白盒测试

- 逻辑覆盖与路径测试
- 基本路径测试（参考 书第9章）
- 数据流测试（参考 书第10章）

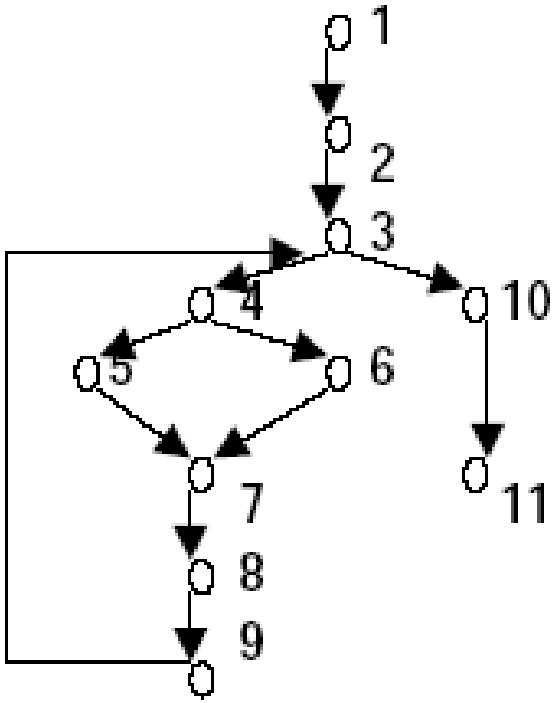
- 数据流问题
  - 如果程序中某一语句的执行能改变程序变量V的值，则称该语句定义变量V。
    - 输入定义
    - 赋值定义
  - 如果一语句的执行使用了内存中变量V的值，则称该语句使用变量V。
    - 谓词（判定）使用
    - 计算使用
    - 输出使用
    - …
  - 如：x = y + z

- 六七十年代，早期的数据流分析
- 关注 定义/使用异常
  - 变量被定义，但是从来未被使用。
  - 变量在被使用之前，没有被定义。
  - 变量在被使用之前，被定义了两次。
- 编译技术：
  - 建立程序的索引表
  - 静态/动态测试？

例：变量的定义和使用异常的例程

```
1. int X = 0, Y = 0, Z = 0;
2. X = W+ X;
3. while ( X > Y) {
4.   if (Y == Z)
5.         Y = V + Y;
6.   else
           Z = V – Z;
7.    V = 3 * X;
8.    W = Y / 2;
9.    Z = V – 8;
      }
10. Z = Z * 2;
11. printf("The value of Z is d%", Z);
```

- 此程序的控制流图，以及其中语句定义和使用的变量。



节点	被定义变量	被引用变量
1	X,Y,Z	
2	X	W,X
3		X,Y
4		Y,Z
5	Y	V,Y
6	Z	V,Z
7	V	X
8	W	Y
9	Z	V
10	Z	Z
11		Z

程序中含有两个错误：

（1）在语句2中，使用了未定义变量W。

（2）在语句5，6中，使用了未定义变量V。

两个异常：

（1）语句6, 9 对Z定义了两次，（因此：语句6对Z的定义从未使用过）。

（2）语句8对W的定义从未使用过。

数据流测试

- 是一种路径测试
  - 与“数据流图”无关
  - 关注 变量的“定义－使用”路径
  - 设计测试用例，覆盖这些相关路径或相关语句
    - 定义－使用*(Define-Use)*测试
    - 基于程序片*(Slice)*的测试
- 形式化工作： 20世纪80年代初完成
- 原型工具

定义—使用测试

记：程序P的程序图为G（P），其路径集合为PATHS（P），V是P的一组程序变量 。给出如下的形式化定义：

- 变量v ∈ V的定义结点n∈G(P) ， 记做DEF(v,n),当且仅当变量v的值在对应结点n的语句片段处定义。
- 变量v ∈ V的使用结点n∈G(P)， 记做USE(v,n), 当且仅当变量v的值在对应结点n的语句片段处使用。
  - 使用结点 USE(v,n) 是一个谓词使用（记做P-use）， 当且仅当语句n是判定语句；
  - 否则， USE(v,n) 是计算使用（记做C-use）。

