Master course
Sept. 28, 2016
Wuhan, China

软件测试

03 白盒测试与代码覆盖

玄跻峰
武汉大学 软件工程国家重点实验室

Email：jxuan@whu.edu.cn
URL： http://jifeng-xuan.com/

本次课程内容

白盒测试

- 基于图的测试原理
- 考核内容介绍
- 代码覆盖
- 逻辑覆盖

引言

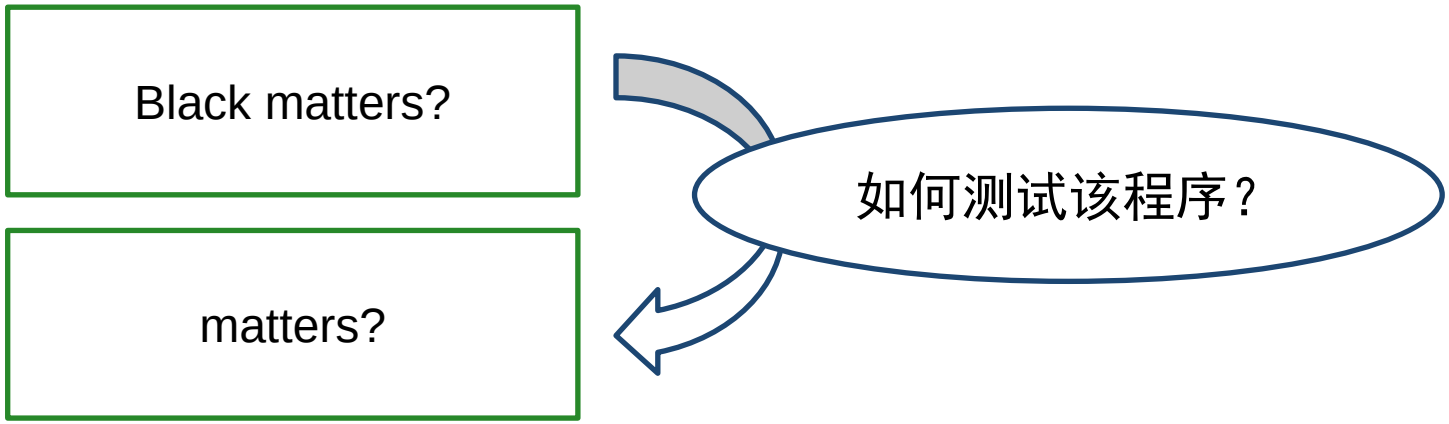
2016年9月23日，美国WSB电视台报道，三匪徒入室抢劫 。

乔治亚州一华裔女子，持枪击退三名入室盗窃者并射杀其中之一。

引言

2016年9月23日，美国WSB电视台报道，三匪徒入室抢劫 。
乔治亚州一华裔女子，持枪击退三名入室盗窃者并射杀其中之一。

视频被传到YouTube，网友评论。但是基于政治正确原因，涉及种族词语被隐去。



白盒测试

测试用例 = 输入数据 + 测试预言

= “Black matters?” + “ matters?”

测试充分性

定义：

针对需求R中的每个r，测试用例集合T中至少有一个测试用例t，能够表明被测程序P满足r，则认为T对于(P, R)是充分的。

测试充分性

定义：

针对需求R中的每个r，测试用例集合T中至少有一个测试用例t，能够表明被测程序P满足r，则认为T对于(P, R)是充分的。

P: 返回x和y中的最小正数，若没有，返回-1

```
int minPositive (int x, int y) {
1    int z = -1;
2    if (x > 0)
3        z = x;
4    if (y > 0)
5        if (z > 0)
6            if (y < z)
7                z = y;
8    return z;
}
```

语句覆盖：4个需求，1,3,7,8

条件覆盖：4个需求，2,4,5,6

路径覆盖：8个需求，

1,2,3,4,5,6,7,8

1,4,5,6,7,8

1,4,5,8

1,4,8

1,8

1,2,3,4,5,8

1,2,3,4,8

1,2,3,8

无向图

定义2.1 无向图：

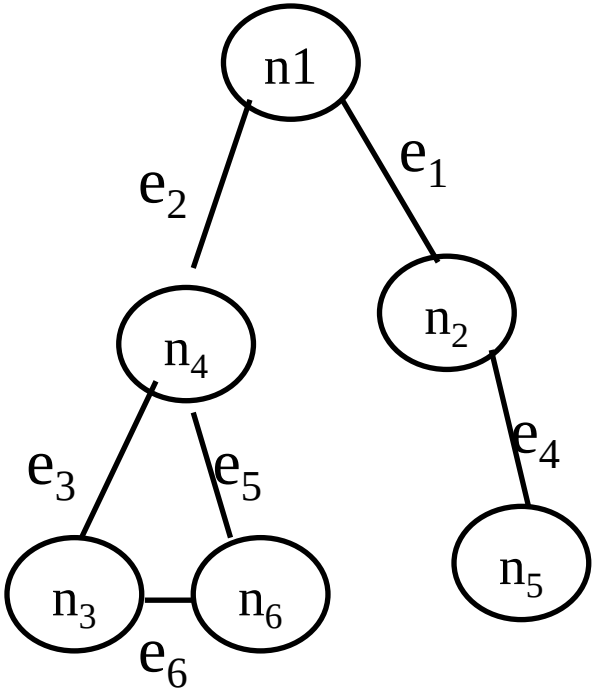
图G=(V，E)由节点的有限(并且非空)集合V和节点无序对偶集合E组成。

$V=\{n_1，n_2，\ldots，n_m\}$

$E=\{e_1，e_2，\ldots，e_p\}$

其中每条边$e_k=\{n_i，n_j\}$，n_i、$n_j \in V$。集合$\{n_i，n_j\}$是一个无序对偶，记做$(n_i，n_j)$。

图的例子



$V=\{ n_1, n_2, n_3, n_4, n_5, n_6, n_7\}$

$E=\{ e_1, e_2, e_3, e_4, e_5, e_6\}$

$n_7 =\{(n_1, n_2), (n_1, n_4), (n_3, n_4),$

$(n_2, n_5), (n_4, n_6), (n_3, n_6)\}$

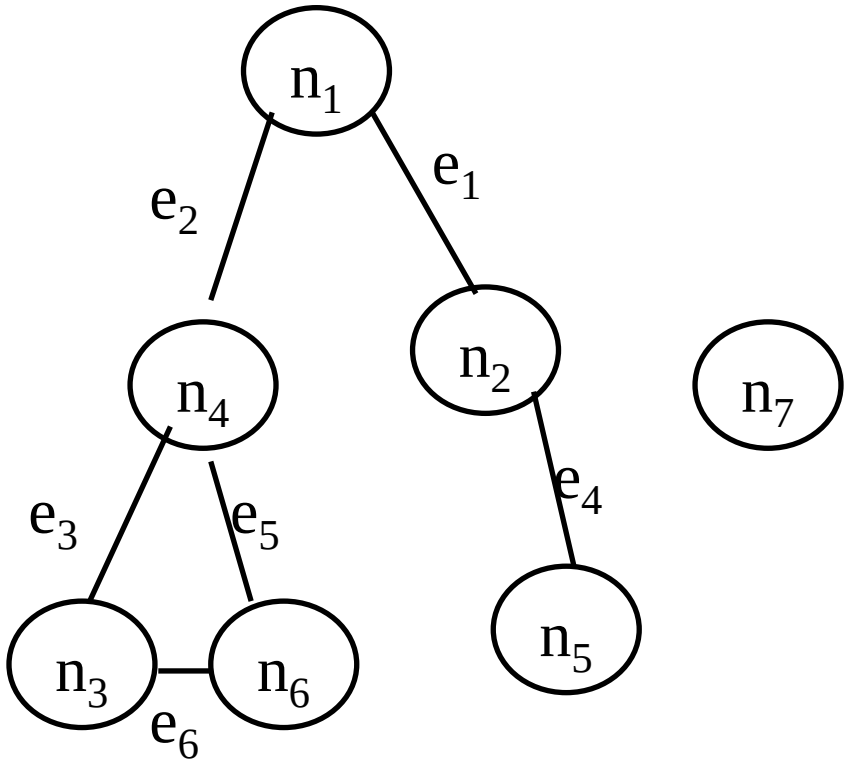
节点的度

定义2.2节点的度：

节点的度是以该节点作为端点的边的条数

节点n的度记做deg(n)

图的例子



$deg(n_1)=2$

$deg(n_2)=2$

$deg(n_3)=2$

$deg(n_4)=3$

$deg(n_5)=1$

$deg(n_6)=2$

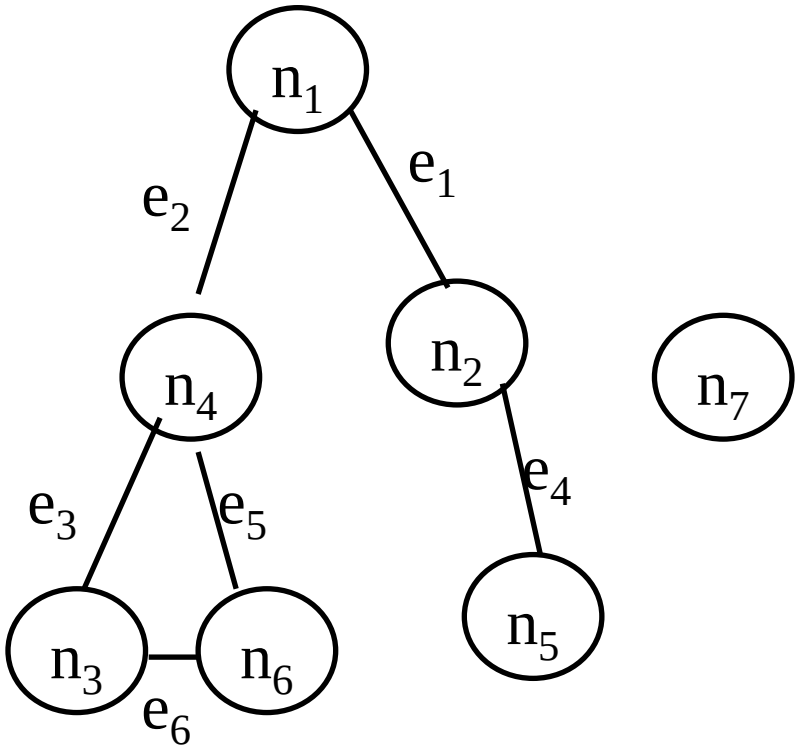
$deg(n_7)=0$

关联矩阵

定义2.3 关联矩阵：

拥有m个节点和n条边的图G=(V，E)的关联矩阵是一种m×n矩阵，其中第i行第j列的元素是1，当且仅当节点i是边j的一个端点，否则该元素是0。

图的例子



	e1	e2	e3	e4	e5	e6
n1	1	1	0	0	0	0
n2	1	0	0	1	0	0
n3	0	0	1	0	0	1
n4	0	1	1	0	1	0
n5	0	0	0	1	0	0
n6	0	0	0	0	1	1
n7	0	0	0	0	0	0

