对判定的测试（上）

逻辑覆盖：对判定的测试

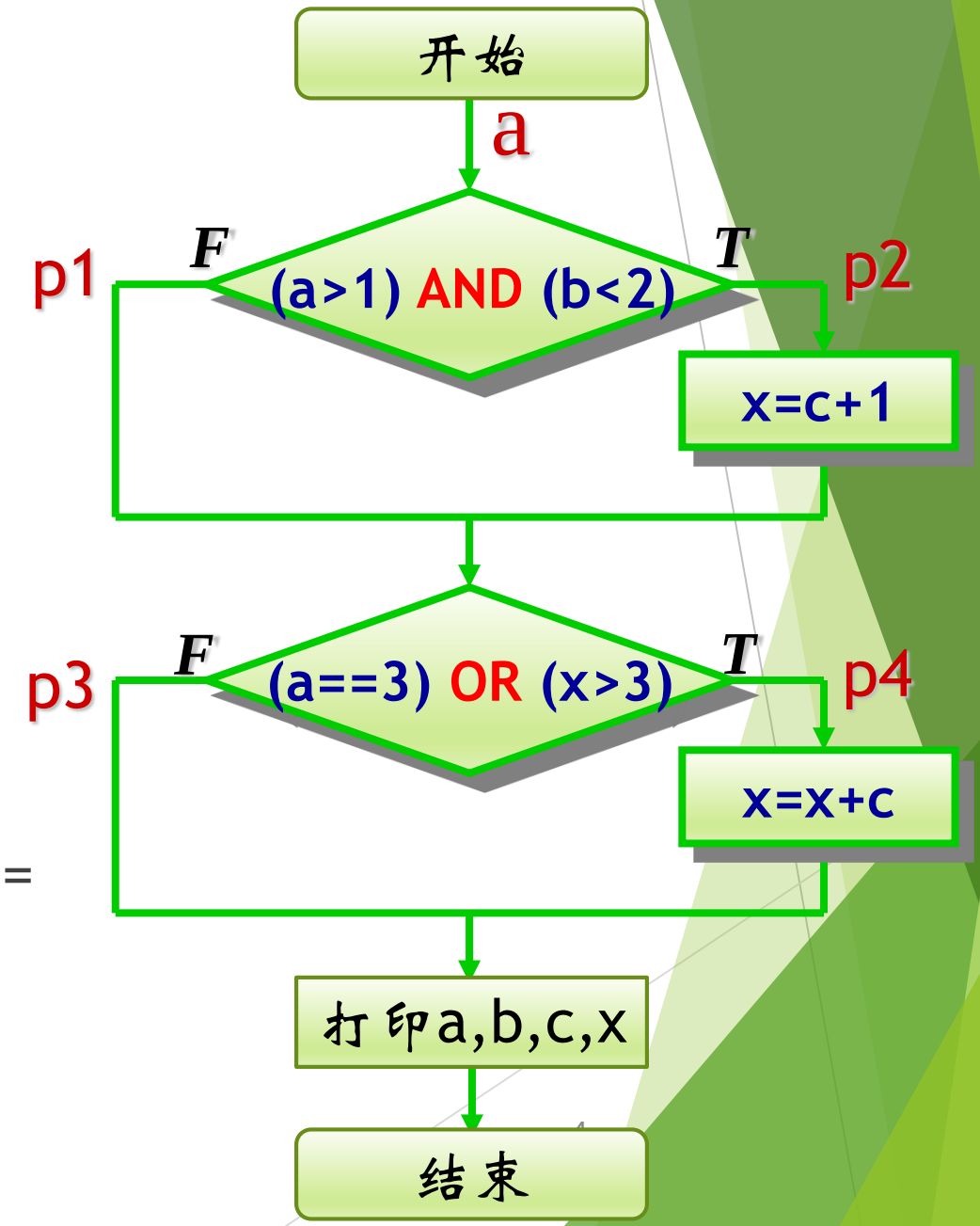
- 关注点：判定表达式本身的复杂度
- 原理：通过对程序逻辑结构的遍历，来实现测试对程序的覆盖
- 原则：对程序代码中所有的逻辑值，都需要测试真值（True）和假值（False）的情况

要解决的问题

- 对于选定的覆盖指标，如何控制测试用例规模，提高测试用例典型性，避免测试漏洞？
- 如何选择合适的覆盖指标？

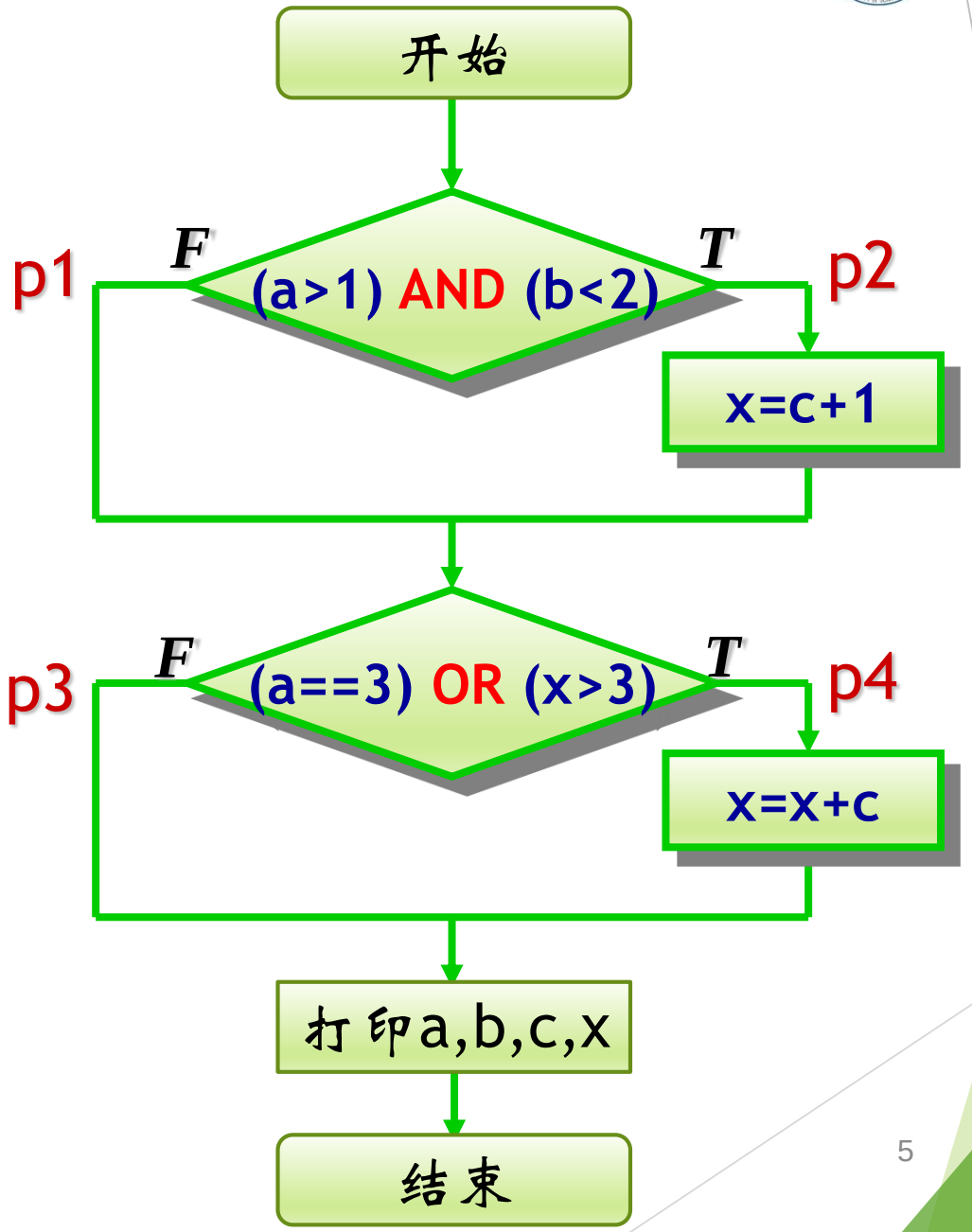
使用的例子

```
int Func6( int a, int b, int c, int x )
{
1  if( (a>1) && (b<2) )
2    x = c + 1;
3  if( (a==3) || (x>3) )
4    x = x + c;
5  printf( "a = %d, b = %d, c = %d, x = %d\n", a, b, c, x );
6  return x;
}
```



