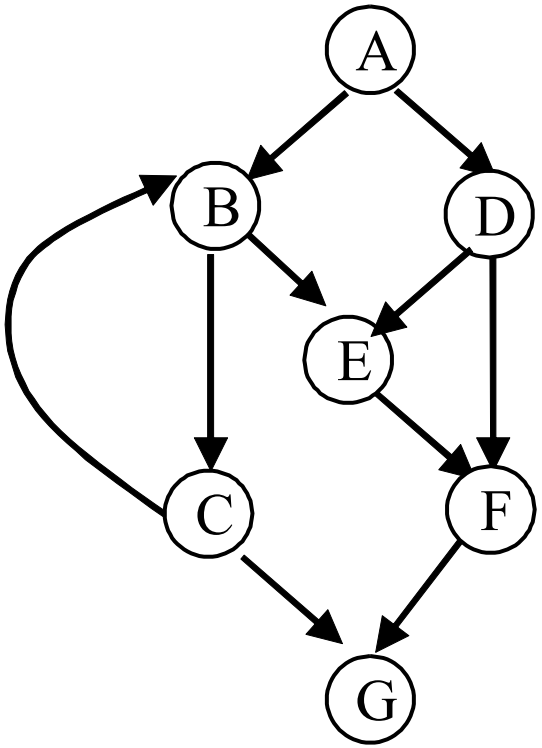
= A、B、C、B、E、F、G

如何寻找McCabe路径？

图的搜索与遍历算法！

- 深度优先算法
- 广度优先算法





基本复杂度

McCabe在圈复杂度上的工作,在一定意义上,对程序设计的改进要大于对测试的改进。

对于结构化程序设计的程序，可以将if-then-else构造压缩为一个节点。

则三角形程序最终可以压缩为只有一条路径的图！

而对于非结构化程序，必然增加McCabe圈的数量。

一般的，一个单元模块的最大复杂度V（G）<10。

7.1.4 基于路径的测试讨论

基于路径的测试，从另一个角度去考虑程序测试的完备性，但是没有考虑数据功能，即使满足DD-路径覆盖，和考虑圈复杂度指标，也仅是测试的下限指标。

如左图：S：已说明行为的路径集合；

P：已编程实现的路径集合；

T：拓扑可达的路径集合；

区域：1：测试的重点，可测；

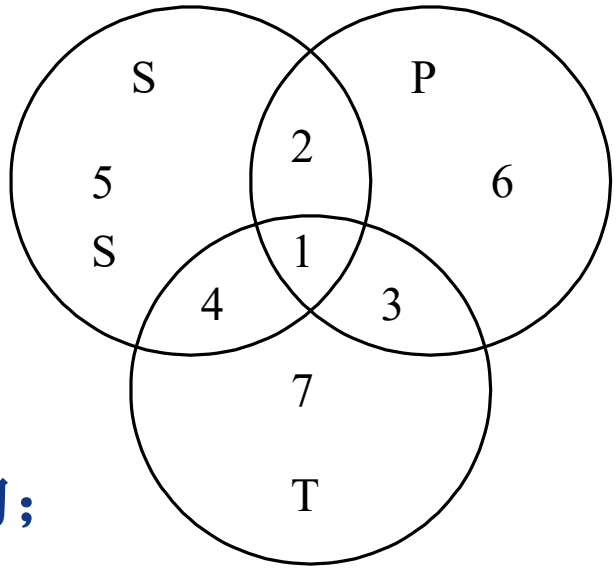
2、6：已实现的，

3：已实现和可达的，

4：已描述的，拓扑可达，却没有实现的；要增加，可测；

5：没有实现的已说明路径，不可测，

7：未说明，未实现，拓扑可达，测试无意义！



可行的和在拓扑结构上可能的路径

7.2 数据流测试

数据流测试是从关注程序中数据及其使用的角度，来设计测试用例的。类似一种路径测试覆盖，但关心的是数据变量而不是程序结构。

定义-使用（def-use）测试

1. 对于程序P，有程序图G（P），V是程序中变量的集合，节点表示语句片，边表示节点序列。

2. 节点n∈G（P）是变量v ∈V的定义节点，记做DEF（v，n），当且仅当变量v的值有对应节点n的语句片处定义。

输入语句、赋值语句、循环控制语句和过程调用语句，即：向内存地址写入值的语句。

3. 节点n ∈G（p）是变量v ∈V的使用节点，记做USE（v，n），当且仅当变量v的值在对应节点n的语句片处使用。

   输出语句、赋值语句、条件语句、循环控制语句和过程调用语句，都是可能的使用语句。但它们不改变变量的值。即：从内存地址读取值的语句。

4. 使用节点USE（v，n）是一个谓词使用（记做P-use），当且仅当语句n是谓词语句；否则， USE（v，n）是计算使用（记做C-use）。

5. 关于变量v的定义-使用路径（记做du-path）是PATHS（P）中的路径，使得对某个v ∈V，存在定义节点DEF（v，m）和和使用节点USE（v，n），使得m和n是该路径的起始和终止节点。