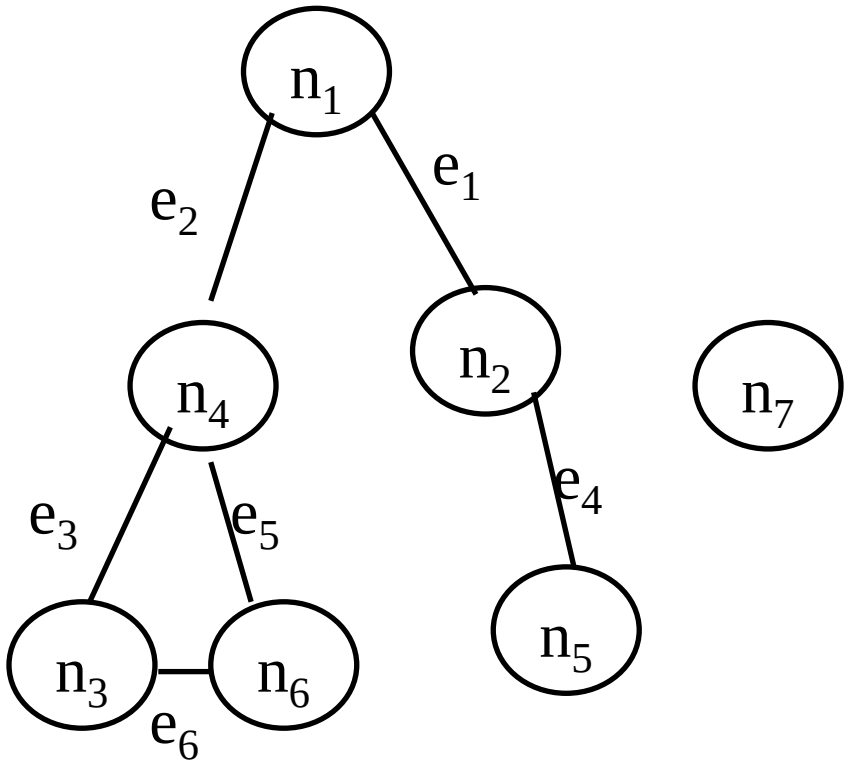
列的表项和为2

相邻矩阵

定义2.4 相邻矩阵

拥有m个节点和n条边的图G=(V，E)的相邻矩阵是一种m×m矩阵，其中第i行第j列的元素是1，当且仅当节点i和节点j之间存在一条边，否则该元素是0。

图的例子



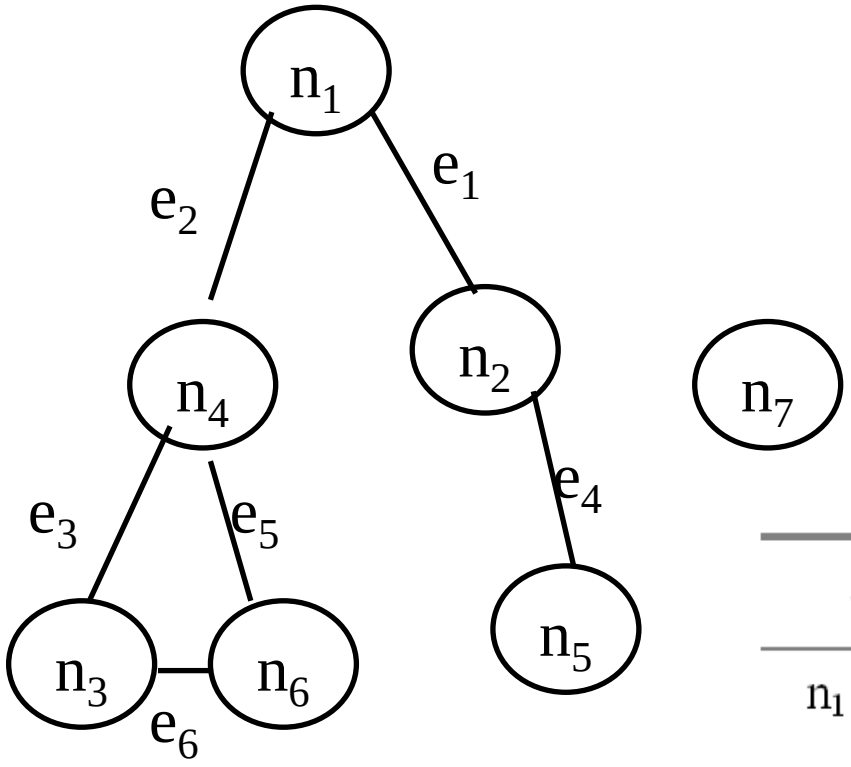
	n1	n2	n3	n4	n5	n6	n7
n1	0	1	0	1	0	0	0
n2	1	0	0	0	1	0	0
n3	0	0	0	1	0	1	0
n4	1	0	1	0	0	1	0
n5	0	1	0	0	0	0	0
n6	0	0	1	1	0	0	0
n7	0	0	0	0	0	0	0

路径

定义2.5 路径：

路径是一系列的边，对于序列中的任何相邻边对偶e_i、e_j，边都拥有相同的(节点)端点。

图的例子



路径可以使用二项形式的矩阵乘法和加法，直接通过图的相邻矩阵生成

路　径	节点序列	边序列
n_1 和 n_5 之间	n_1, n_2, n_5	e_1 ,e_4
n_6 和 n_5 之间	n_6, n_4, n_1, n_2, n_5	e_5 ,e_2, e_1, e_4
n_3 和 n_2 之间	n_3, n_4, n_1, n_2,	e_3 ,e_2, e_1

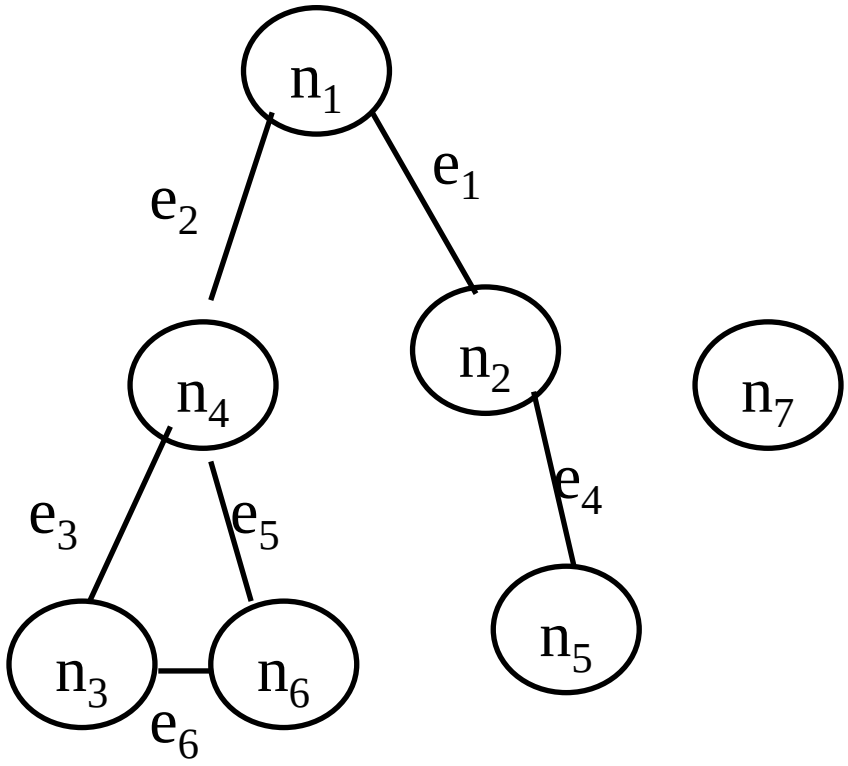
连接性

定义2.6 连接性：

节点n_i和n_j是被连接的，当且仅当它们都在同一条路径上

- 自反:因为每个节点显然都在到其本身长度为0的路径上
- 对称:由于如果n_i和n_j在一条路径上，则n_j和n_i也在同一条路径上
- 传递:由于如果n_i和n_k在一条路径上，n_k和n_j在一条路径上，则n_i和n_j也在同一条路径上

组件和压缩图



定义2.7组件：图的组件是相连节点的最大集合。

定义2.8压缩图：给定图G=(V,E)，其压缩图通过用压缩节点替代每个组件构成。

圈复杂度

定义2.9 圈复杂度 (cyclomatic complexity)：

图G的圈数由V(G)=e-n+2给出，其中：e是G中的边数。n是G中的节点数。

V(G)=6-7+2=1

圈复杂度所反映的是“判定条件”的数量，所以圈复杂度实际上就是等于判定节点的数量再加上1，也即控制流图的区域数，对应的计算公式为：V(G)=区域数=判定节点数+1。

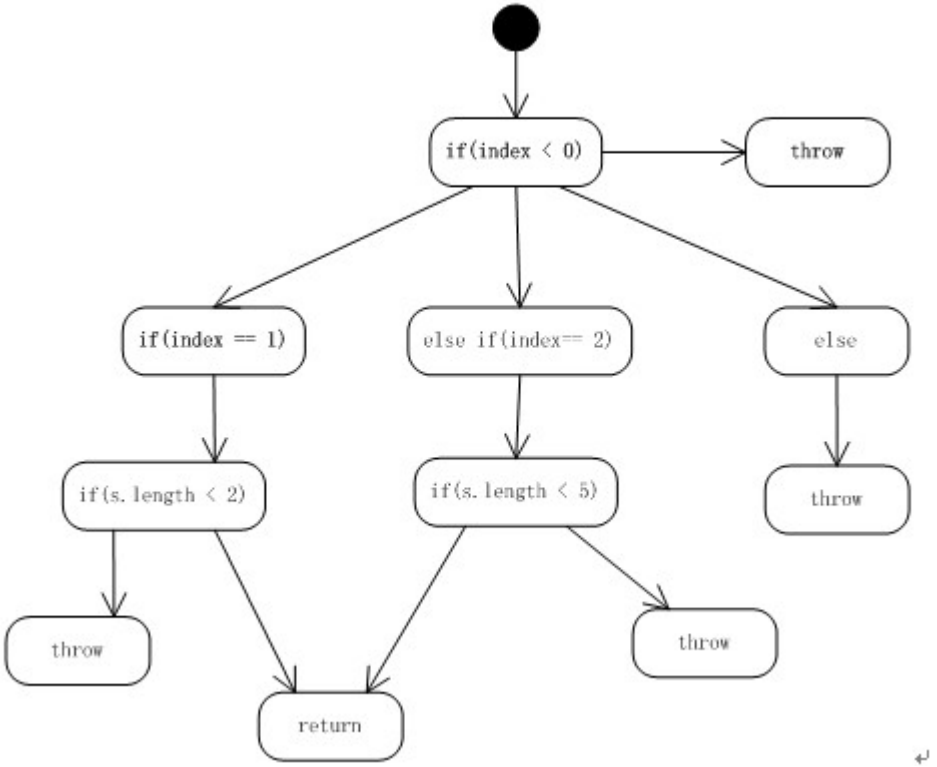
圈复杂度

圈复杂度主要与分支语句（if、else、Switch等）的个数成正相关。当一段代码中含有较多的分支语句，其逻辑复杂程度就会增加。

圈复杂度

```
public String case2(int index, String string) {
  String returnString = null;
  if (index < 0) {
      thrownew IndexOutOfBoundsException("exception <0 ");
  }
  if (index == 1) {
      if (string.length() < 2) {
         return string;
      }
      returnString = "returnString1";
  } elseif (index == 2) {
      if (string.length() < 5) {
         return string;
      }
      returnString = "returnString2";
  } else {
      thrownew IndexOutOfBoundsException("exception >2 ");
  }
  returnreturnString;
}
```

圈复杂度



根据公式 V(G) = e – n + 2 = 12 – 8 + 2 = 6 。圈复杂度为6。

考核内容简介

在线的测试用例填写

覆盖某一准则的测试用例

提交文档或课上展示

你在日常科研或项目中，遇到的困难的bug，
以及如何发现其根源并解决的。

- 如果已经解决，
  通过什么途径（例如测试方法）解决；
- 如果尚未解决，
  那么bug的潜在可能成因是什么，应该如何探索。

有向图

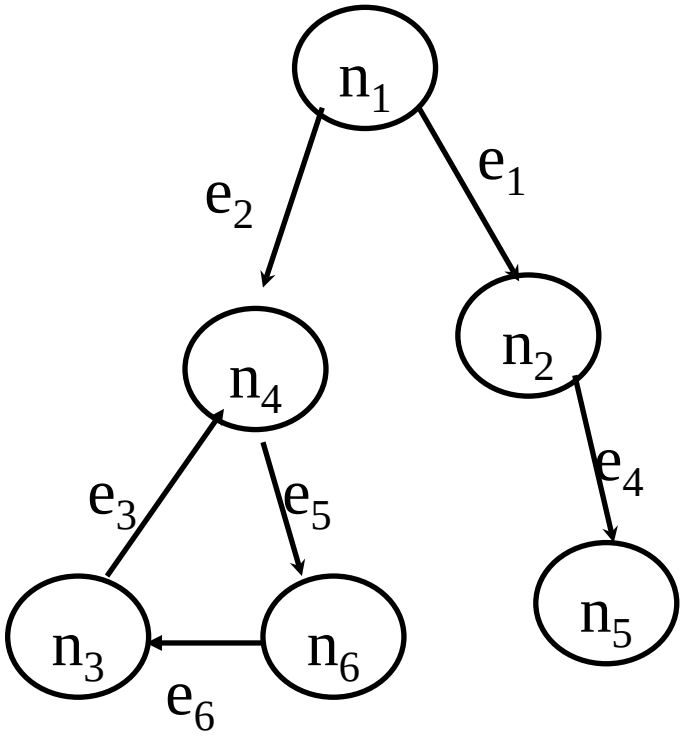
定义2.10 有向图

有向图D=(V，E)包含：一个节点的有限集合V={n_1，n_2，…，n_m}，一个边的集合 E={e_1，e_2，…，e_p}，其中每条边e_k=< n_i，n_j>，是节点n_i、n_j∈V的一个有序对偶。