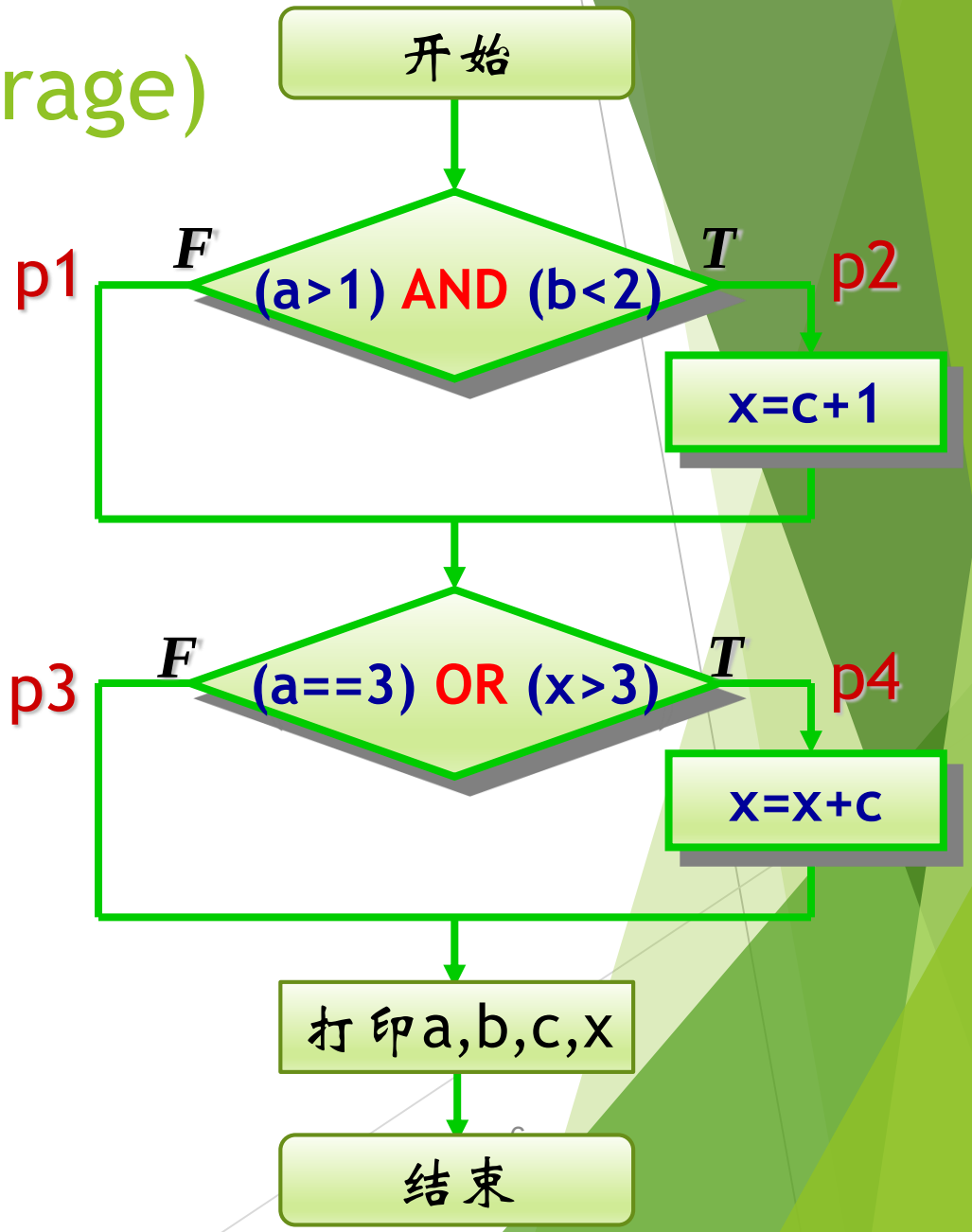
使用的例子

- 4个基本逻辑判定条件
  - T1: a>1
  - T2: b<2
  - T3: a==3
  - T4: x>3
- 4条执行路径
  - L13: p1→p3
  - L14: p1→p4
  - L23: p2→p3
  - L24: p2→p4