6. 定义：关于变量v的定义清除路径（记做dc-path），是具有起始和终止节点DEF（v，m）和USE（v，n）的PATHS（P）中的路径，使得该路径中没有其它节点是v的定义节点。

事例 佣金问题

```
7.  lockPrice=45.0
8.  stockPrice=30.0
9.  barrelPrice=25.0
10. totalLocks=0
11. totalStocks=0
12. totalBarrels=0

13. Input(Locks)
14. While NOT(Locks==-1)
15. Input(stocks,barrels)
16. totalLocks=totalLocks+Locks
17. totalStocks=totalStocks+stocks
18. totalBarrels=totalBarrels+barrels
19. Input(Locks)
20. ENDWhile
```

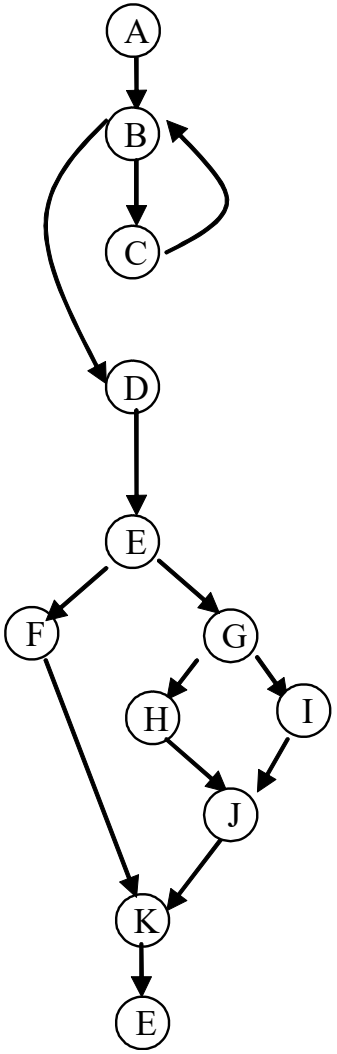
```
21. Output("Locks sold:",totalLocks)
22. Output("Stocks sold:",totalStocks)
23. Output("Barrels sold:",totalBarrels)
24. lockSales=lockPrice*totalLocks
25. stockSales=stockPrice*totalStocks
26. barrelSales=barrelPrice*totalBarrels
27. Sales=lockSales+stockSales+barrelDSales
28. Output("Total Sales: ",Sales)

29. If (sales>1800)
30. Then
31.   commission=0.10*1000.0
32.   commission=commission+0.15*800.0
33. Commission=commission+0.20*(sales-1800.0)
34. Elseif(sales>1000.0)
```

```
35.     Then
36.       commission=0.10*1000.0
37.       commission=commission+0.15*(sales-1000.0)
38.      Else
39.       commission=0.10*sales
40.      Endlf
41.  Endlf
42.  Output(“commission is $”,commission)
43.  End commission
```

变量名	定义节点	使用节点
lockPrice		
stockPrice		
barrelPrice		
totalLocks		
totalStock		
totalBarrels		
locks		
stocks		
barrels		
locksales		
stocksales		
barrelsales		
sales		
commission		

DD-路径



- 定义使用路径

1.变量stocks: 有DEF(stocks,15)和USE(stocks,17),

路径<15,17>是一个stocks的定义-使用路径,是定义清除路径.

2. 变量locks:有DEF(locks,13)、DEF（locks，19）USE（locks，14）、USE（locks，16）。

产生的路径：p1=<13,14>

p2=<13,14,15,16>

p3=<19,20,14>

p4=<19,20,14,15,16>

3.变量totalLocks:问题比较复杂,有2个定义节点,DEF(totalLocks,10)和DEF(totalLocks,16);3个使用节点,USE(totalLocks,16)、USE(totalLocks，21)和USE（totalLocks，24）。

有路径：

p1=<10,11,12,13,14,15,16> 是定义清除的。

p2=<10,11,12,13,14,15,16,17,18,19,20,14,21>因为节点16是可循环的,存在totalLocks再定义现象,不是定义清除的。

p3=<10,11,12,13,14,15,16,17,18,19,20,14,21,22,23,24>

=<p2,22,23,24> ,当然不是定义清除的.

p4=<16,16> 不作为定义-使用路径

p5=<16,17,18,19,20,14,21>

p6=<16,17,18,19,20,14,21,22,23,24>

其它变量？Sales?