一般图和有向图之间的差别与说明式和命令式程序设计语言之间的差别有很强的类比性

C和Prologue，实体/关系(ER)模型

图的例子



n_7

V={ n_1, n_2, n_3, n_4, n_5, n_6, n_7}

E={ e_1, e_2, e_3, e_4, e_5, e_6}

={<n_1, $n_{2>}$, <n_1, n_4>, <n_3, n_4>, <n_2, n_5>, <n_4, n_6>, <n_6, n_3> }

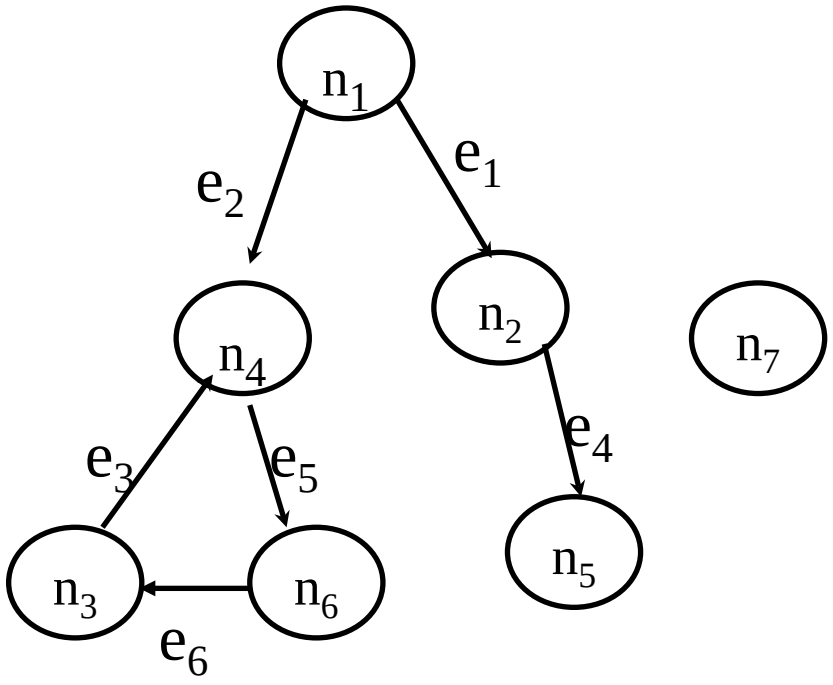
内度与外度

定义2.11 内度与外度：

有向图中节点的内度，是将该节点作为终止节点的不同边的条数。节点n的内度记做indeg（n）。

有向图中节点的外度，是将该节点作为开始节点的不同边的条数。节点n的外度记做outdeg(n)。

图的例子



$deg(n_i) = indeg(n_i) + outdeg(n_i)$

$indeg(n_1)=0$ $outdeg(n_1)=2$

$indeg(n_2)=1$ $outdeg(n_2)=1$

$indeg(n_3)=1$ $outdeg(n_3)=1$

$indeg(n_4)=2$ $outdeg(n_4)=1$

$indeg(n_5)=1$ $outdeg(n_5)=0$

$indeg(n_6)=1$ $outdeg(n_6)=1$

$indeg(n_7)=0$ $outdeg(n_7)=0$

有向图的相邻矩阵

定义2.12有向图的相邻矩阵：

拥有m个节点的有向图D=(V，E)的相邻矩阵是一种m×m矩阵，其中第i行第j列的元素是1，当且仅当节点i到节点j存在一条边，否则该元素是0。

	n_1	n_2	n_3	n_4	n_5	n_6	n_7
n_1	0	1	0	1	0	0	0
n_2	0	0	0	0	1	0	0
n_3	0	0	0	1	0	0	0
n_4	0	0	0	0	0	1	0
n_5	0	0	0	0	0	0	0
n_6	0	0	1	0	0	0	0
n_7	0	0	0	0	0	0	0

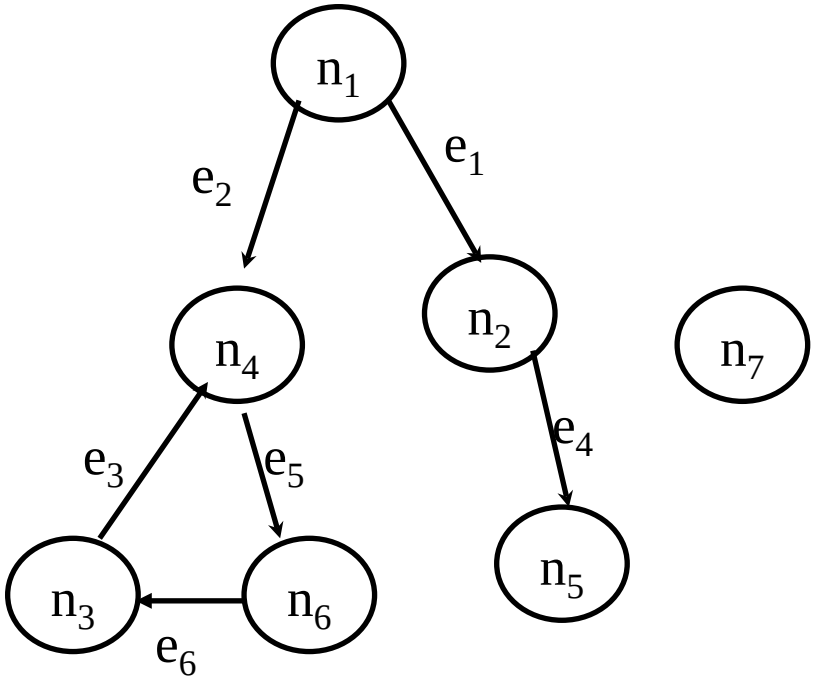
有向图的路径

定义2.13路径： (有向)路径是一系列边，使得对于该序列中的所有相邻边对偶e_i、e_j来说，第一条边的终止节点是第二条边的初始节点。

定义2.14 环路：环路是一个在同一个节点上开始和结束的有向路径。

定义2.15半路径：(有向)半路径是一系列边，使得对于该序列中至少有一个相邻边对偶e_i、e_j来说，第一条边的初始节点是第二条边的初始节点，或第一条边的终止节点是第二条边的终止节点。

图的例子



路　径	节点序列	边序列
n_1 和 n_6 之间	n_1, n_4, n_6	e_2 ,e_5
n_1 和 n_5 之间	n_1, n_2, n_5	e_1, e_4
n_3 和 n_6 之间	n_3, n_4, n_6	e_3 ,e_5

半 路 径	节点序列	边序列
n_1 和 n_3 之间	n_1, n_4, n_3	e_2 ,e_3
n_2 和 n_4 之间	n_2, n_1, n_4	e_1, e_2
n_5 和 n_6 之间	n_5, n_2, n_1, n_4, n_6	e_4 ,e_1,e_2,e_5

有向图连接性

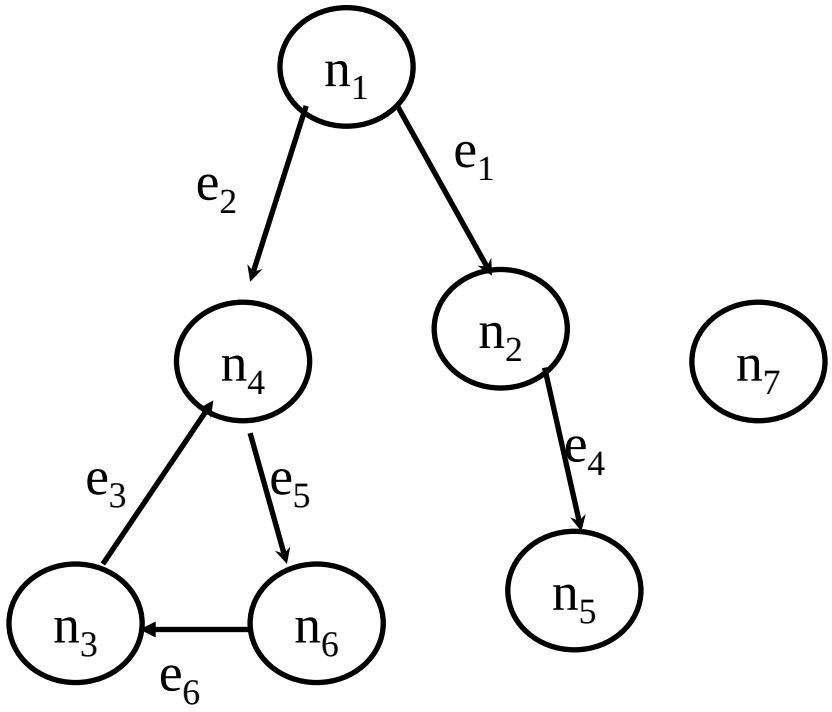
定义2.17 0-连接：对有向图中的两个节点n_i 和n_j，当且仅当n_i 和n_j之间没有路径，称n_i 和n_j是0-连接的。

定义2.18 1-连接：对有向图中的两个节点n_i 和n_j，当且仅当n_i 和n_j之间有一条半路径，但是没有路径，称n_i 和n_j是1-连接的。

定义2.19 2-连接：对有向图中的两个节点n_i 和n_j，当且仅当n_i 和n_j之间有一条路径，称n_i 和n_j是2-连接的。

定义2.20 3-连接：对有向图中的两个节点n_i 和n_j，当且仅当从n_i 到n_j有一条路径，并且从n_j到n_i有一条路径，称n_i 和n_j是3-连接的。

有向图连接性



n_1和n_7 是0-连接

n_2和n_6 是1-连接

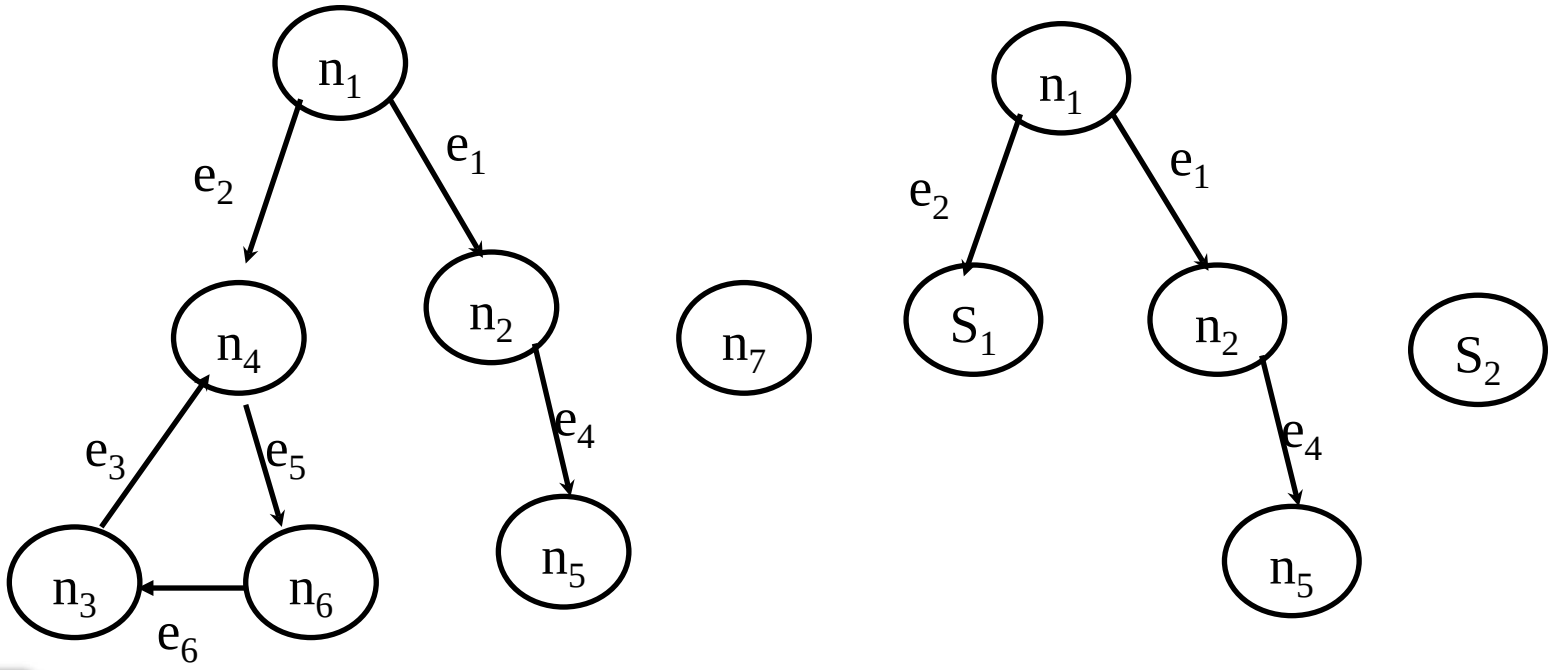
n_1和n_6 是2-连接

n_3和n_6 是3-连接。

强组件

定义2.21强组件：有向图的强组件是3-连接节点的最大集合。

强组件是集合{ n_3、n_4、n_6}和{n_7},



用于测试的图

程序图

有限状态机

Petri网

事件驱动的Petri网

状态图

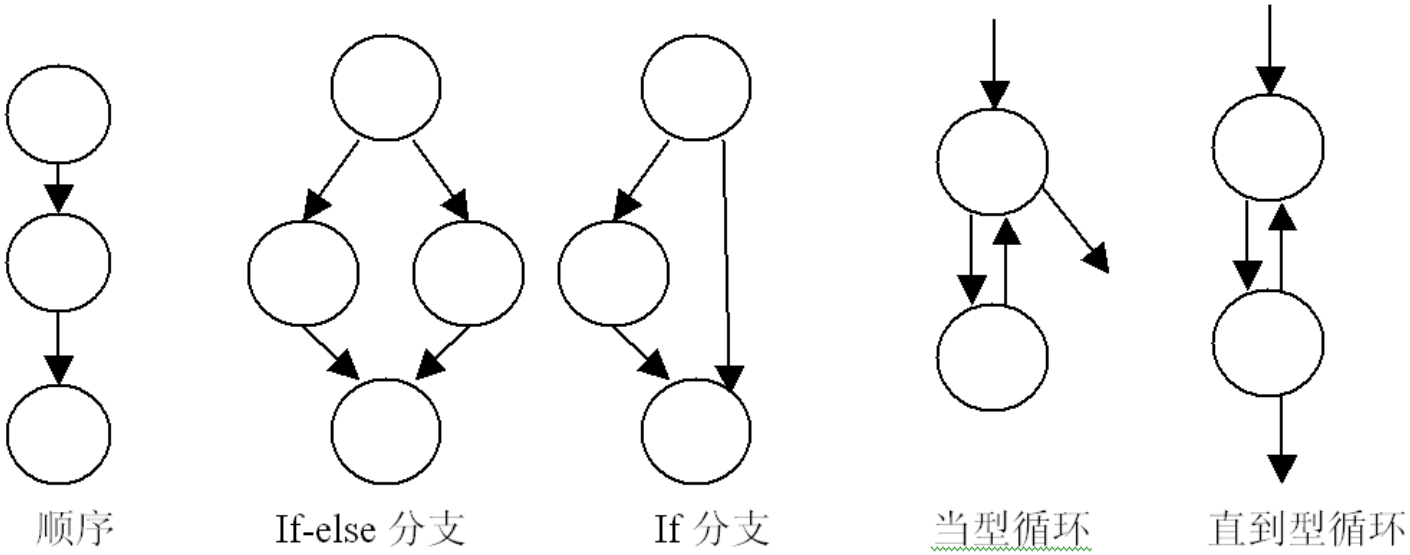
程序图

定义2.22 程序图：

给定一个采用命令式程序设计语言编写的程序，其程序图是一种有向图，其中节点是程序语句，边表示控制流(从节点i到节点j有一条边，当且仅当对应节点j的语句可以立即在节点i对应的语句之后执行)。

