第二部分 软件测试基础知识

2.12 动态白盒测试（2）

本节教学目标

- 独立路径测试要点分析
- 重难点：独立路径法设计测试用例

内容回顾

- 动态白盒测试：
  - 逻辑覆盖
  - 路径覆盖
    - 根据程序源代码，画出程序图
    - 计算环复杂度
    - 设计独立路径（去掉不可行路径，增加必要路径）
    - 转换测试用例

目录

基于独立路径测试分析

- Path1:A,6,8,12,13,16,18,20,21,B,34,35
- Path2:A,6,7,16,18,20,21,B,34,35
- Path3:A,6,8,9,16,18,20,21,B,34,35
- Path4:A,6,8,12,15,16,18,20,21,B,34,35
- Path5:A,6,8,12,13,16,18,33,34,35
- Path6:A,6,8,12,13,16,18,20,21,28,34,35

路径测试

- 路径测试的修改：
- 独立路径提取：
  - Path1:A,6,8,12,13,16,18,20,21,28,34,35
  - Path2:A,6,7,16,18,20,21,B,34,35
  - Path3:A,6,8,9,16,18,20,21,28,34,35
  - Path4:A,6,8,12,13,16,18,20,21,28,34,35
  - Path5:A,6,8,12,13,16,18,33,34,35

NextDate函数路径测试分析

- NextDate函数各独立路径的执行概率

序号	独立路径	访问的判定分支	执行概率	备注
1	Path1	E3,e6,e8,e13,e17	0.059%	测试12月31日的情况
2	Path2	E2,e3,e16	0.28%	测试31天月份的月末日期
3	path3	E3,e5,e13,e17	0.94%	测试30天月份的月末日期
4	Path4	E3,e6,e9,e13,e17	0.18%	测试非闰年的2月月末
5	path5	Ee,e6,e8,e14	2.02%	测试闰年的2月普通日期

- NextDate函数各补充路径的执行概率

序号	独立路径	访问的判定分支	执行概率	备注
1	Path6	E2,e14	56.38%	测试31天月份的普通日期
2	Path7	E3,e5,e14	32.26%	测试31天月份的普通日期
3	Path8	E3,e6,e9,e14	6.05%	测试非闰年的2月普通日期

NextDate函数路径测试

- NextDate函数路径测试用例

序号	输入	预期输出	备注
001	2012-12-31	2013-1-1	Path1
002	2012-7-31	2012-8-1	Path2
003	2012-6-30	2012-7-1	Path3
004	2011-2-28	2011-3-1	Path4
005	2012-2-15	2012-2-16	Path5
006	2012-7-15	2012-7-16	Path6
007	2012-6-15	2012-6-16	Path7
008	2011-2-15	2011-2-16	Path8

基于独立路径测试分析

- 基于程序图和环复杂度的路径测试方法有效吗？
- 能保证测试的完备性和无冗余吗？
  - 从NextDate函数的例子中看，由于存在不可行路径，使得最终得到的独立路径集合不能保证测试的完备性
  - 补充3条路径后，又不能保证测试的无冗余性
  - 所有独立路径执行概率远远低于补充路径的概率，如果不补充，反而不能保证大概率情况下程序的正常处理
- 独立路径测试有意义吗？
- 其实基于独立路径测试方法具有重要意义

基于独立路径测试分析

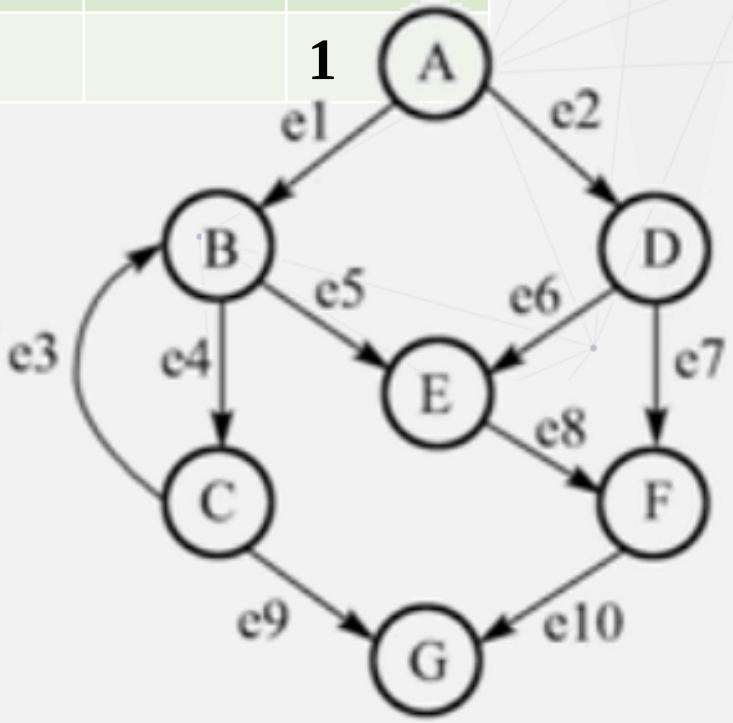
- 独立路径测试的理论基础保证了测试的完备性和无冗余性
  - 情况一：该路径唯一包含某些判定分支，若将该路径剔除，其唯一包含的这些判定分支将无法访问到
  - 情况二：该路径包含的所有判定分支都是在其他路径中访问过的，但该路径对多个判定分支的组合情况，访问方式是唯一的，一旦剔除，则测试也存在漏洞