定义—使用测试

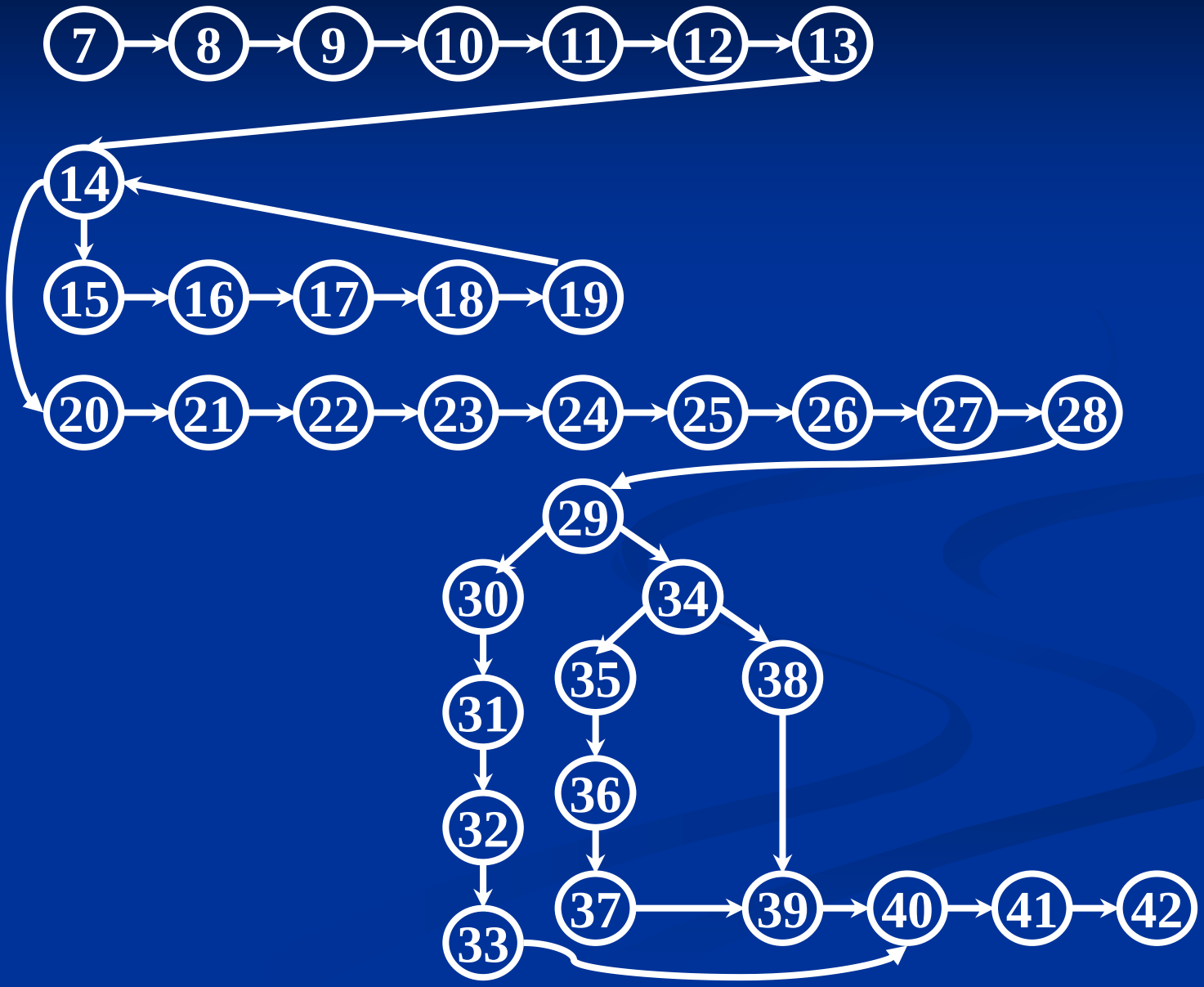
记：程序P的程序图为G（P），其路径集合为PATHS（P），V是P的一组程序变量 。给出如下的形式化定义：

- 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，存在定义结点DEF（v，m）和使用结点和USE（v，n），分别是该路径的始结点和终结点。
- 关于变量v的定义清晰路径（记做dc-path），是PATHS（P)中具有始结点DEF（v，m）和终结点USE（v，n）的的路径，使得该路径中没有其它的结点是v的定义结点。
  - “非定义清晰路径”的定义—使用路径，是有潜在问题的地方
    - 有问题： 定义—定义—使用
    - 无问题：定义—使用—定义—使用