对该定义也可进行改进，即将节点修改为要么是整个语句，要么是语句的一部分。

程序图



唯一的源节点和唯一的汇节点
GOTO语句　　复杂

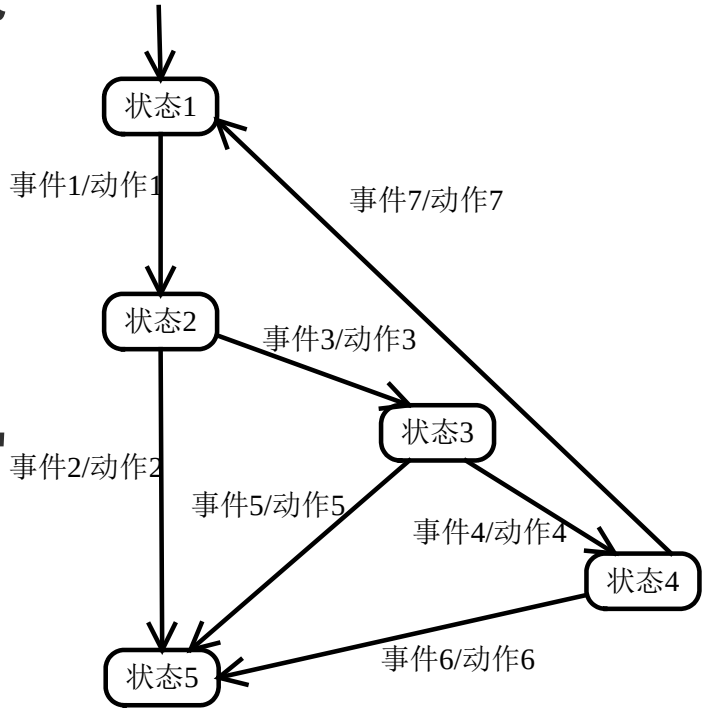
- 非可执行语句 （注释和数据说明语句）
- 拓扑结构可能的路径和在语义上可能不可行

有限状态机

有限状态机已经成为需求规格说明的一种相当标准的表示方法。

有限状态机是一种有向图，其中状态是节点，转移是边。

"分子"是引起转移的事件，"分母"是与该转移关联的行为



改变测试的概念

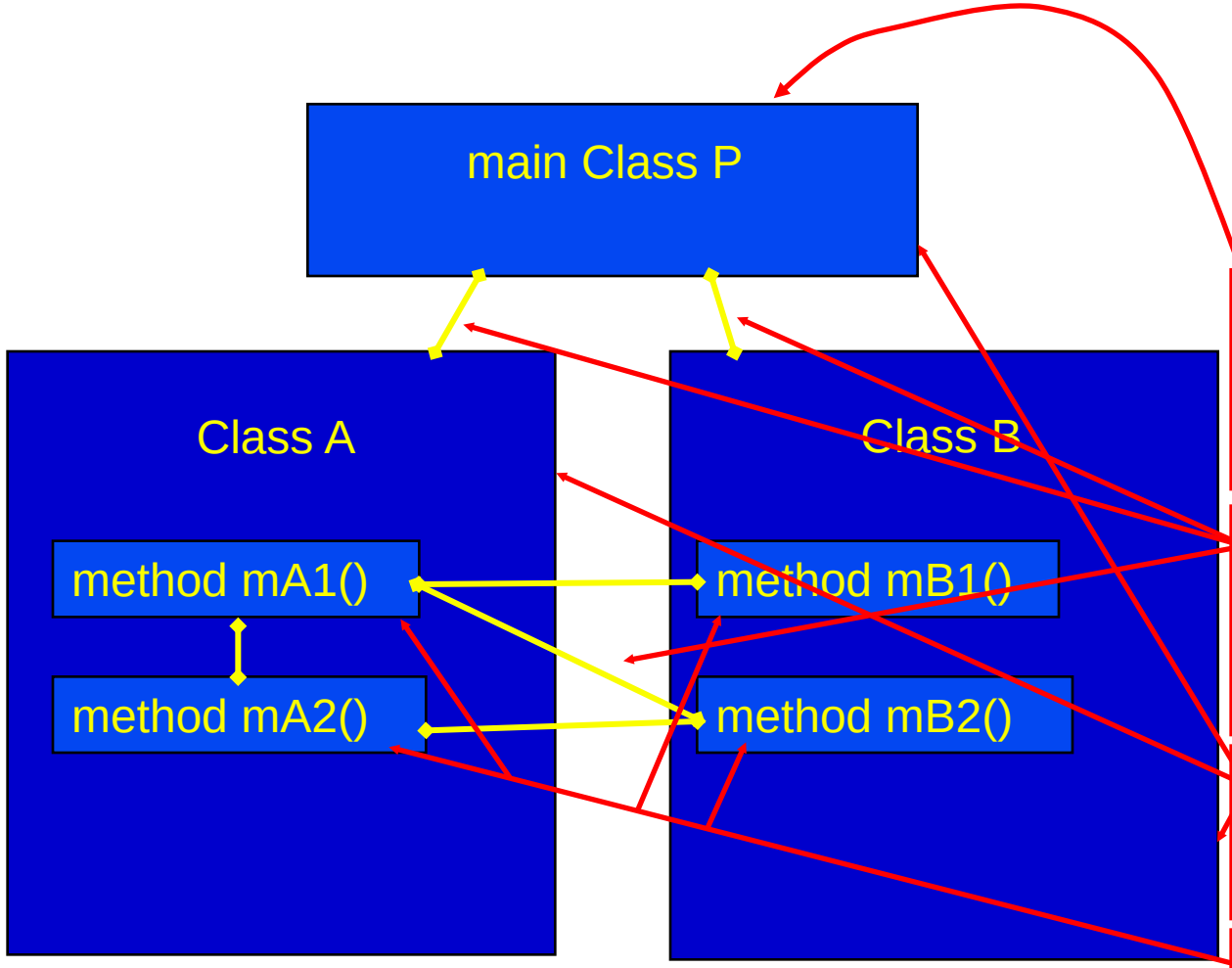
旧的观点：在特定软件开发阶段的测试

- 单元, 模块, 集成, 系统 …

新的观点： 根据结构和准则来测试

- 图,逻辑表达式,语法,输入空间

Old：在不同层次的测试



- 系统测试:
  测试所有的系统功能
- 集成测试:
  测试模块之间如何交互
- 模块测试:
  测试每个类，文件，模块，组件
- 单元测试:
  测试每个单元（方法）

New：测试覆盖准则

测试员的工作简化为：　定义软件模型，然后找到方法去覆盖它。

定义2.24 测试需求 测试需求是软件制品的一个特殊元素，测试用例必须要满足或覆盖它。

定义2.25 覆盖准则　覆盖准则是一个或一组规则，它将测试需求加在一个测试集上。

New：测试覆盖准则

定义 2.26 覆盖 对于覆盖准则C，给定一组测试需试集T满足C是指当且仅当对于每一个在TR中的测试需求tr，T中至少存在一个测试t，使t满足tr。

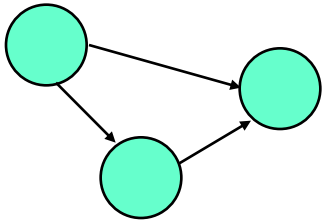
定义 2.27 覆盖率 给定一组测试需求TR和一个测试集T，覆盖率是被T满足的测试需求的数量与TR大小的比率。

定义 2.28 准则包含性 当且仅当每个测试集满足覆盖准则C1的同时也满足覆盖准则C2，则称C1包含C2。

New：基于结构的标准

结构：给软件建模的四种方式

1. 图



2. 逻辑表达式

(not X or not Y) and A and B

3. 输入域的特征

A: {0, 1, >1}
B: {600, 700, 800}
C: {swe, cs, isa, infs}

4. 语法结构

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

图覆盖准则

定义 2.29 测试中图G的形式化定义：

N代表结点的集合，它是一个非空集合。

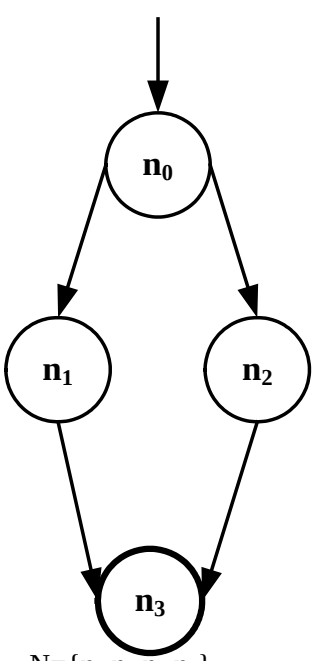
N_0代表始结点集合，其中N_0属于N，它是一个非空集合。

N_f代表终结点集合，其中N_f属于N，它是一个非空集合。

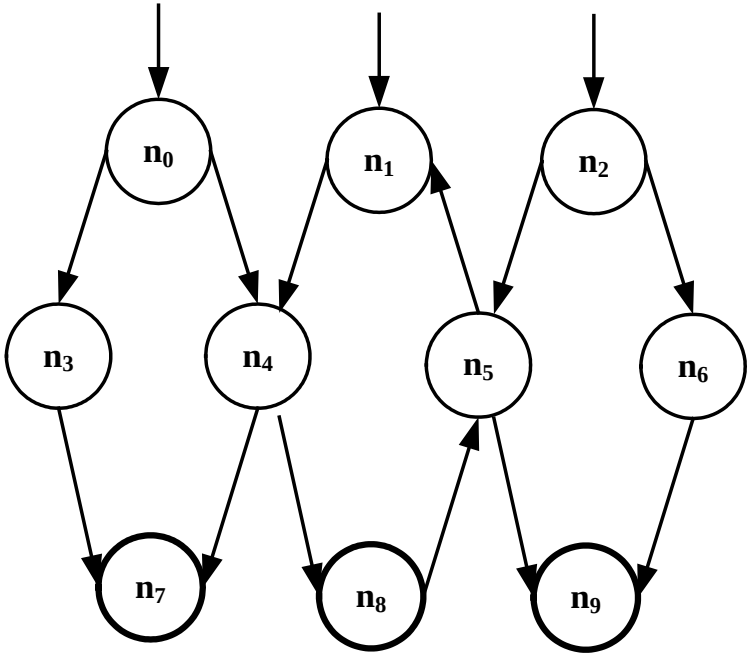
E代表边的集合，其中E是N×N的子集。

- N、N_0以及N_f来说每一个至少要包含一个结点
- 可以存在多余一个的始结点
- 必须至少存在一个终结点

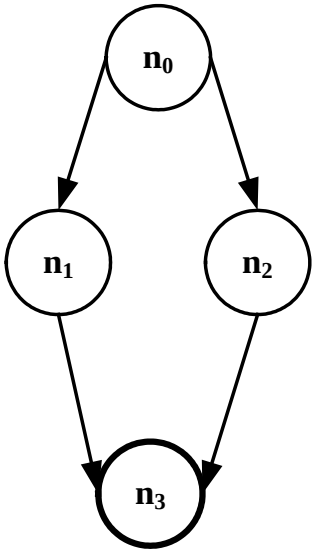
图



$N=\{n_0,n_1,n_2,n_3\}$
$N_0=\{n_0\}$
$E=\{<n_0,n_1>,<n_0,n_2>,<n_1,n_3>,<n_2,n_3>\}$
(a)有单个始结点的图



$N=\{n_0,n_1,n_2,n_3,n_4,n_5,n_6,n_7,n_8,n_9\}$
$N_0=\{n_0,n_1,n_2\}$
$|E|=12$
(b)有多个始结点的图



$N=\{n_0,n_1,n_2,n_3\}$
$|E|=4$
(c)没有始结点的图

可达性

$reach_G$（x）：从参数x开始的语法可达的

路径的实例	
1	n_0，n_3，n_7
2	n_1，n_4，n_8，n_5，n_1
3	n_2，n_6，n_9

无效路径的实例	
1	n_0，n_7
2	n_3，n_4
3	n_2，n_6，n_8

（a） 路径实例

可达性实例	
1	reach(n_0)=N-{ n_2，n_6}
2	reach(n_0，n_1，n_2)=N
3	reach(n_4)= { n_1，n_4，n_5，n_7，n_8，n_9}
4	reach([n_6，n_9])= { n_9 }