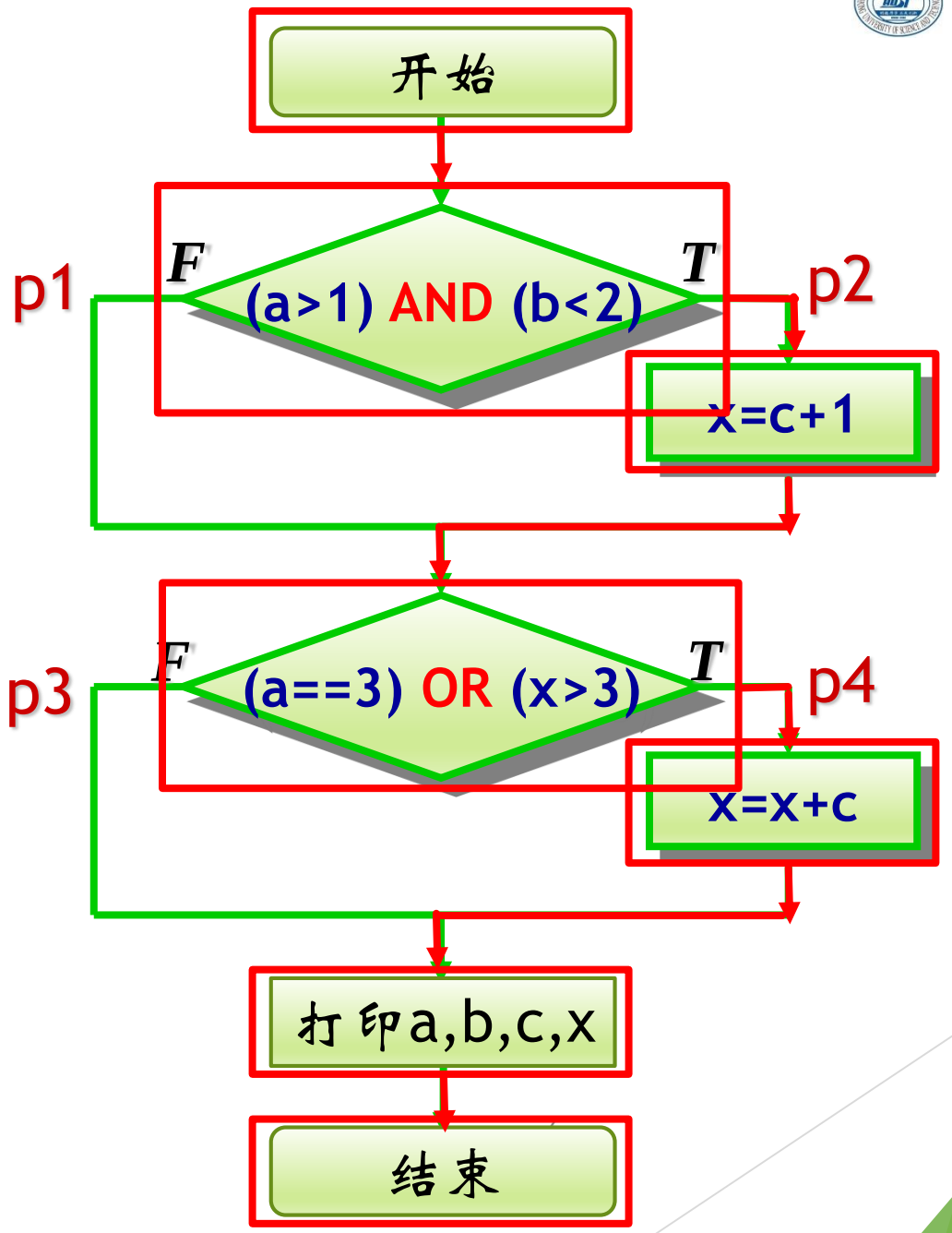


1. 语句覆盖(Statement coverage)

- 设计测试用例时应保证程序中每一条可执行语句至少应执行一次
- 点覆盖



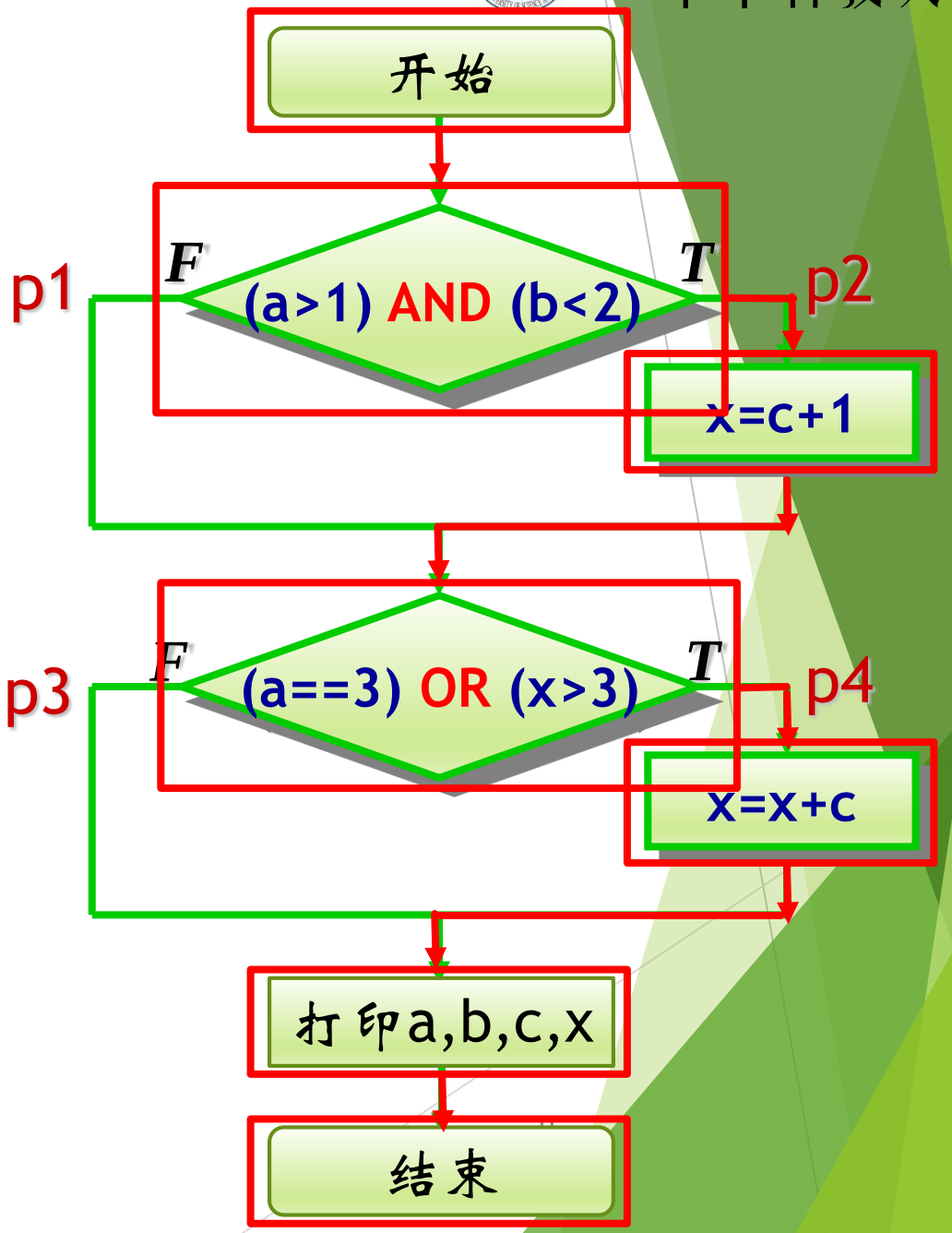
测试用例设计



测试用例设计

输入条件	测试用例	测试用例
T1:a>1	T	T
T2:b<2	T	T
T3:a=3	T	T
T4:x>3	T	F
(a>1) AND (b<2)	T	T
(a==3) OR (x>3)	T	T
覆盖率	**100%**	**100%**