例：定义/使用测试——佣金问题

- 程序：program commission.doc
- 程序图：

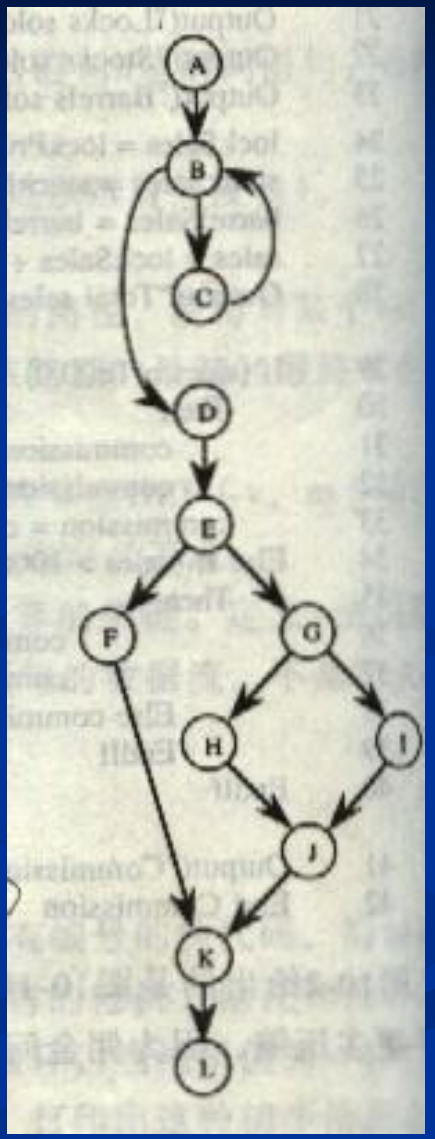
定义—使用测试：佣金问题的程序图



比较：

- 基本路径测试：
  关注判定覆盖（如:右图一控制流图，DD路径图），不考虑变量的定义-使用关系

- 定义一使用测试：
  部分路径覆盖，要考虑程序中的变量定义一使用关系



佣金问题–变量的定义/使用节点

变量	定义节点	使用节点
lockPrice	7	24
stockPrice	8	25
barrelPrice	9	26
totalLocks	10, 16	16, 21, 24
totalStocks	11, 17	17, 22, 25
totalBarrels	12, 18	18, 23, 26
locks	13, 19	14, 16
stocks	15	17
barrels	15	18
lockSales	24	27
stockSales	25	27
barrelSales	26	27
sales	27	28, 29, 33, 34, 37, 38
commission	31, 32, 33, 36, 37, 38	32, 33, 37, 41

佣金问题—部分变量的定义清晰路径

变量	路径 （始结点，终结点）	是定义清晰路径吗？
lockPrice	(7, 24)	是
…	…	…
totalStocks	(11, 22)	否
…	…	…
commission	(31, 37)	-（不可行）
commission	(31, 32)	是
commission	(31, 33)	否
…	…	…

注意：“累加赋值语句”对应的结点， 既是定义结点，有是使用结点
如： commission=commission+0.15*800.0

佣金问题—部分变量的定义清晰路径

- “累加赋值语句”对应的结点， 既是定义结点，有是使用结点
  - 如： commission=commission+0.15*800.0
- 覆盖路径时，不考虑包含“累加赋值语句”的路径， 如 (11, 22) , (31, 33), …

定义—使用路径测试 的路径覆盖准则

设T是拥有变量集合V的程序P的程序图G（P）中的一个路径集合，假设其中的每个路径都是可行的。给出如下覆盖准则

- 集合T满足程序p的全定义—使用路径准则，当且仅当所有变量v∈V，T包含从v的每个定义节点到v的每个使用结点（及其后续节点）的定义清晰路径，并且这些路径要么有一次的环经过，要么没有环路（即：循环体所对应的定义清晰路径，最多只覆盖一次）。