（b）可达性实例

测试路径

定义 2.30 测试路径

一个路径p，可能长度是0，从N_0中某结点开始到N_f中某结点结束。

访问和遍历

如果n在测试路径p中，则称测试路径p访问结点n。

如果边e在测试路径p中，则称测试路径p访问边e

如果q是测试路径p的子路径，则称测试路径p遍历子路径q。

路径[n_0,n_1,n_3,n_4,n_6],

访问结点n_0和n_1，

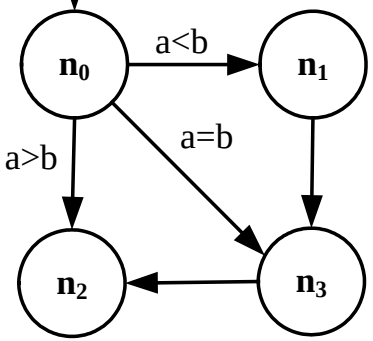
访问边（n_0，n_1）和（n_3，n_4），

遍历子路径[n_1，n_3，n_4]

映射关系

$path_G$（t）表示图G中测试用例t所执行的测试路径的映射关系

path（T）是T中测试所执行的测试路径的集合：$path_G$（T）={$path_G$（t）|t∈T}



(a)针对测试输入为a，b，输出为a+b的测试用例

映射到

测试用例t_1: (a=0,b=1) → [测试路径 p_1:n_0,n_1,n_3,n_2]

测试用例t_2: (a=1,b=1) → [测试路径 p_2:n_0,n_3,n_2]

测试用例t_3: (a=2,b=1) → [测试路径 p_3:n_0,n_2]

(b)测试用例和测试路径之间的映射

图覆盖

定义2.31 图覆盖

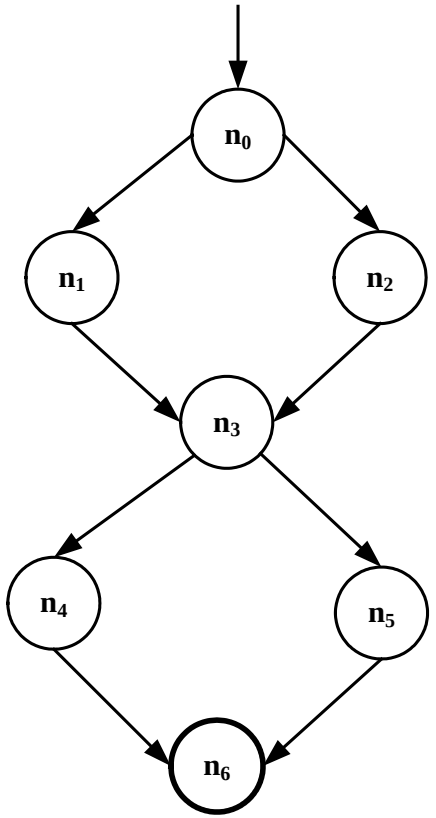
给定一个针对图准则C的测试需求的集合TR。测试集合T在图G上满足准则C当且仅当对TR中的每一个测试需求tr，path(T)中至少存在一条测试路径p满足tr。

结构化的覆盖准则

1. 结点覆盖准则

定义2.32 结点覆盖（形式化定义）：
对于每个结点 $n \in reach_G(N_0)$，TR包含谓词“访问n”。

TR={visit n_0，visit n_1，visit n_2，visit n_3，visit n_4，visit n_5，visit n_6}



结构化的覆盖准则

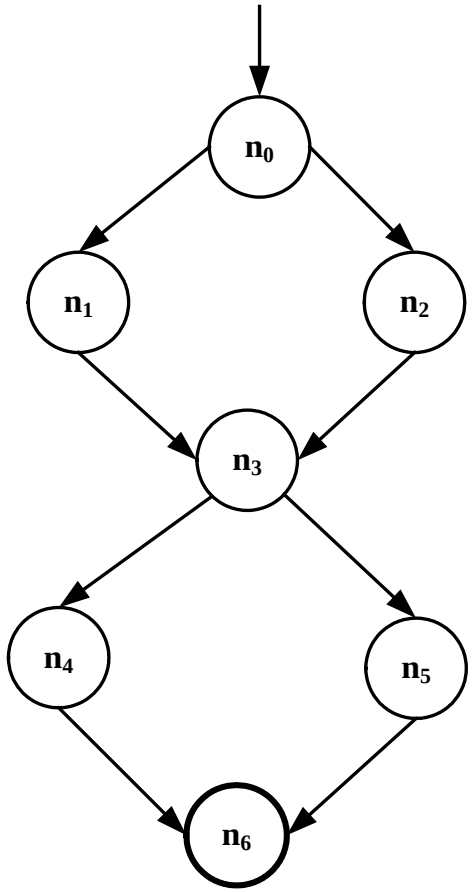
1. 结点覆盖准则

准则2.1 结点覆盖（NC）： TR包含图G中每个可达的结点。

TR={n_0,n_1,n_2,n_3,n_4,n_5,n_6}

p1=[n_0,n_1,n_3,n_4,n_6]满足第一个，第二个，第四个，第五个和第七个测试需求

p2=[n_0,n_2,n_3,n_5, n_6]满足第一个，第三个，第四个，第六个和第七个测试需求。

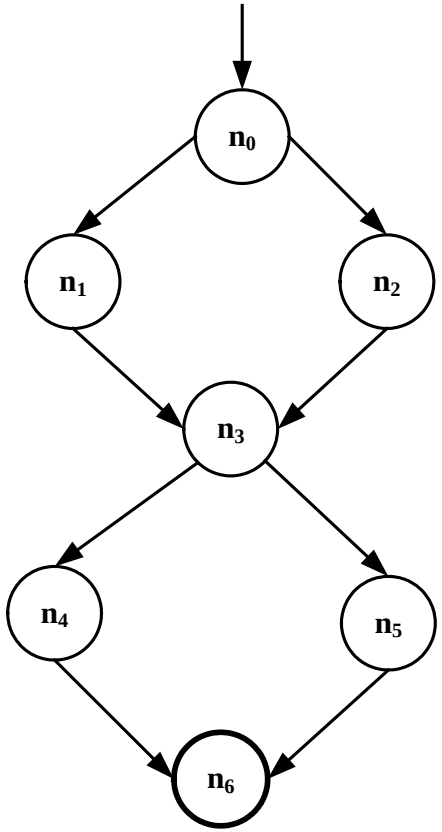


结构化的覆盖准则

1. 结点覆盖准则

如果测试集合T包括{t_1,t_2}，而path(t_1)=p_1和path(t_2)=p_2，那么T满足图G上的结点覆盖

定义2.33 结点覆盖（NC）（标准定义）：测试集合T满足图G上的结点覆盖当且仅当对N中每个语法可达的结点n，在path（T）中存在某条路径p，其中p访问n。



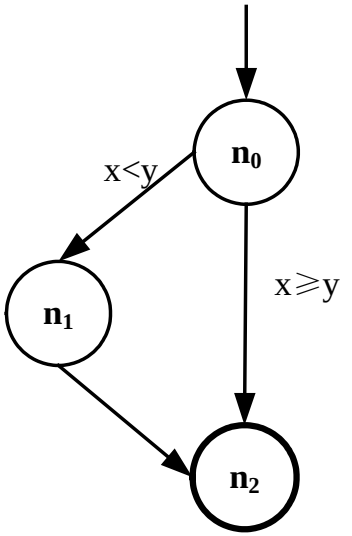
结构化的覆盖准则

2. 边覆盖准则

准则2.2 边覆盖（EC）：TR包含图G中长度不大于1的可达路径。

边覆盖的测试需求也明显的包括结点覆盖的测试需求。

准则2.3 双边（Edge-Pair）覆盖：TR包含图G中长度不大于2的可达路径。



$path(t_1)=[n_0,n_1,n_2]$
$path(t_2)=[n_0,n_2]$

$T_1=\{t_1\}$
T_1满足图的结点覆盖
（a）结点覆盖

$T_2=\{t_1,t_2\}$
T_2满足图的边覆盖
（b）边覆盖

结构化的覆盖准则

3. 回路覆盖准则

定义2.34 简单路径：如果在从n_i到n_j的一条路径中，除了始结点和终结点可以是同样的外，没有任何结点出现多于一次，则称该路径是简单的。

相当小的程序都可能有大量的简单路径

定义2.35 主路径：如果从n_i到n_j的路径是一条简单路径并且它没有作为任何其他简单路径的合适的子路径出现，则称它为主路径。

准则2.4 主路径覆盖（PPC）：TR包含图G中每一条主路径。

结构化的覆盖准则

回路路径即一个长度非零的起点和终点相同的主路径

准则 2.5 简单回路覆盖（SRTC）：针对图G中每个可达的结点，TR包含至少一条回路路径，此路径以该结点为起点和终点。

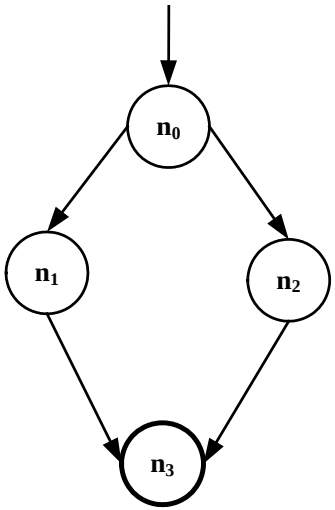
准则 2.6 完全回路覆盖（CTRC）：针对图G中每个可达的结点，TR包含所有回路路径。

结构化的覆盖准则

4. 路径覆盖准则

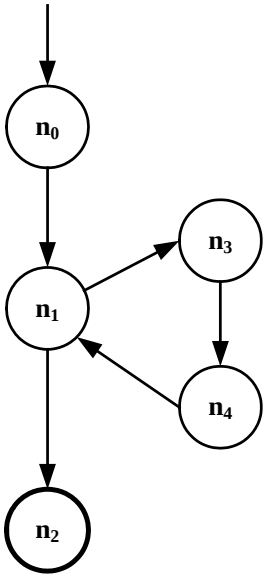
准则 2.7 完全路径覆盖（CPC）：TR包含图G中的所有路径。

准则 2.8 指定路径覆盖（SPC）：TR 包含一个测试路径集S，S被作为一个参数提供。



主路径={[n_0,n_1,n_3],[n_0,n_2,n_3]}
path(t_1)=[n_0,n_1,n_3]
path(t_2)=[n_0,n_2,n_3]
T_1={t_1,t_2}
T_1满足图的主路径覆盖

(a)无循环的图的主路径覆盖



主路径={[n_0,n_1,n_2],[n_0,n_1,n_3,n_4],
[n_1,n_3,n_4,n_1],[n_3,n_4,n_1,n_3],
[n_4,n_1,n_3,n_4],[n_3,n_4,n_1,n_2]}
path(t_3)=[n_0,n_1,n_2]
path(t_4)=[n_0,n_1,n_3,n_4,n_1,n_3,n_4,n_1,n_2]
T_2={t_3,t_4}
T_2满足图的主路径覆盖

(b)有循环的图的主路径覆盖

结构化的覆盖准则

5. 遍历，边路遍历和绕路遍历

处理不可行的测试需求

子路径中每个结点和每条边必须按它们在子路径中出现的顺序严格地被访问

定义 2.36 遍历：测试路径p遍历子路径q，当且仅当q是p的一个子路径。

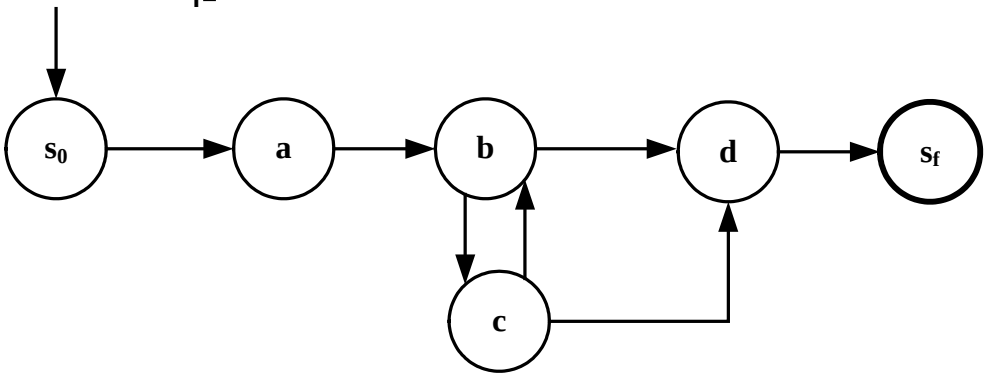
定义 2.37 边路遍历：测试路径p边路遍历子路径q，当且仅当q中的每条边以同样的顺序出现在p中。