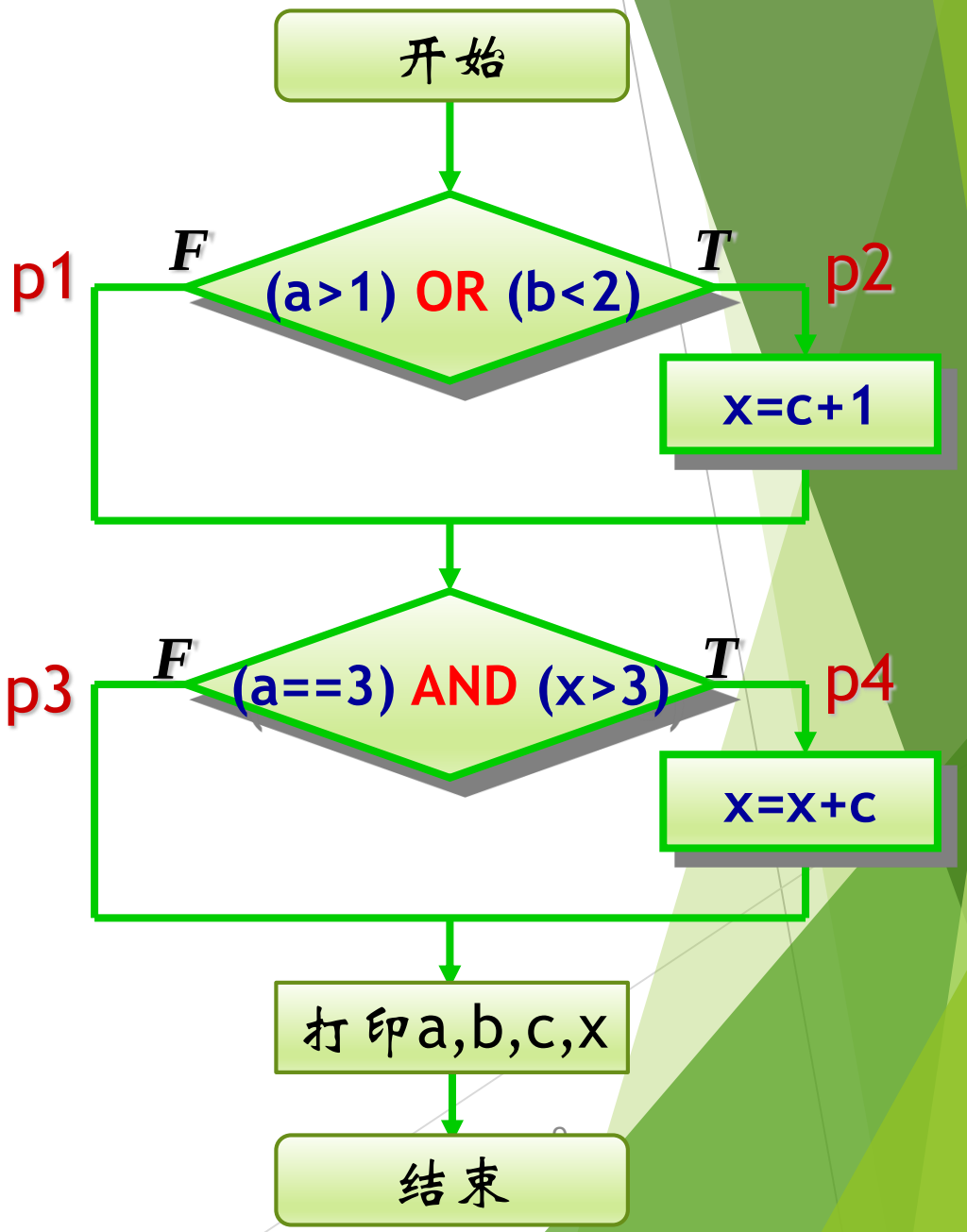
测试用例	输入				预期输出
	a	b	c	x	x
TC1	3	1	3	0	7
TC2	3	1	2	0	5