- 集合T满足程序P的全定义准则，当且仅当所有变量v∈V，T包含从v的每个定义结点到v的一个使用的定义清晰路径。

- 集合T满足程序p的全使用准则，当且仅当所有变量v∈V，T 包含从v的定义结点到v的每个使用结点（及其后续节点）的定义清晰路径。

定义—使用路径测试 的覆盖准则

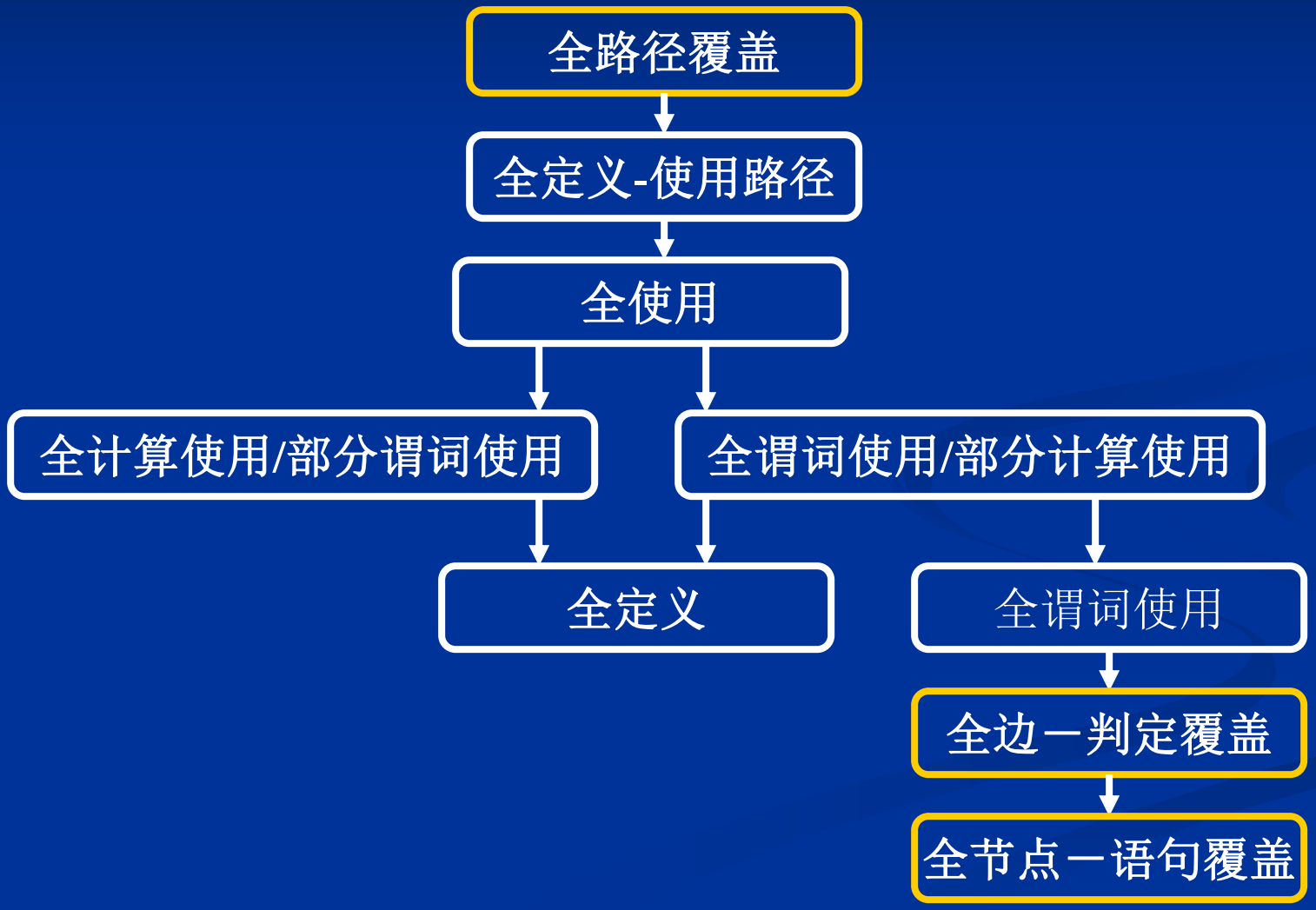
设T是拥有变量集合V的程序P的程序图G（P）中的一个路径集合，假设其中的每个路径都是可行的。给出如下覆盖准则

- 集合T满足程序p的全谓词使用 / 部分计算使用准则，当且仅当所有变量v ∈V，T包含从v的每个定义节点到v的所有谓词使用结点的定义清晰路径，并且如果v的一个定义没有谓词使用，则定义清晰路径导致至少一个计算使用结点。