定义 2.38 绕路遍历：测试路径p绕路遍历子路径q，当且仅当q中的每个结点以同样的顺序出现在p中。

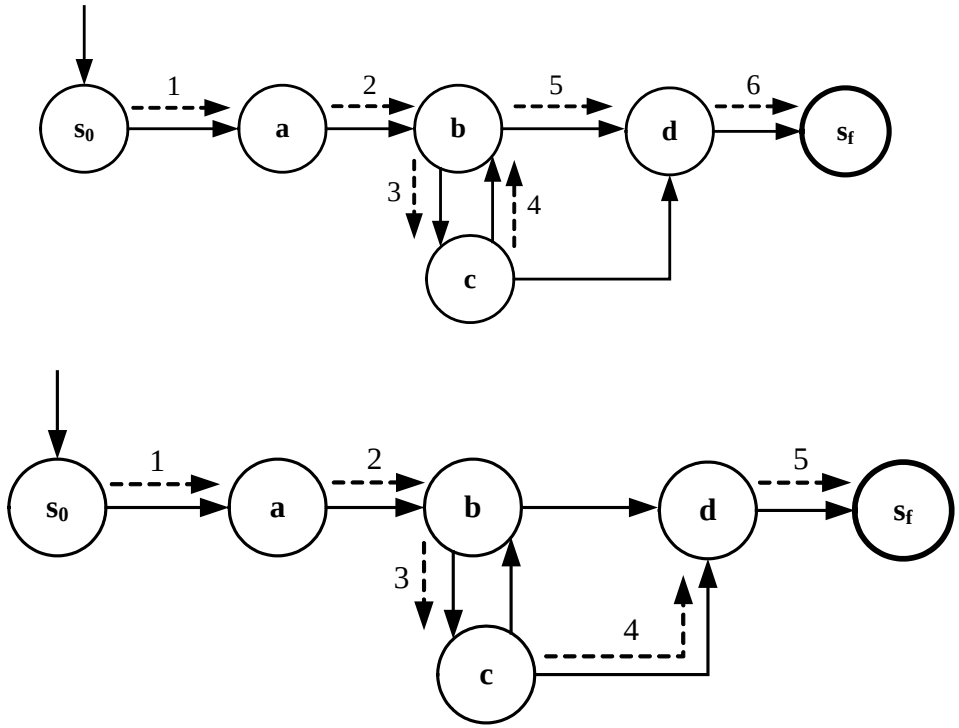
结构化的覆盖准则

q=[a,b,d]

p=[s_0,a,b,c,b,d,s_f]



结构化的覆盖准则



[b, c, b]是[a, b, d]的一个边路

[b, c, d]是[a, b, d]的一个绕路

绕路有能力彻底地改变预期测试的行为

结构化的覆盖准则

6. 最有效遍历

定义2.39 最有效遍历：令TR_{tour}作为能够遍历的测试需求的子集，$TR_{sidetrip}$是能够用边路遍历的测试需求的子集。注意到$TR_{tour} \subseteq TR_{sidetrip}$。测试路径集T完成最有效遍历，如果对$TR_{tour}$中每条路径p，T中存在某条路径直接遍历p并且对$TR_{sidetrip}$中的每条路径p，T中存在某条路径或直接遍历p，或边路遍历p。

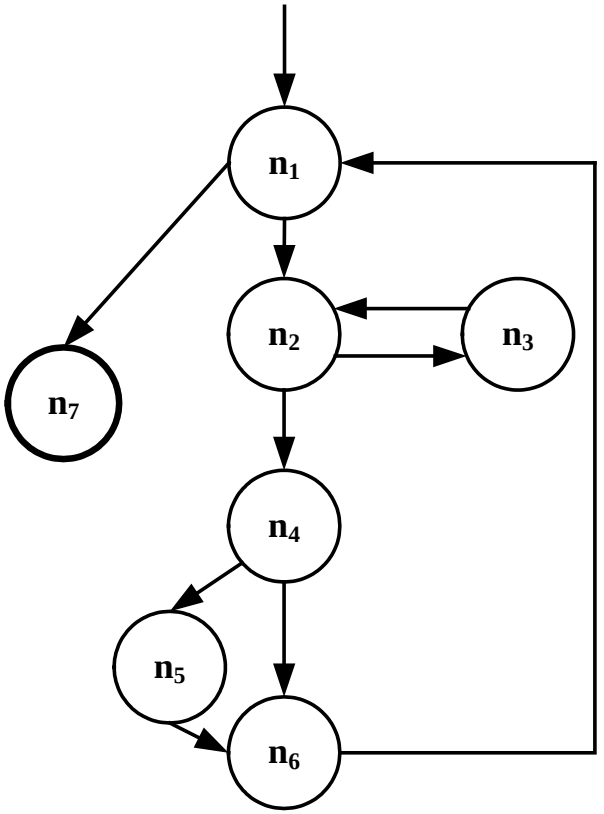
最有效遍历的实际优点是尽可能多的满足测试需求，而且每个测试需求尽可能通过最严格的方式满足

案例分析：寻找主要测试路径

主路径可以通过从长度为0的路径开始寻找，然后扩展到1，等等。

！叹号表示这条路径不能扩展，没有输出的边

*星号表示这条路径不能够进一步扩展，因为初始结点和最后结点相同（已经成为一个环）



案例分析：寻找主要测试路径

长度为0的简单路径7个：

1） [1]

2） [2]

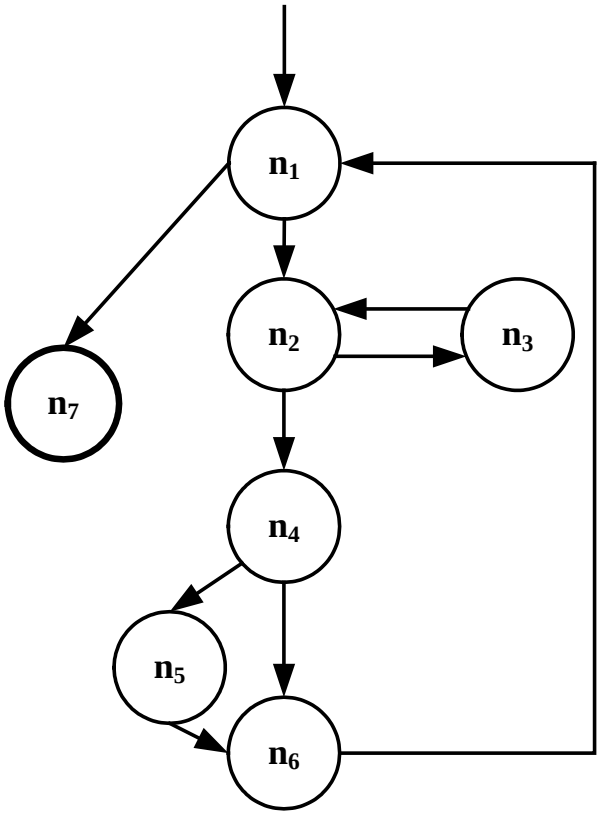
3） [3]

4） [4]

5） [5]

6） [6]

7） [7]!



案例分析：寻找主要测试路径

长度为1的简单路径9个

1） [1,2]

2） [1,7] !

3） [2,3]

4） [2,4]

5） [3,2]

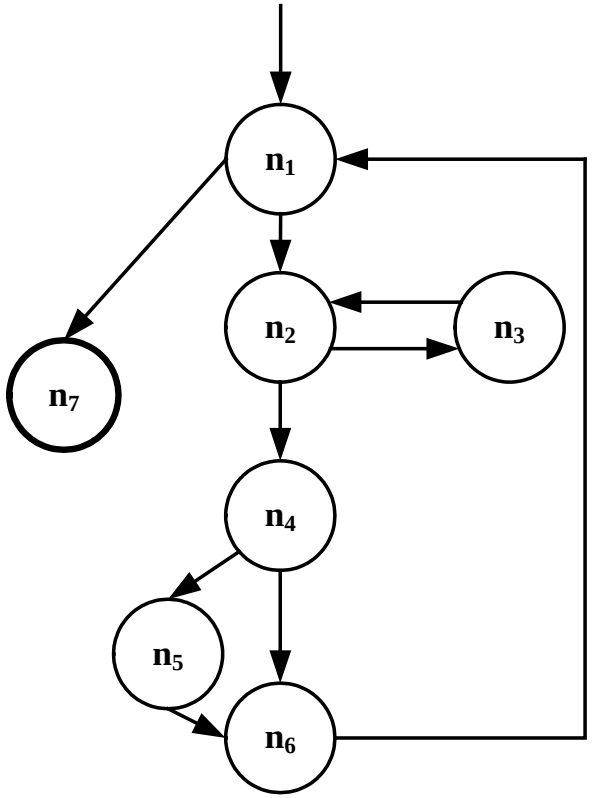
6） [4,5]

7） [4,6]

8） [5,6]

9） [6,1]

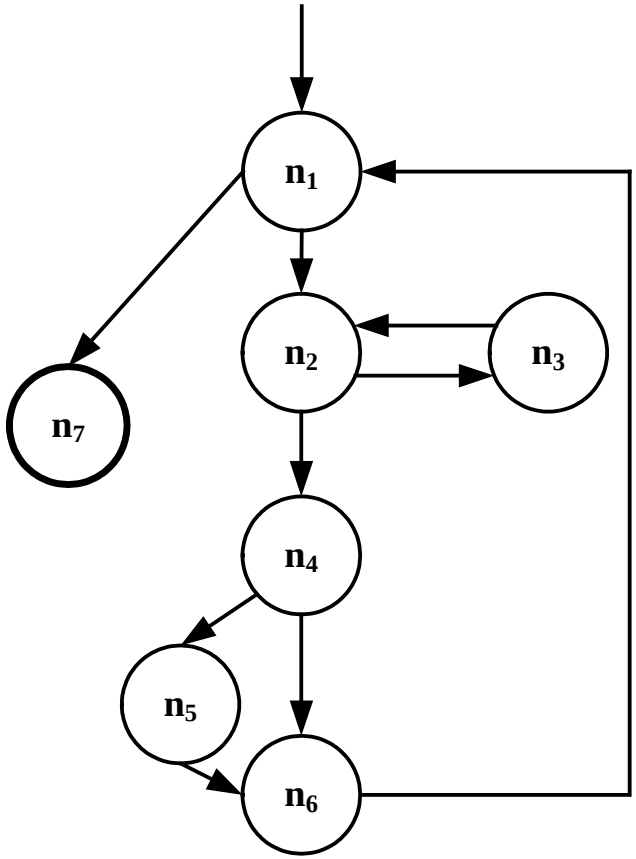
通过每个从路径中终结点可达的结点扩展这条路径，除非该结点已经在路径中，且不是初始结点。



案例分析：寻找主要测试路径

长度为2的简单路径12个

1） [1，2，3]
2） [1，2，4]
3） [2，3，2]*
4） [2，4，5]
5） [2，4，6]
6） [3，2，4]
7） [3，2，3] *
8） [4，5，6]
9） [4，6，1]
10） [5，6，1]
11） [6，1，2]
12) [6，1，7]!



案例分析：寻找主要测试路径

长度为3的简单路径有13个：

1） [1，2，4，5]
2） [1，2，4，6]
3） [2，4，5，6]
4） [2，4，6，1]
5） [3，2，4，5]
6） [3，2，4，6]
7） [4，5，6，1]
8） [4，6，1，7]!
9） [4，6，1，2]
10）[5，6，1，2]
11）[5，6，1，7]!
12）[6，1，2，3]
13）[6，1，2，4]

有2条长度为3的路径不能够扩展，因为它们已到达终结点7；路径[6，1，2，3] 不能够进一步扩展，因为[6，1，2，3，2]并不是简单路径，因此也就不是主要路径。

案例分析：寻找主要测试路径

长度为4的简单路径共15个：

1） [1，2，4，5，6]
2） [1，2，4，6，1]*
3） [2，4，5，6，1]
4） [2，4，6，1，7]!
5） [2，4，6，1，2]*
6） [3，2，4，5，6]
7） [3，2，4，6，1]
8） [4，5，6，1，2]
9） [4，5，6，1，7]!
10） [4，6，1，2，3]
11） [4，6，1，2，4]*
12） [5，6，1，2，3]
13） [5，6，1，2，4]
14） [6，1，2，4，5]
15） [6，1，2，4，6]*

案例分析：寻找主要测试路径

长度为5的简单路径共9个：

1） [1，2，4，5，6，1]*
2） [2，4，5，6，1，2]*
3） [2，4，5，6，1，7]!
4） [3，2，4，5，6，1]
5） [3，2，4，6，1，7]!
6） [4，5，6，1，2，4]*
7） [4，5，6，1，2，3]
8） [5，6，1，2，4，5]*
9） [6，1，2，4，5，6]*

长度为6的路径只有一个
[3，2，4，5，6，1，7]！

可以通过排除任何作为某个其他简单路径的（合适的）子路径的路径来计算主路径。
注意：每条没有叹号和星号的简单路径可以被排除，因为它们可以被扩展，因此是某些其他简单路径的子路径。

案例分析：寻找主要测试路径

13条主路径：

19）[2，3，2]*

23） [3，2，3]*

42）[1，2，4，6，1]*

45）[2，4，6，1，2]*

51）[4，6，1，2，4]*

55）[6，1，2，4，6]*

56）[1，2，4，5，6，1]*

57）[2，4，5，6，1，2]*

60）[3，2，4，6，1，7]!

61）[4，5，6，1，2，4]*

63）[5，6，1，2，4，5]*

64）[6，1，2，4，5，6]*

65）[3，2，4，5，6，1，7]!