存在缺陷的程序

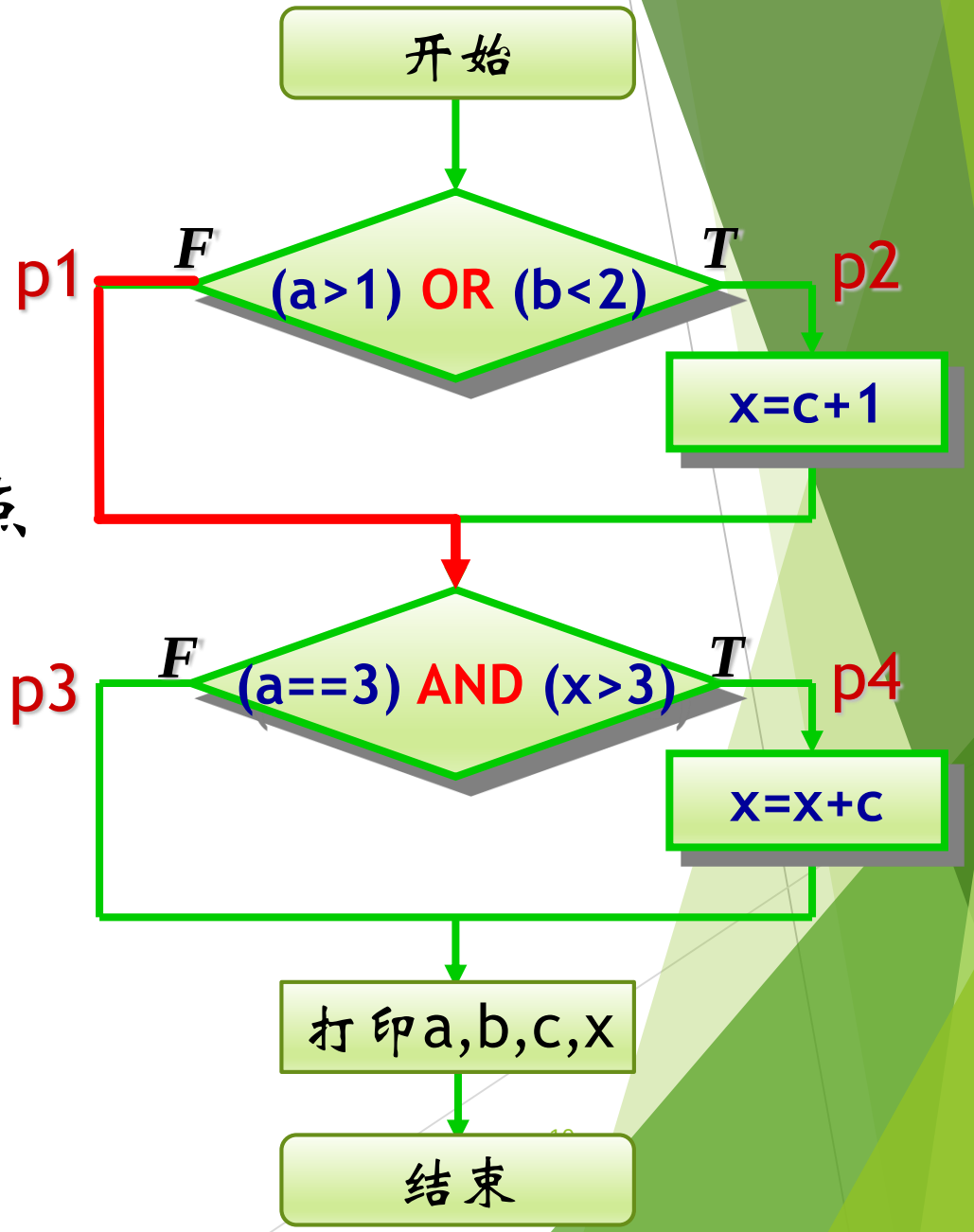
测试用例	输入				预期输出
	a	b	c	x	x
TC1	3	1	3	0	7
TC2	3	1	2	0	5