定义-使用路径测试覆盖指标

- 全定义准则：每个定义节点到一个使用的定义清除路径。
- 全使用准则：每个定义节点到所有使用节点以及后续节点的定义清除路径。
- 全谓词使用/部分计算使用准则：每个定义节点到所有谓词使用的定义清除路径，若无谓词使用，至少有一个计算使用的定义清除路径。
- 全计算使用/部分谓词使用准则：每个定义节点到所有计算使用的定义清除路径，若无计算使用，至少有一个谓词使用的定义清除路径。
- 全定义-使用路径准则：每个定义节点到所有使用节点以及后续节点的定义清除路径。包括有一次环路和或无环路的路径。

数据流覆盖的层次结构关系图**9-5 P116**。

基于程序片的测试

定义：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中的变量值作出贡献的所有语句（编号）的集合。

USE（使用）的形式有：

谓词使用、计算使用、输出使用、定位使用、迭代使用

DEF（定义）的形式有：

输入定义、赋值定义

事例：佣金问题

- 变量Locks有2个使用节点14、16，2个定义节点13、19，则：

S1：S（Locks,13）={13}

S2：S（Locks,14）={13,14,19,20} 14？

S3：S（Locks,16）={13,14,19,20} 16？

S4：S（Locks,19）={13,14,19,20}

- 变量stocks和barrels要受循环变量locks的影响,所以

S5:S(Stocks,15) ={13,14,15,19,20}

S6:S(Stocks,17) ={13,14,15, 19,20}，17？

S7:S(Barrels,15) ={13,14,15,19,20}

S8:S(Barrels,18) ={13,14,15, 19,20}，18？

- 变量totalLocks,节点10是赋值定义,节点16既是赋值又是计算,且存在循环。而节点21和24分别是输出和计算节点。则：

S9：S（totalLocks,10）={10}

S10：S(totalLocks,16) ={10,13,14,16,19,20}

S11:S(totalLocks,21)=S(totalLocks,24)=S10

- 变量totalStocks、totalBarrels与totalLocks相似，不再说明。

- 变量lockPrice，stockPrice、barrelPrice都是1个定义，1个使用，不涉及循环，不多说明。
- sales变量只有1个定义节点27，有28、29、33、34、37、38共6个使用节点，但影响其值的变量是除commission以外的所有变量。所以，

sales变量在所有相关语句上的程序片都一样：

$S_{24\text{-}30}$：

={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}

- 变量commission有6个定义节点，4个使用节点。所以

S_{31}：S（commission，31）={7-20，24-31}

S_{32}：S（commission，32）={7-20，24-32}

S_{33}：包含所有变量的影响，S(commission,33）={7~20，24-33}

S_{34}：S（commission，36）={7-20，24-36}

S_{35}：S（commission，37）={7-20，24~37}

S_{36}：S（commission，39）={7-20，24~39}

S_{37}：S（commission，42）={7-2-，24-39}

程序片的方法,将程序进行了分段划分。就如佣金问题的**commission**是最终结果，它的计算依赖于**sales**的计算；而**sales**的计算，又依赖于**lockSales**、**stockSales**、**barrelSales**的计算；而**lockSales**、**stockSales**、**barrelSales**的计算又分别依赖于**totalLocks**、**totalStocks**、**totalbarrels**的计算等等。

如果所有变量的定义和使用是正确的，当然程序就是正确的！

- 不同变量程序片的补集，为变量的分段划分提供了方法。

如：S（commission，42）-S（sales，28）

={7-39} — {7-27}

={28~39}；

将commission的计算分为sales计算和之后的计算，为错误的定位提供了依据。

7.3 测试的效率

什么时候测试可以停止?多少是足够的测试?

1. 当时间用完时——缺少标准
2. 当继续测试没有产生新失效时
3. 当继续测试没有发现新缺陷时

基于经验的有效方法

4. 当无法考虑新测试用例时—原因?
5. 当回报很小时—基于分析的方法
6. 当达到所要求的覆盖时—结构化测试的指标
7. 当所有缺陷都已经清除时—难以实现

7.3.1 漏洞与冗余

以三角形程序为例

我们可以看到路径共有11条!