找到测试路径来遍历主路径：
从最长的主路径开始，然后将它们扩展到图中的始结点和终结点

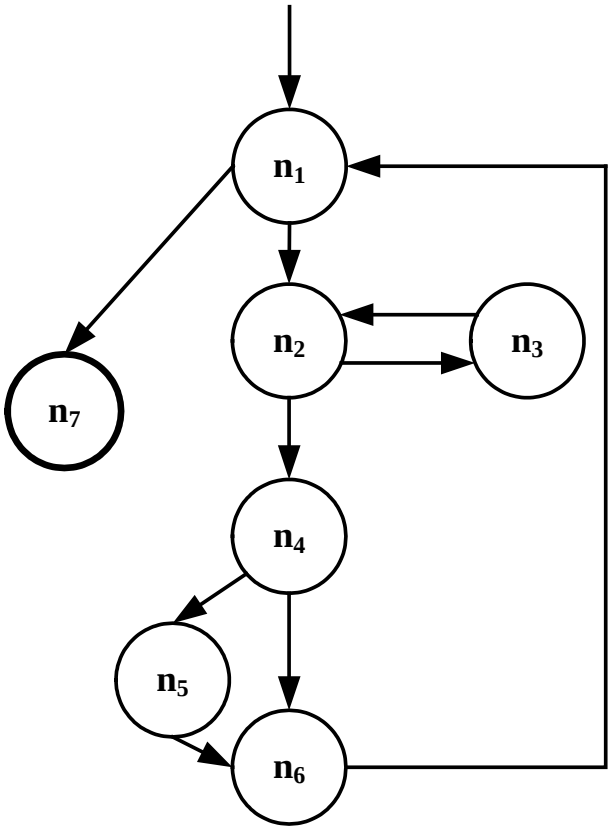
[1，2，4，6，1，2，4，6，1，7]
遍历3条主路径51，45和42

[1，2，3，2，4，6，1，7]

[1，2，4，5，6，1，2，4，5，6，1，7]，
遍历主路径56，57，61和63。

[1，2 ，3，2 ，3，2，4，5，6，1，7]遍历了2条主路径19和23

案例分析：寻找主要测试路径



[1, 2, 4, 6, 1, 2, 4, 6, 1, 7]

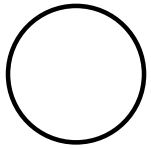
[1, 2, 3, 2, 4, 6, 1, 7]

[1, 2, 4, 5, 6, 1, 2, 4, 5, 6, 1, 7],

[1, 2, 3, 2, 3, 2, 4, 5, 6, 1, 7]

数据流覆盖准则

“du链”—Define and Use Chain

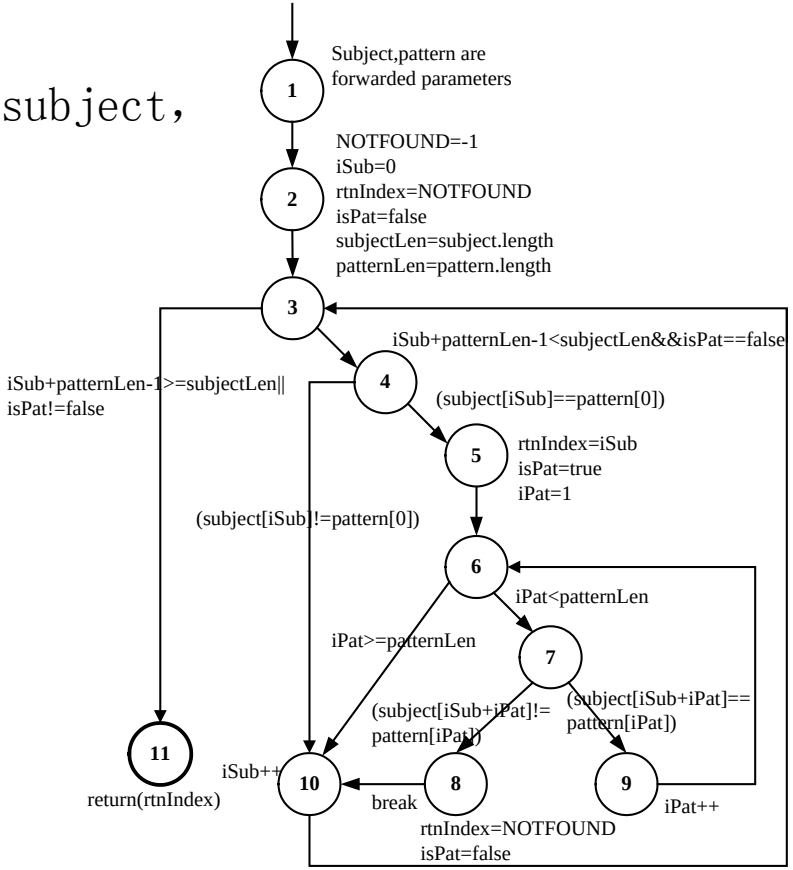


数据流覆盖准则

定义2.40 定义清除路径：从l_i到l_j的路径是关于变量v定义清除路径的是指：如果路径上每个结点n_k和每个边e_k，$k \neq i$且$k \neq j$，v不在集合def（n_k）和def（e_k）中。

数据流覆盖准则

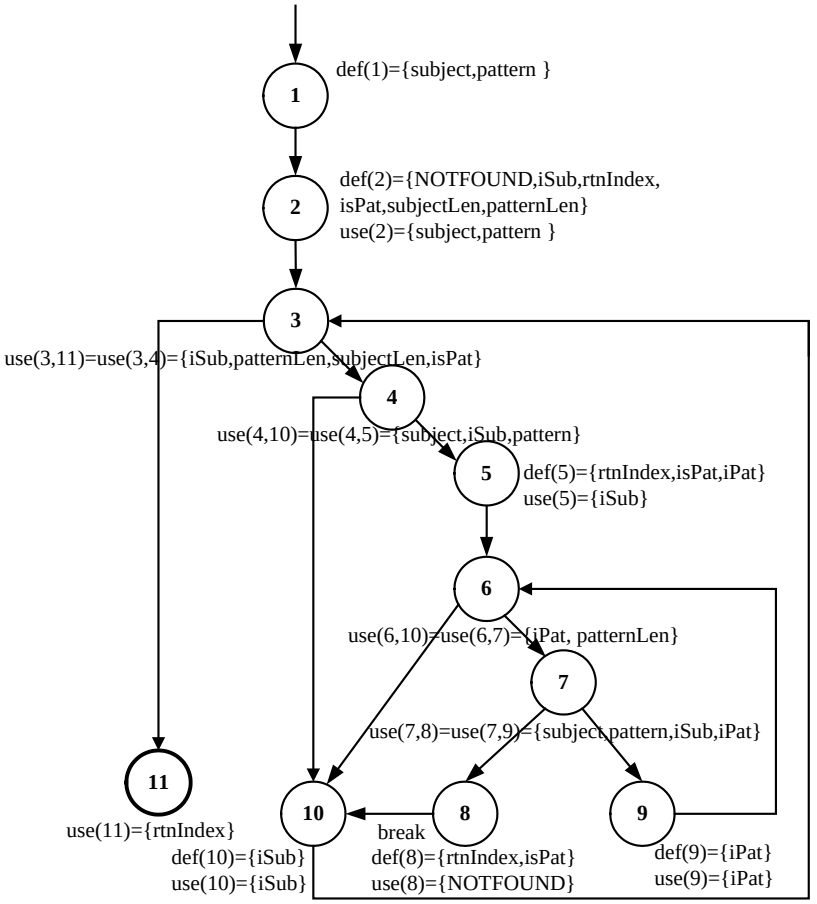
use（4，10）≡use（4，5）≡{subject，iSub，pattern}



数据流覆盖准则

语句iPat++，相当于iPat=iPat+1

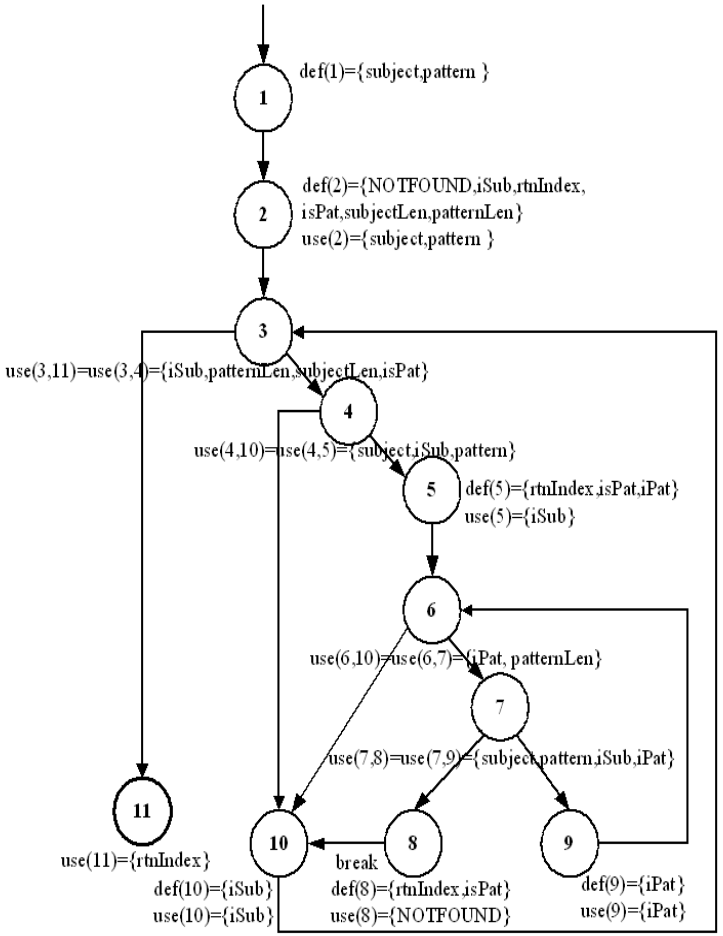
结点5到结点9是iPat的定义清除路径



数据流覆盖准则

结点10中iSub定义路径集为：
du(10, iSub)={[10, 3, 4], [10, 3, 4, 5],
[10, 3, 4, 5, 6, 7, 8], [10, 3, 4, 5, 6, 7, 9],
[10, 3, 4, 5, 6, 10], [10, 3, 4, 5, 6, 7, 8, 10],
[10, 3, 4, 10], [10, 3, 11]}

分为 “定义对”集：
du(10, 4, iSub)={[10, 3, 4]}
du(10, 5, iSub)={[10, 3, 4, 5]}
du(10, 8, iSub)={[10, 3, 4, 5, 6, 7, 8]}
du(10, 9, iSub)={[10, 3, 4, 5, 6, 7, 9]}
du(10, 10, iSub)={[10, 3, 4, 5, 6, 10],
[10, 3, 4, 5, 6, 7, 8, 10], [10, 3, 4, 10]}
du(10, 11, iSub)={[10, 3, 11]}



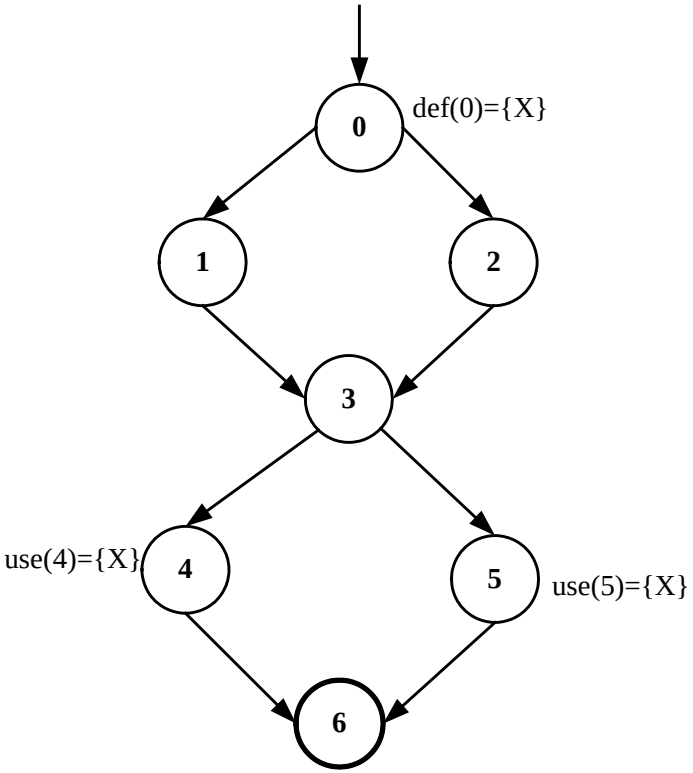
数据流覆盖准则

定义2.40 du遍历：如果测试路径p遍历关于v的子路径d，并且p与d相对应的部分是关于v的定义清除路径，则称测试路径p为du遍历v的子路径d。

准则2.9 全定义覆盖（ADC）：对每一个“定义路径”集合S=du（n，v），TR（测试需求）至少包含S中的一条路径d准则4.10全使用覆盖（AUC）： 对每一个“定义对”集合S=du（ni，nj，v），TR至少包含S中的一条路径d。