- 集合T满足程序p的全计算使用 / 部分谓词使用准则，当且仅当所有变量v∈v，T包含从v的每个定义节点到v的所有计算使用结点的定义清晰路径，并且如果v的一个定义没有计算使用，则定义清晰路径导致至少一个谓词使用结点。

数据流覆盖准则—Rapps / Weyuker层次结构

（1985年）



数据流测试

- 使用/定义路径测试
- 基于程序片(Slice)的测试
  - 关注：
    - 影响某个变量在某个点上的取值的一组程序语句，称为“程序片Slice”
    - 将注意力集中到感兴趣的语句片，而不考虑无关的语句
  - 给定一个程序P和一个给出语句编号的程序图G(P)，以及P中的一个变量集合V，变量集合V在语句n上的一个程序片Slice，记做S(V，n)，是P中在语句n（含语句n本身）以前对V中的变量值作出贡献的所有语句编号的集合。
    - 作出贡献：直接或间接地影响变量的值
    - 输入定义语句，赋值定义语句，某些谓词使用语句

```
13  Input(locks)
14  While NOT( locks = = -1 )   'Input –1 to indicate end of data
15              Input(stocks，barrels)
16              totalLocks=totalLocks+locks
17              totalStocks=totaIStocks+stocks
18              totalBarrels=totalBarrels+barrels
19              Input(locks)
20  EndWhile
```

例： 变量locks上的片

S_1： S (locks，13) = { 13 }

S_2： S (locks，14) = { 13，14，19 }

S_3： S (locks，16) = { 13，14，16，19}

S_4： S (locks，19) = { 19 }

比较

- 基本路径测试
  - 适合测试 控制密集型的程序
- 数据流测试
  - 适合测试
    - 计算密集型的程序
    - 控制密集型的程序：（全谓词使用/部分计算使用覆盖准则）
  - 比基本路径测试更精细
  - 分类
    - 使用/定义路径测试
    - 基于程序片的测试
      - 将注意力集中到感兴趣的语句片，而不考虑无关的语句
        - 比定义/使用路径测试更精细
      - 程序片不能很好地映射到测试用例，因为无关语句仍然在执行路径中

白盒测试的 总结与分析

- 什么时候可以停止测试？
  - 软件交付时间到了
  - 软件的所有缺陷都被清除了
  - 当继续测试，却不能发现新的缺陷，或者只能发现很少的不重要的缺陷时
  - 所能用的测试方法和测试用例都用过了
  - 当达到了所要求的覆盖指标时