- TC1：不能发现缺陷
- TC2：可以部分发现缺陷



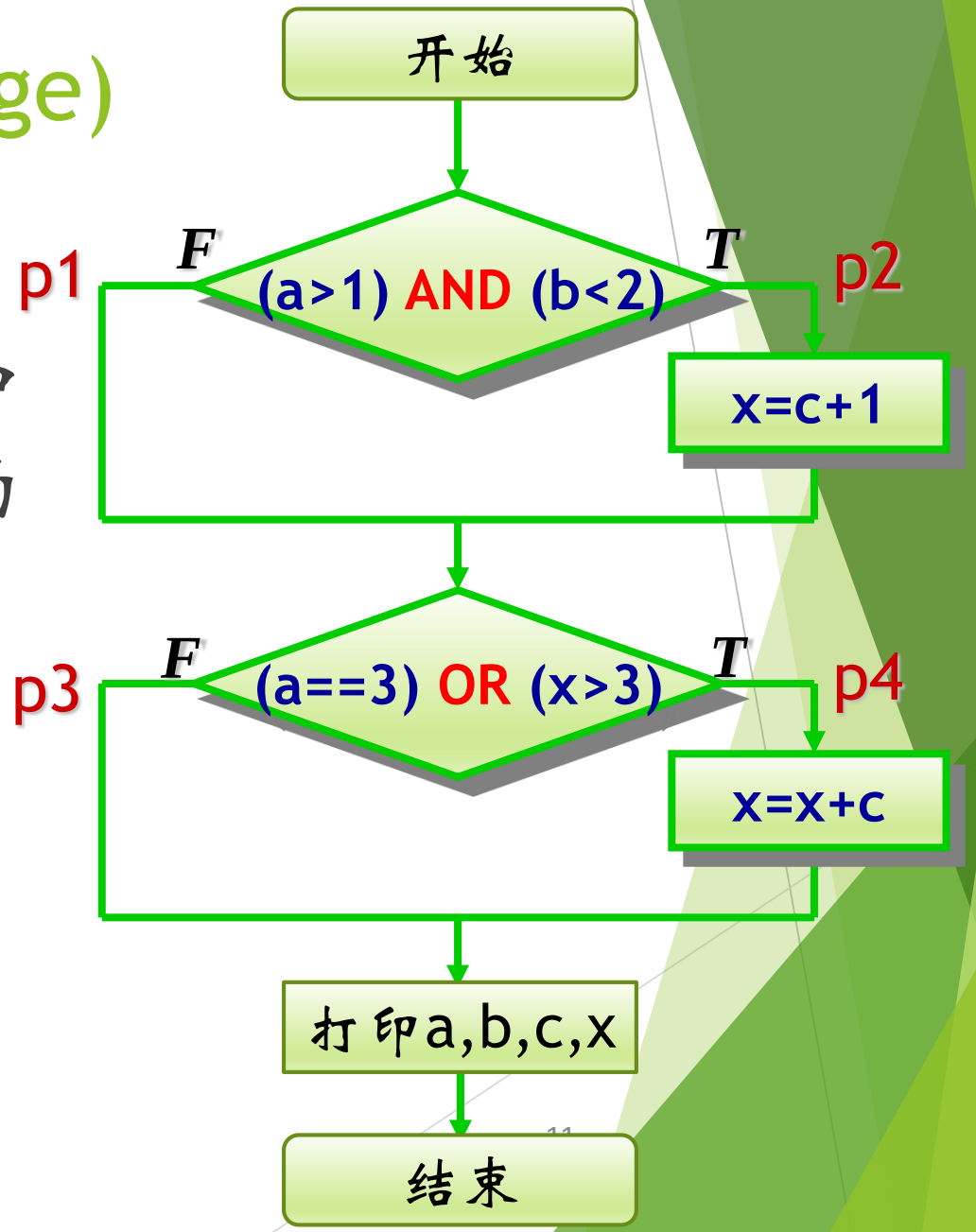
语句覆盖是最弱的覆盖准则

- 局限性
  - 关注语句，而非关注判定节点
  - 对隐式分支无效
- 对策
  - 优选测试数据
  - 更强的覆盖准则：判定覆盖



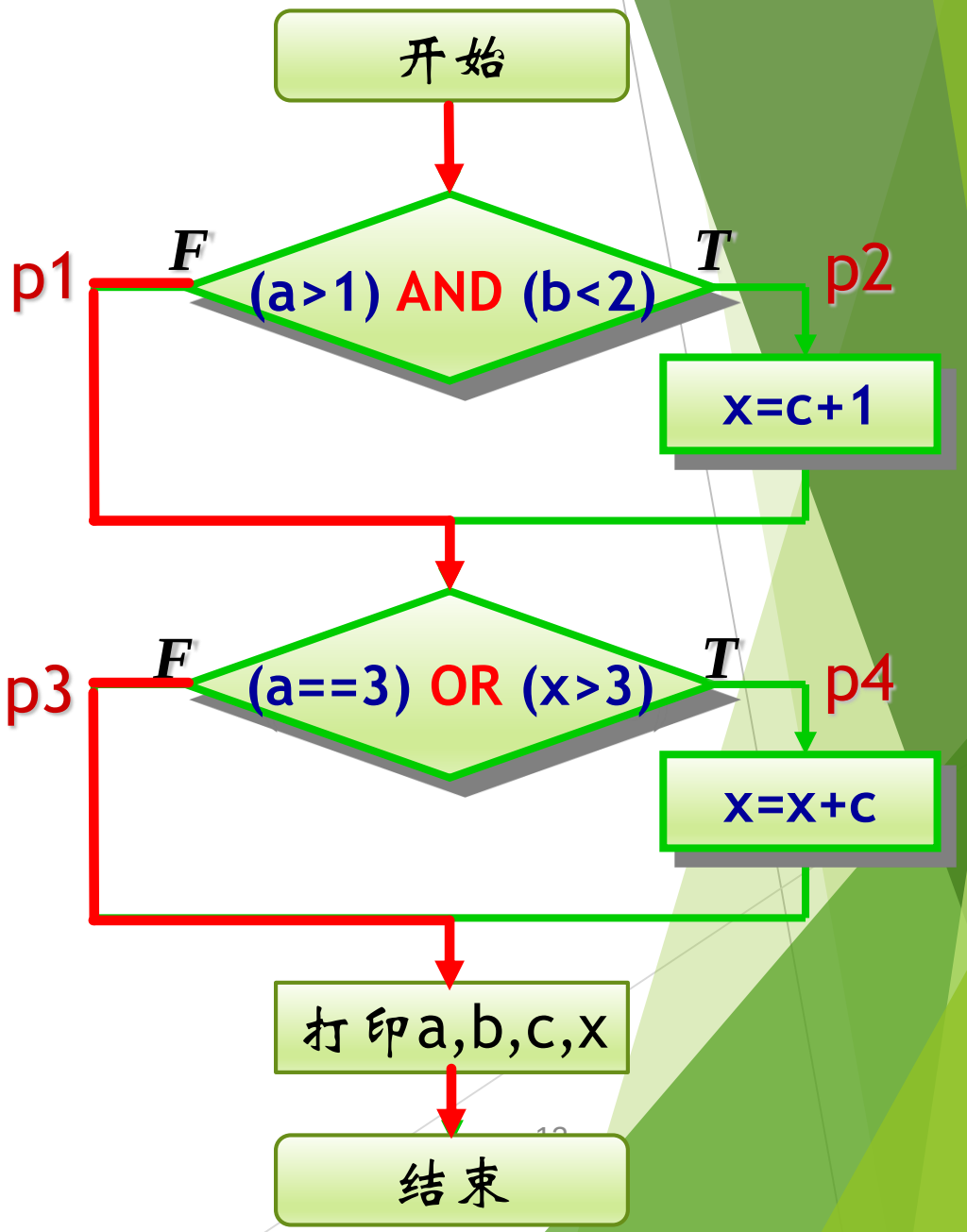
2. 判定覆盖(Branch coverage)