P1:1-2-3-4-5-6-7-13-16-18-20

P2:

P3:

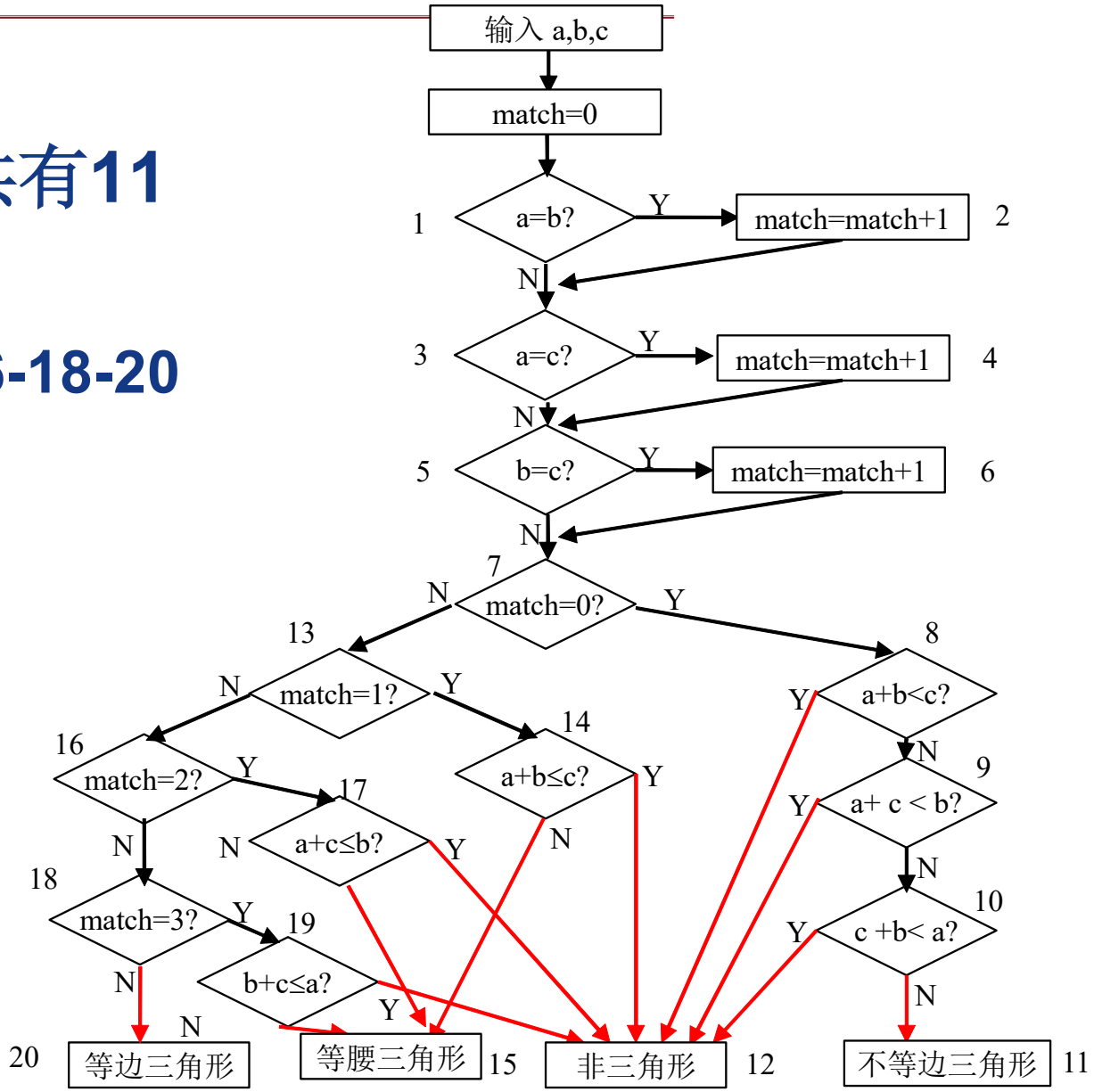
P4:

……

……

P10:1-3-5-7-8-9-10-12

P11:1-3-5-7-8-9-10-11



现在,如果使用边界测试法,共产生? 个测试用例,这里仅给出一组!

表10-1（P129）

测试用例	a	b	c	预期输出	路径
1	100	100	1	等腰三角形	P6
2	100	100	2	等腰三角形	P6
3	100	100	100	等边三角形	P1
4	100	100	199	等腰三角形	P6
5	100	100	200	非三角形	P7
…	…	…	…	…	…

- 测试覆盖的路径有:

p1、p2、p3、p4、p5、p6、p7。

- 如果采用最坏情况测试，测试用例为$5^3=125$个，能够覆盖全部11条路径，但是冗余很多！
- 决策表测试?