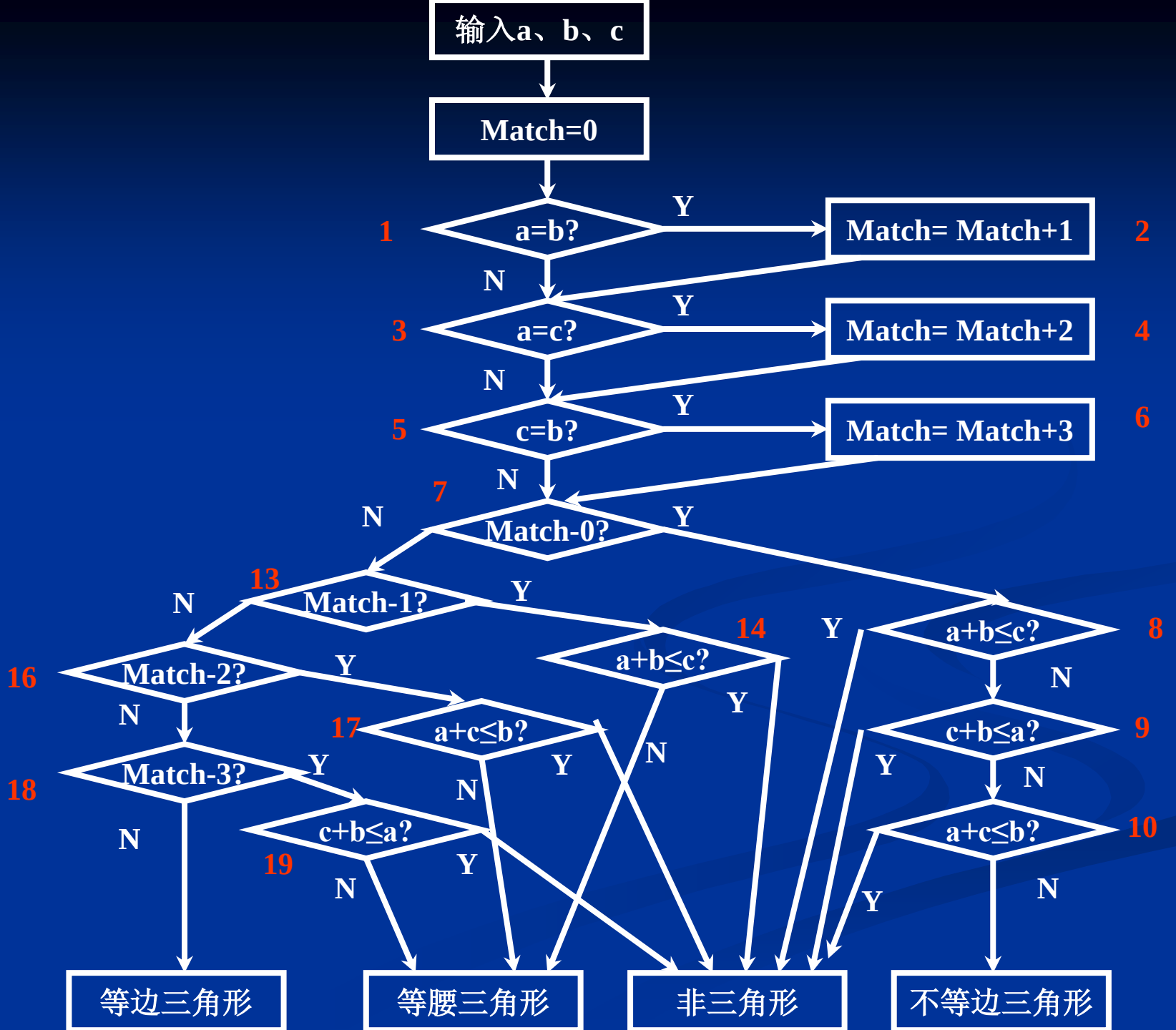
- 什么时候可以停止测试？
  - 软件交付时间到了 （最常见）
  - 软件的所有缺陷都被清除了 （不可能）
  - 当继续测试，却不能发现新的缺陷，或者只能发现很少的不重要的缺陷时 （？）
  - 所能用的测试方法和测试用例都用过了 （？）
  - 当达到了所要求的覆盖指标时

- 白盒测试的特点
  - 基于被测程序的源代码生成测试用例
  - 支持严格定义、数据分析和覆盖度量。
    - 覆盖指标(可量化)：
      - 基本路径数，定义/使用路径数，程序片数，…
    - 用于：
      - 判断比较测试的效果
      - 判断测试是否可以结束

案例分析

- 利用基本路径覆盖 评估黑盒测试的冗余和遗漏问题
- 例：评估“三角形问题—边界值分析测试”

三角形问题的传统实现


输入a、b、c
Match=0
1
a=b?
Y
Match= Match+1
2
N
3
a=c?
Y
Match= Match+2
4
N
5
c=b?
Y
Match= Match+3
6
N
7
Match-0?
N
Y
13
Match-1?
N
Y
14
a+b≤c?
Y
8
a+b≤c?
16
Match-2?
Y
N
Y
N
17
a+c≤b?
Y
N
9
c+b≤a?
Y
N
18
Match-3?
Y
N
19
c+b≤a?
Y
N
10
a+c≤b?
Y
N
等边三角形
等腰三角形
非三角形
不等边三角形

基本路径—11条

路径	节点序列	描述
P1	1-2-3-4-5-6-7-13-16-18-20	等边三角形
P2	1-3-5-6-7-13-16-18-19-15	等腰三角形（b=c）
P3	1-3-5-6-7-13-16-18-19-12	非三角形（b=c）
P4	1-3-4-5-7-13-16-17-15	等腰三角形（a=c）
P5	1-3-4-5-7-13-16-17-12	非三角形（a=c）
P6	1-2-3-5-7-13-14-15	等腰三角形（a=b）
P7	1-2-3-5-7-13-14-12	非三角形（a=b）
P8	1-3-5-7-8-12	非三角形（a+b≤c）
P9	1-3-5-7-8-9-12	非三角形（c+b≤a ）
P10	1-3-5-7-8-9-10-12	非三角形（a+c≤b ）
P11	1-3-5-7-8-9-10-11	不等边三角形