- 设计测试用例时应保证程序中每个判定节点取得每种可能的结果至少一次。
- 边覆盖



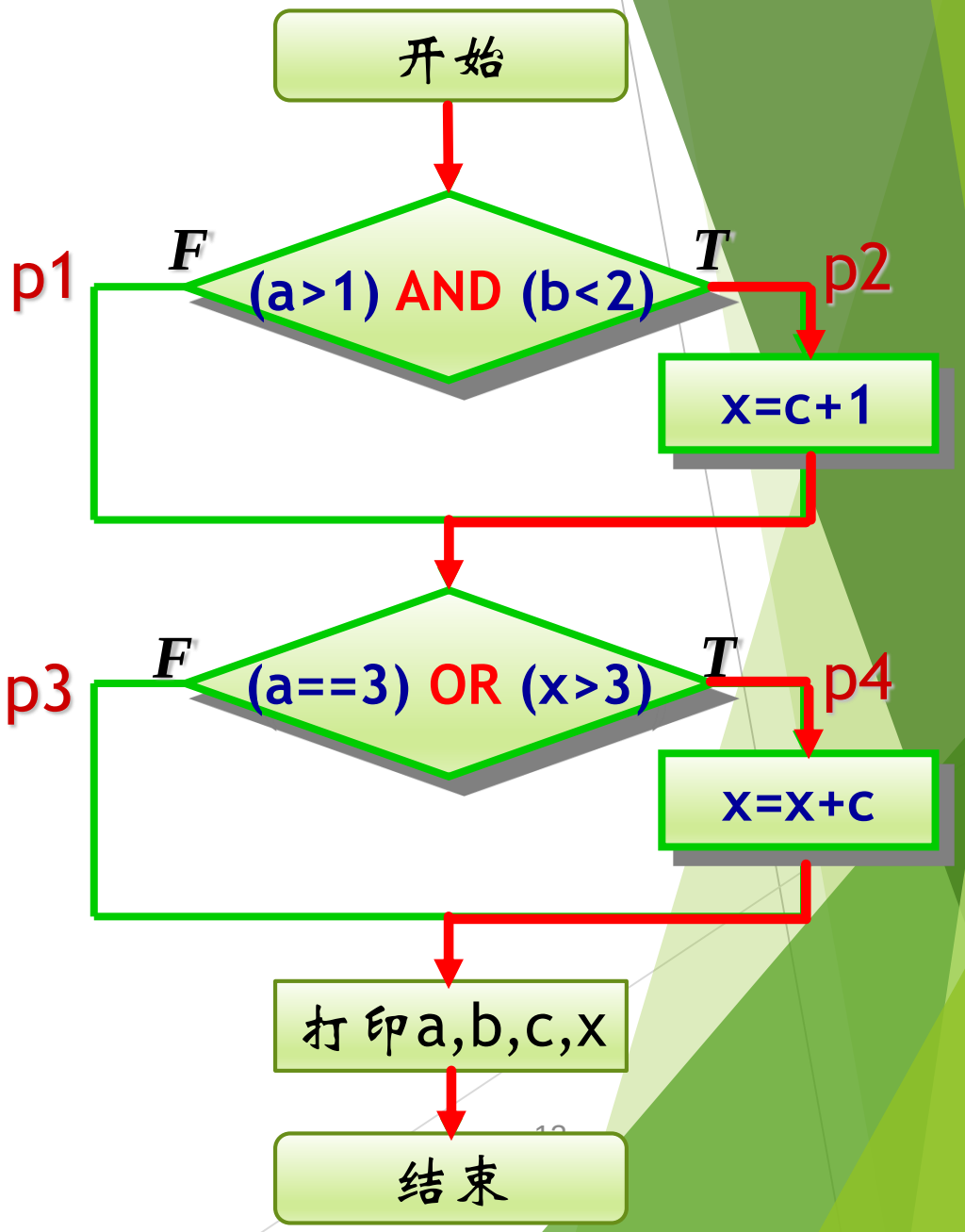
测试用例设计

- 满足判定覆盖
- 用例需要执行路径L13



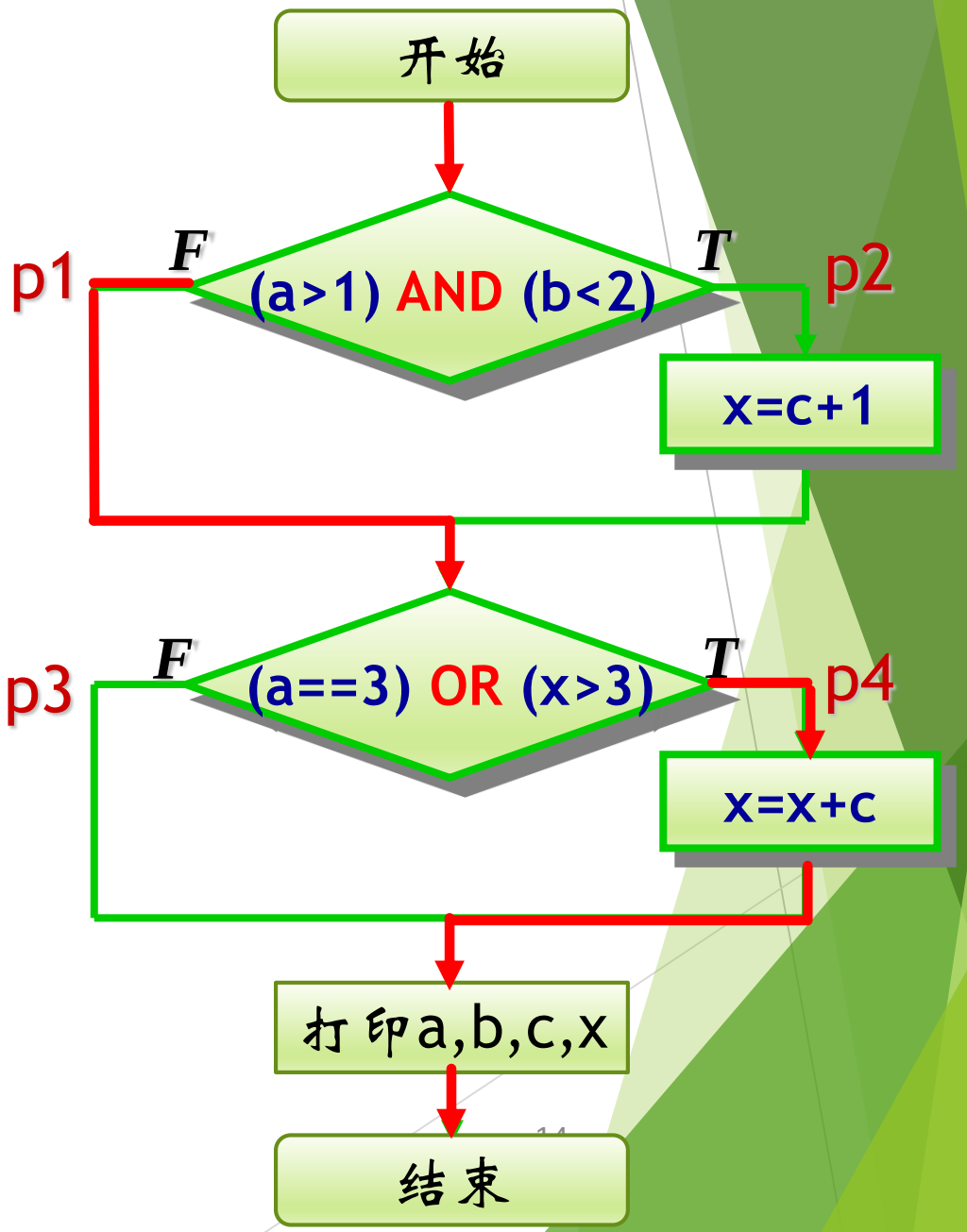
测试用例设计

- 并执行路径L24



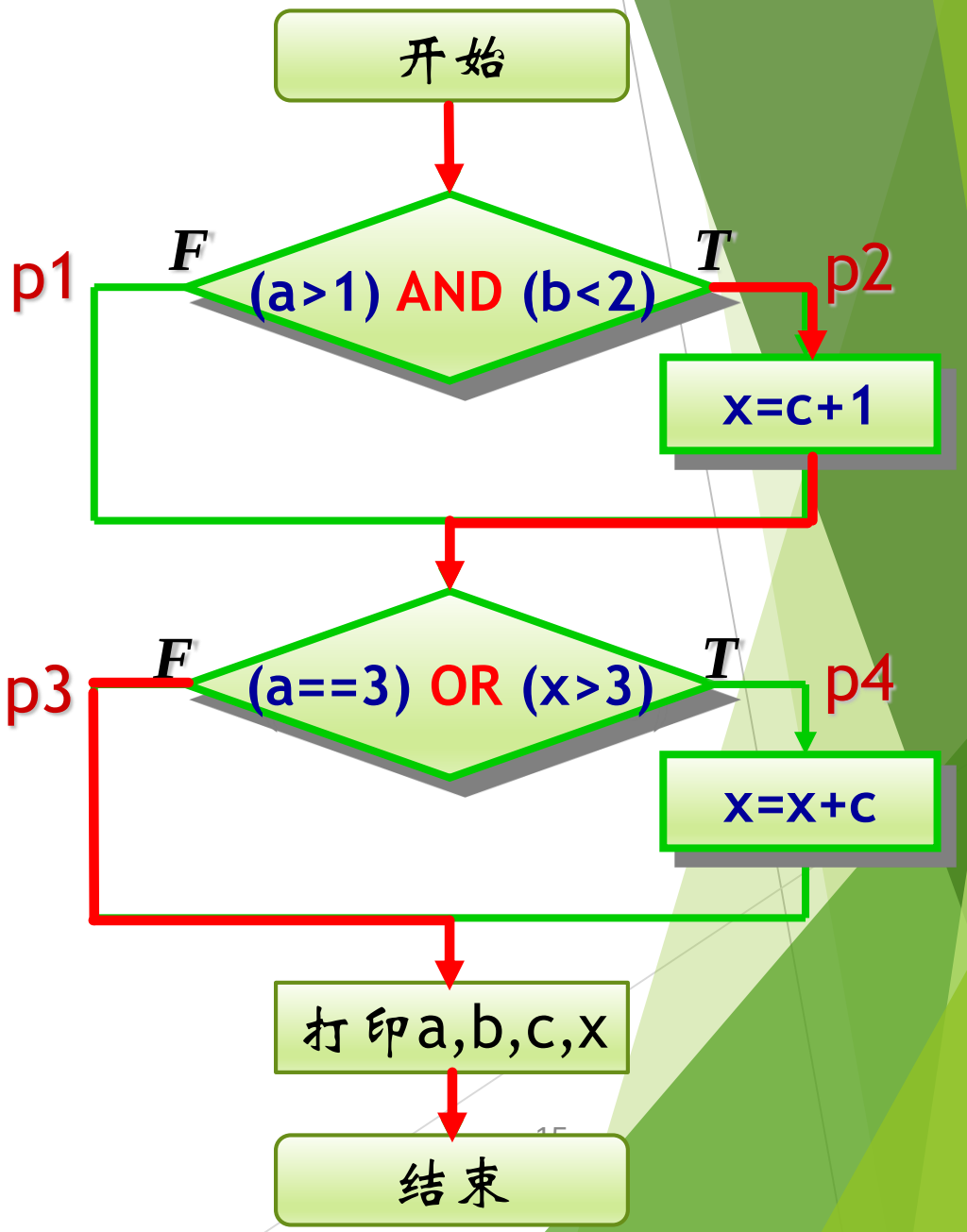
测试用例设计

- 用例需要执行路径L14



测试用例设计

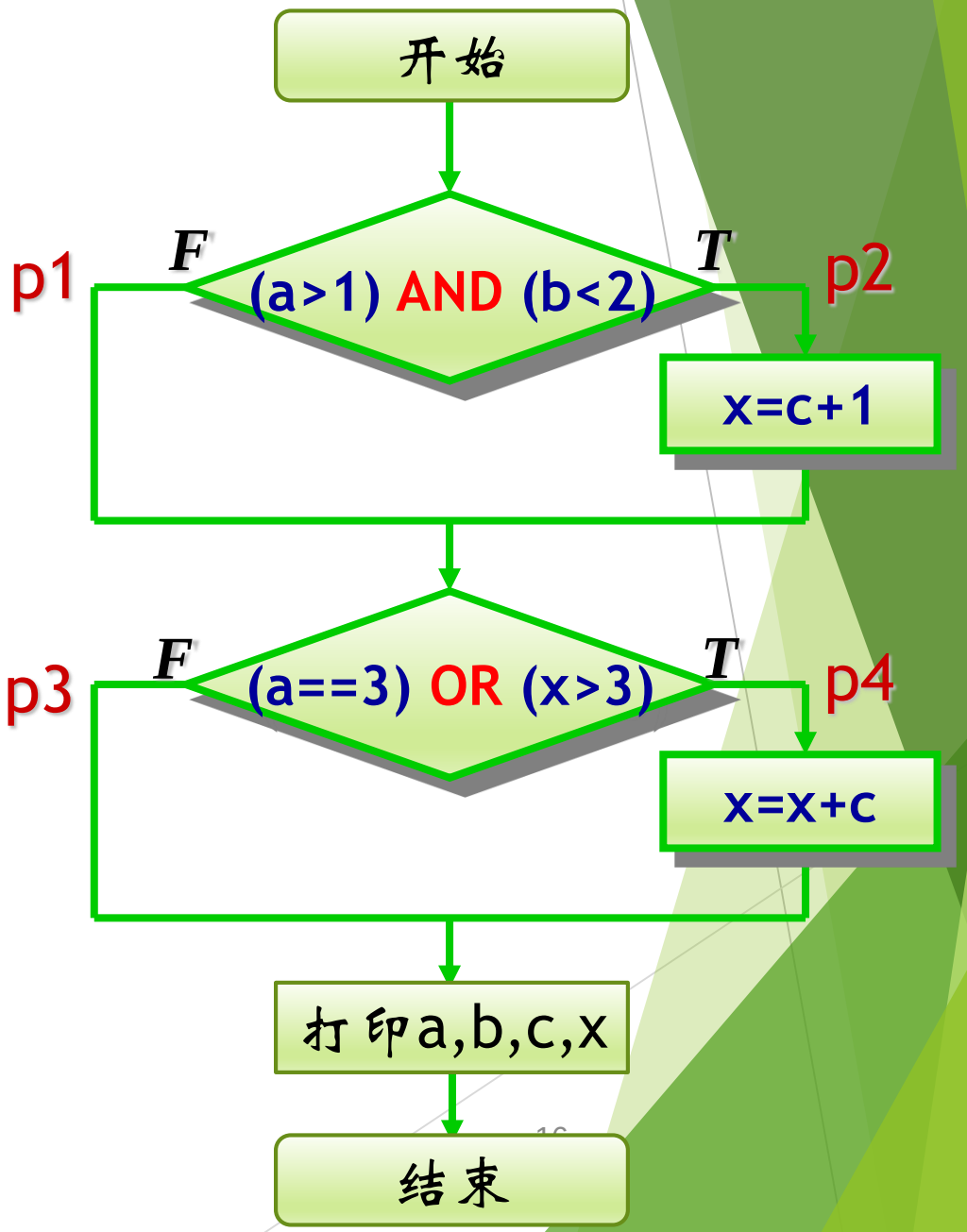
- 并执行路径L23



测试用例设计

ID	输入				预期输出	通过路径	判定覆盖	语句覆盖
	a	b	c	x				
TC3	2	1	3	0	7	L24	100%	100%
TC4	4	2	2	0	0	L13		
TC5	1	2	2	4	6	L14	100%	100%
TC6	2	1	1	2	2	L23		

判定覆盖可以发现判定表达式中所有的缺陷吗？



存在缺陷的程序

ID	输入				预期输出	通过路径	判定覆盖	语句覆盖
	a	b	c	x				
TC3	2	1	3	0	7	L24	100%	100%
TC4	4	2	2	0	0	L13		
TC5	1	2	2	4	6	L14	100%	100%
TC6	2	1	1	2	2	L23		

- TC3+TC4：可以发现缺陷
- TC5+TC6：不能发现缺陷