准则2.11全du路径覆盖（ADUPC）：对每一个“定义对”集合S=du（ni，nj，v），TR包含S中的每一条路径d。

数据流覆盖准则

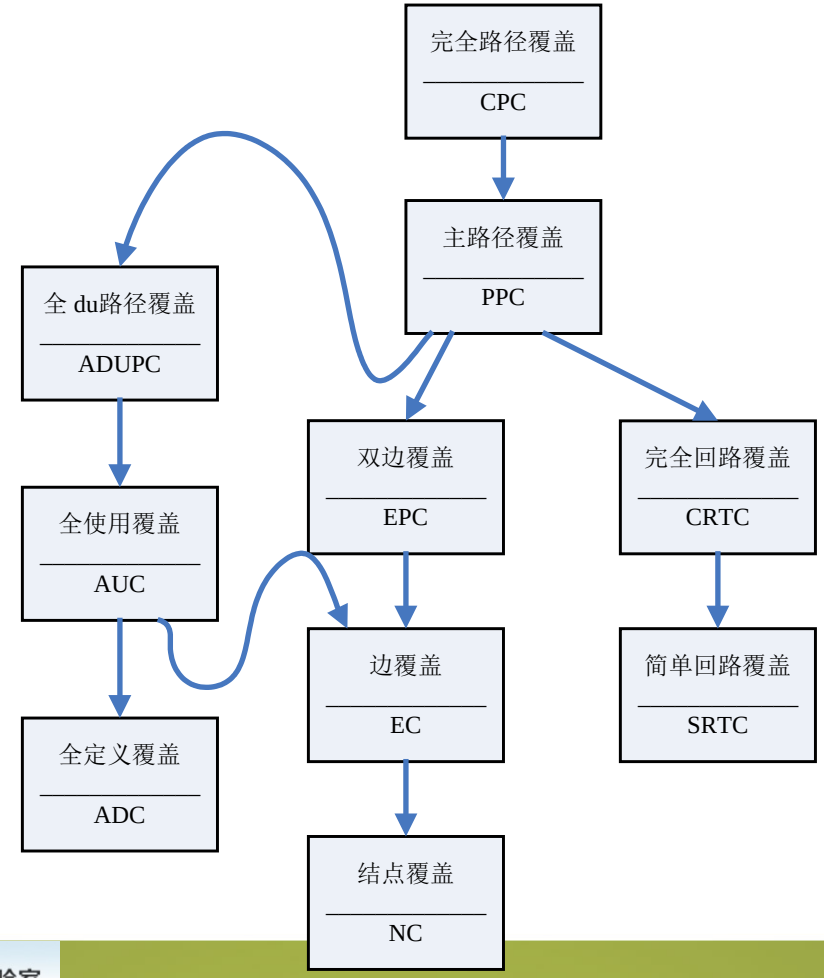


全定义路径
0-1-3-4

全使用路径
0-1-3-4
0-1-3-5

全 DU 路径
0-1-3-4
0-1-3-5
0-2-3-4
0-2-3-5

图覆盖准则之间的包含关系



Test Case is Complex

One test, two faces

One test, two results

```
@Test
public void testFoo ( )
{
    ...
    assertion ( ) ;
    ...
    assertion ( ) ;
    ...
    assertion ( ) ;
    ...
}
```

pass

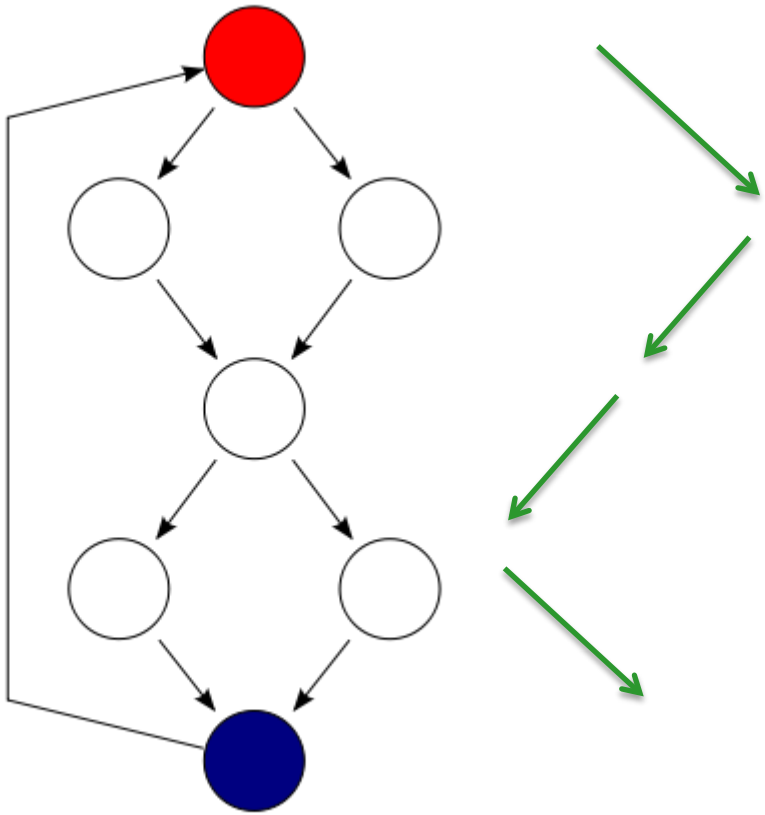
fail

Test Case is Complex

One test, two faces

One test, two results

One test, two executions



We made programmers tolerate too much …

A programmer has to write code to automatically face all test cases.

逻辑覆盖准则

1. 逻辑谓词和子句

谓词是指取值为布尔值的一种表达式，也是最顶端的结构，

$$((a > b) \vee C) \wedge p(x)$$

布尔变量

比较操作符

非布尔变量

函数调用

逻辑操作符

- $\neg$ - 否定操作符
- $\wedge$ - 合取操作符
- $\vee$ - 析取操作符
- $\rightarrow$ - 蕴含操作符
- $\oplus$ - 异或操作符
- $\leftrightarrow$ - 等价操作符

逻辑覆盖准则

1. 逻辑谓词和子句

子句是一个不包含任何逻辑操作符的谓词。

$$((a > b) \vee C) \wedge p(x)$$

子句:
关系表达式(a = b)
布尔变量C
函数调用p(x)

等价形式

$$((a = b) \vee C) \wedge ((a = b) \vee p(x))$$

逻辑覆盖准则

1. 逻辑谓词和子句

逻辑表达式来源

```
if ((a > b)||C)&&(x < y)
    o.m( );
else
    o.n( );
```

$$((a > b) \vee C) \wedge (x < y)$$

逻辑覆盖准则

1. 逻辑谓词和子句

逻辑表达式来源

有限状态机中的状态转换

button2=true(when gear=park)

gear = park burton2 = true

规约的前置条件

"pre: stack not full AND object reference parameter not null"

$\neg\, \mathrm{stackFull}() \wedge \ \mathrm{newObj} \neq \mathrm{null}$

逻辑覆盖准则

2. 逻辑表达式覆盖准则

P为谓词集合，C为P中谓词的子句集

每个谓词 $p \in P$ 令 C_p 为p中的子句，即 $C_p = \{c | c \in p\}$

C为P中是中谓词的子句的并集，$C = \bigcup_{p \in P} C_p$

准则2.12　谓词覆盖（PC）：对每个 $p \in P$ ，TR包含两个需求条件：p值为真，和p值为假

逻辑覆盖准则

2. 逻辑表达式覆盖准则

准则2.12　谓词覆盖（PC）：对每个 $p \in P$，TR包含两个需求条件：p值为真，和p值为假

$$((a < b) \vee C) \wedge p(x)$$

(a=5，b=6，C=true，p(x)=true)
(a=5，b=4，C=false，p(x)=false)

(a=5，b=4，C=true，p(x)=true)
(a=5，b=4，C= true，p(x)=false)

逻辑覆盖准则

2.逻辑表达式覆盖准则

准则2.13　子句覆盖（CC）：对每个 $c \in C$，　TR包含两个需求条件：c值为真，和c值为假

$$((a < b) \vee C) \wedge p(x)$$

((a=5, b=6), (C=true), p(x)=true)
((a=5, b=4), (C=false), p(x)=false)

逻辑覆盖准则

2. 逻辑表达式覆盖准则

子句覆盖并不包括谓词覆盖，且谓词覆盖也不包括子句覆盖

$$p = a \vee b$$

表 4-1 逻辑值组合的四个测试输入

	a	b	$a \vee b$
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

T_{23} = {2, 3}满足子句覆盖，但是不满足谓词覆盖

T24 = {2, 4}满足谓词覆盖，但是不满足子句覆盖

逻辑覆盖准则

2. 逻辑表达式覆盖准则

准则2.14　组合覆盖（CoC）：对每个 $p \in P$，TR具有这样的测试需求：　C_p中的每个子句均可取得真值组合中的每种可能情况。

$$(a \vee b) \wedge c$$

表 4-2　谓词 $(a \vee b) \wedge c$ 完整的真值表

	a	b	c	$(a \vee b) \wedge c$
1	T	T	T	T
2	T	T	F	F
3	T	F	T	T
4	T	F	F	F
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

逻辑覆盖准则

2. 逻辑表达式覆盖准则

n个独立子句的谓词p有2^n种可能的真值指派

$$((a<b)\vee C)\wedge p(x)$$

单个子句“活动”

(a=5，b=4，C=true，p(x)=true)
(a=5，b=6，C=false，p(x)=false)

(a=5，b=4，C=true，p(x)=true)
(a=5，b=4，C= true，p(x)=false)

逻辑覆盖准则

3 活动子句覆盖准则

主子句c_i是我们重点关注的子句，所有其他的子句 $c_j, j \neq i$ 为次子句

定义2.41　判定：给定谓词p的一个主子句c_i，如果次子句$c_j \in p, j \neq i$ 可取值使改变c_i的真值即可改变p的真值成立，则称c_i判定p。

$p = a \vee b$

如果b为假，那么子句a判定p。因为此时p的值确实是a的值。如果b值为真，那么a不能判定p，因为无论a的值如何p值均为真。

逻辑覆盖准则

3. 活动子句覆盖准则

从测试的角度看，我们希望在子句判定谓词的条件下测试每个子句。

这个问题就像是让一个队伍中的不同成员管理队伍一样，在他们尝试之前我们并不知道谁可能是有力的领导者。

考虑谓词p = a ∨ b，如果在b判定p的条件下我们不改变b的值，那么我们无从知道b是否得到正确的使用。

测试集T_{14} = { TT, FF}既满足子句覆盖也满足谓词覆盖，但均不能有效地测试a和b

逻辑覆盖准则

3. 活动子句覆盖准则

定义2.42 活动子句覆盖（ACC）：对每个 $p \in P$ 及每个主子句 $c_i \in C_p$，选择次子句 $c_j \in p, j \neq i$ 使得c_i判定p。TR对每个c_i包含两个需求： c_i值为真和c_i值为假。

p=a ∨ b

a判定p,测试需求

{(a=true, b= false), (a=false, b=false)}

b判定p ,测试需求

{(a=false, b=true), (a=false, b=false)}

n个子句的谓词， n+1个不同的测试需求可充分满足活动子句覆盖，而不是2n个

逻辑覆盖准则

3. 活动子句覆盖准则

准则2.15 一般活动子句覆盖（GACC）：对每个 $p \in P$ 及每个主子句 $c_i \in C_p$ ，选择次子句 $c_j \in p, j \neq i$ 使得c_i判定p。TR对每个c_i包含两个需求： c_i值为真和c_i值为假。当c_i值为真及假时，次子句c_j的所选值不必要相同.

逻辑覆盖准则

3. 活动子句覆盖准则

准则 2.16 相关活动子句覆盖(CACC)：对每个 $p \in P$ 及每个主子句 $c_i \in C_p$，选择次子句 $c_j \in p, j \neq i$ 使得c_i判定p。TR对每个c_i包含两个需求： c_i值为真和c_i值为假。当c_i值为真及假时，次子句c_j的所选值必须使得：对主子句c_i的一个值来说使p为真，取其他值时p为假。

逻辑覆盖准则

3. 活动子句覆盖准则

$$a \wedge (b \vee c)$$

对 a 判定 p 的值，那么表达式b ∨ c的值必须为真。这个条件可以用以下三个方法来获得： b 为真c为假， b 为假c为真， b 、c均为真。

关于子句a满足CACC:

两个测试输入{TTF, FFT}

表 4-4 $a \wedge (b \vee c)$满足 CACC 可能的选择

	a	b	c	$a \wedge (b \vee c)$
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F

逻辑覆盖准则

3. 活动子句覆盖准则

准则2.17 受限活动子句覆盖（RACC）：对每个 $p \in P$ 及每个主子句 $c_i \in C_p$ 选择次子句 $c_j \in p, j \neq$ 使得c_i判定p。TR对每个c_i包含两个需求： c_i值为真和c_i值为假。当c_i值为真及假时，次子句c_j的所选值必须相同。

逻辑覆盖准则

3. 活动子句覆盖准则

$$a \wedge (b \vee c)$$

关于子句a满足CACC的九个测试需求集中只有三个关于子句a满足RACC 。

表 4-5　$a \wedge (b \vee c)$满足 RACC 可能的选择

	a	b	c	$a \wedge (b \vee c)$
1	T	T	T	T
5	F	T	T	F
2	T	T	F	T
6	F	T	F	F
3	T	F	T	T
7	F	F	T	F

白盒测试与代码覆盖

欢迎提问