边界值分析生成的测试用例

所覆盖的基本路径

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
6	100	1	100	等腰三角形	P4
7	100	2	100	等腰三角形	P4
8	101	101	101	等边三角形	P1
9	100	199	100	等腰三角形	P4
10	100	200	100	非三角形	P5
11	1	100	100	等腰三角形	P2
12	2	100	100	等腰三角形	P2
13	200	200	200	等边三角形	P1
14	199	100	100	等腰三角形	P2
15	200	100	100	非三角形	P3

边界值分析生成的测试用例所覆盖的基本路径

基本路径	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
边界值测试用例	3	3	1	3	1	3	1	0	0	0	0

边界值分析测试：

漏洞：覆盖了7条基本路径，还有4条未覆盖

冗余：共15个测试用例，如：3个测试用例覆盖的是同一条P1路径

比较：黑盒测试生成的测试用例所覆盖的基本路径

基本路径	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
边界值测试用例	3	3	1	3	1	3	1	0	0	0	0
最坏情况测试用例	5	12	6	11	6	12	7	17	18	19	12

边界值分析测试：

漏洞：覆盖了7条基本路径，还有4条未覆盖

冗余：共15个测试用例，如：3个测试用例覆盖的是同一条P1路径

最坏情况边界值测试：

无漏洞：覆盖了全部的11条基本路径

冗余严重：共125个测试用例

黑盒测试的评估指标

假设黑盒测试技术M生成m个测试用例，并且根据标识被测单元中的s个元素的白盒测试指标S来跟踪这些测试用例。当执行m个测试用例时，会经过n个白盒测试元素。

M：黑盒测试技术
S：白盒测试技术 (典型的:基本路径测试)
m：黑盒测试所生成的测试用例个数
s：白盒测试所生成的路径总数
n：黑盒测试所覆盖的路径数

定义

方法M关于指标S的覆盖 C(M，S) = 所覆盖的路径数 n /总路径数 s
方法M关于指标S的冗余 R(M，S) = 测试用例个数 m / 总路径数 s
方法M关于指标S的净冗余 NR(M，S)= 测试用例个数 m / 所覆盖的路径数n

M：黑盒测试技术
S：白盒测试技术 (典型的:基本路径测试)
m：黑盒测试所生成的测试用例个数
s：白盒测试所生成的路径总数
n：黑盒测试所覆盖的路径数

三角形问题的边界值分析和最坏情况测试的覆盖指标

方法	m	n	s	覆盖 C(M,S)=n/s	冗余 R(M,S)=m/s	净冗余 NR(M,S)=m/n
边界值分析测试	15	7	11	0.64	1.36	2.14
最坏情况测试	125	11	11	1	11.36	11.36
目标	s	s	s	1	1	1

- 要减少遗漏和冗余，必须采用
  - “黑白盒技术相结合”的测试方法：

    先用黑盒测试技术生成测试用例，然后用白盒测试找到遗漏的路径从而补充相应的测试用例

白盒测试

- 逻辑覆盖与路径测试
- 基本路径测试（参考 书第9章）
- 数据流测试（参考 书第10章）
- 其它白盒测试技术

其它白盒测试技术

- 程序插装
- 域测试
- 符号测试
- 程序变异—错误驱动测试法
- …

- 静态检查

程序分析与编译技术
在系统与网络安全方面起到了重要作用。

程序污点分析实例研究
A Case Study on Taint Analysis

什么是污点？

- Taint（污点）：所有来自不可靠数据源的数据
  - 用户输入
  - 网络
  - etc.
- 例如
  - 所有常量（字符串）都是 untainted data
  - 所有变量都有可能是 tainted data
  - 来自网络，写入硬盘，之后再重新读入的数据依然是 tainted data

Taint Analysis 的用途

- 查找源码/系统漏洞，提高系统安全性
  - C/Java
  - Web应用（比如：Javascript）
  - 二进制程序（无源码）
- 可分为
  - Static Taint Analysis
  - Dynamic Taint Analysis

具体做什么？

- 利用已有的编译框架（e.g., Soot）实现静态的Taint Analysis算法
- 实例研究
  - 针对具体的程序示例查找其中潜在的漏洞
- 期望结果
  - 一篇研究报告（技术报告或者投稿论文）