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
边界值	1	3	1	3	1	3	1	0	0	0	0
最坏情况	5	12	6	11	6	16	7	17	18	19	12

7.3.2 用于方法评估的指标

假设功能性测试技术M生成m个测试用例,并且根据标识被测单元中的s个元素的结构性测试指标S来跟踪这些测试用例.当执行m个测试用例时,会经过n个结构性测试单元。

定义:方法M关于指标S的覆盖是:n与s的比值,记做C(M,S)。

定义：方法M关于指标S的冗余是m与s的比值，记做R(M,S)

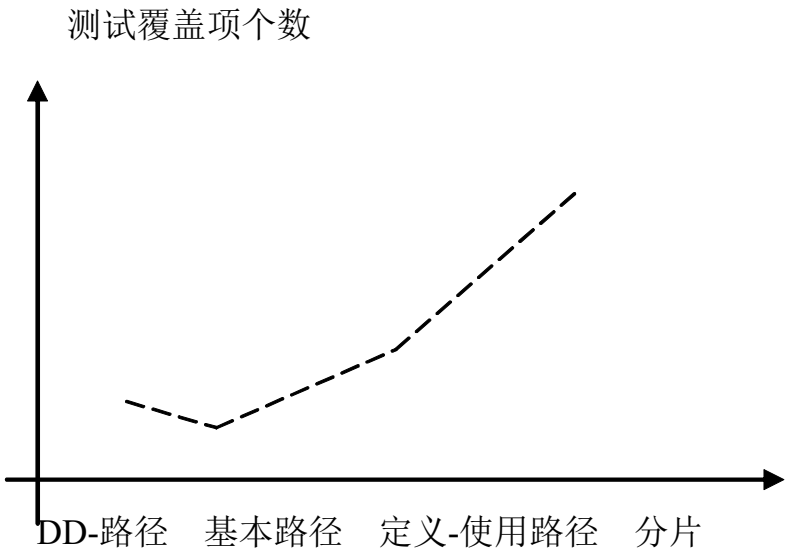
定义:方法M关于指标S的净冗余是m与n的比值，记做NR(M,S)。

三角形程序的指标（P133）

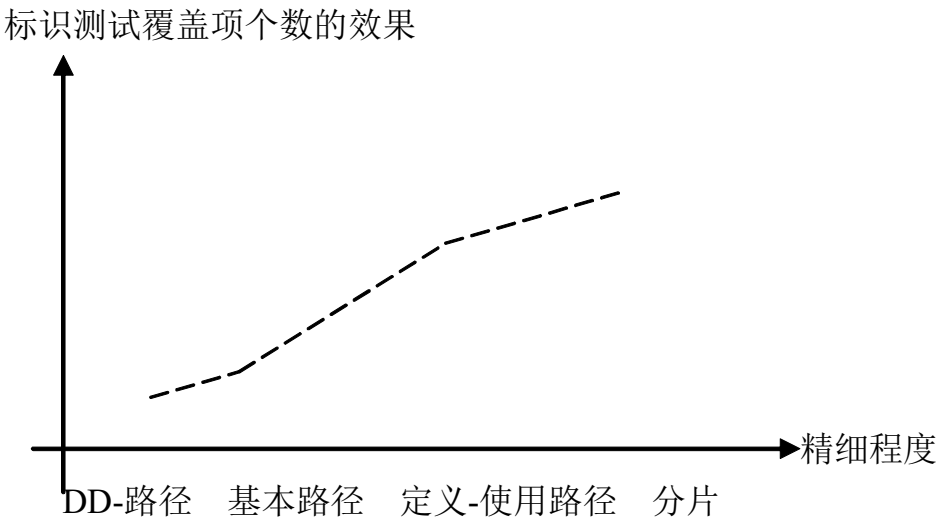
方法	m	n	s	C(M,S)=n/s	R(M,S)=m/s	NR(M,S)=m/n
一般测试	13	7	11	0.64	1.18	1.86
最坏情况	125	11	11	1.0	11.36	11.36
目标	s	s	s	1.0	1.0	1.0

佣金问题的指标

方法	m	n	s	C(M,S)=n/s	R(M,S)=m/s
边界值	25	11	11	1	2.27
决策表	3	11	11	1	0.27
DD-路径	25	11	11	1	2.27
定义—使用路径	25	33	33	1	0.76
程序片	25	40	40	1	0.63



测试覆盖项的趋势



测试方法作用的趋势

问题

- 除了结构化的覆盖率指标外,还有其它的覆盖率指标吗?