基于独立路径测试分析

Path	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
Path1	1			1					1	
Path2		1				1		1		1
Path3	1				1			1		1
Path4	1		1	1					1	
Path5		1					1			1
path6	1		1	1	1			1		1

- 从表中可以看出path2—path5均存在唯一访问的表，因此这4条路径是不可取代的，是独立的

- Path1中的边不具备唯一性，若用其他边表示，必然引入新的边，如：用path2表示path1,会引入e2,e6等；因此path1符合情况二，也属于一条独立路径



基于独立路径测试分析

- 补充一条新的路径path6,根据行列式的计算可知：
  - **Path6 = -path1 + path3 + path4**
- 对于存在不可行路径的程序，以上结论是否仍然成立呢？

基于独立路径测试分析

- Path1:A,6,8,12,13,16,18,20,21,B,34,35
- Path2:A,6,7,16,18,20,21,B,34,35
- Path3:A,6,8,9,16,18,20,21,B,34,35
- Path4:A,6,8,12,15,16,18,20,21,B,34,35
- Path5:A,6,8,12,13,16,18,33,34,35
- Path6:A,6,8,12,13,16,18,20,21,28,34,35

基于独立路径测试分析

Path	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12	e13	e14	e15	e16	e17	e18	e19	e20	e21
Path1	1		1			1		1		1		1	1		1		1		1		1
Path2	1	1		1								1	1		1	1		1			1
Path3	1		1		1		1					1	1		1		1		1		1
Path4	1		1			1			1		1	1	1		1		1		1		1
Path5	1		1			1		1		1		1		1						1	1
Path6	1	1		1								1		1						1	1

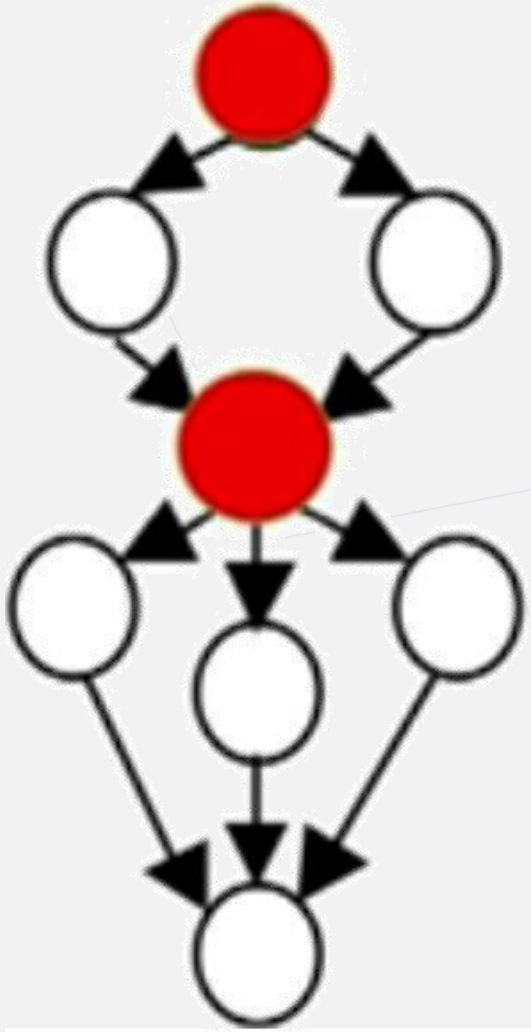
基于独立路径测试分析

- path2---path5均存在唯一访问的边，因此这4条路径是相互独立的，path1中包含的所有边尽管不具备唯一性，但当试图使用其他路径来表示时，必然引入新的边，但是path1满足情况二的独立路径
- Path6 = -path1 + path2 + path5
- Path6 是冗余路径
- NextDate函数源代码中不可行路径导致抽取独立路径时无法得到规定数量的路径，为了避免测试漏洞，需要补充路径

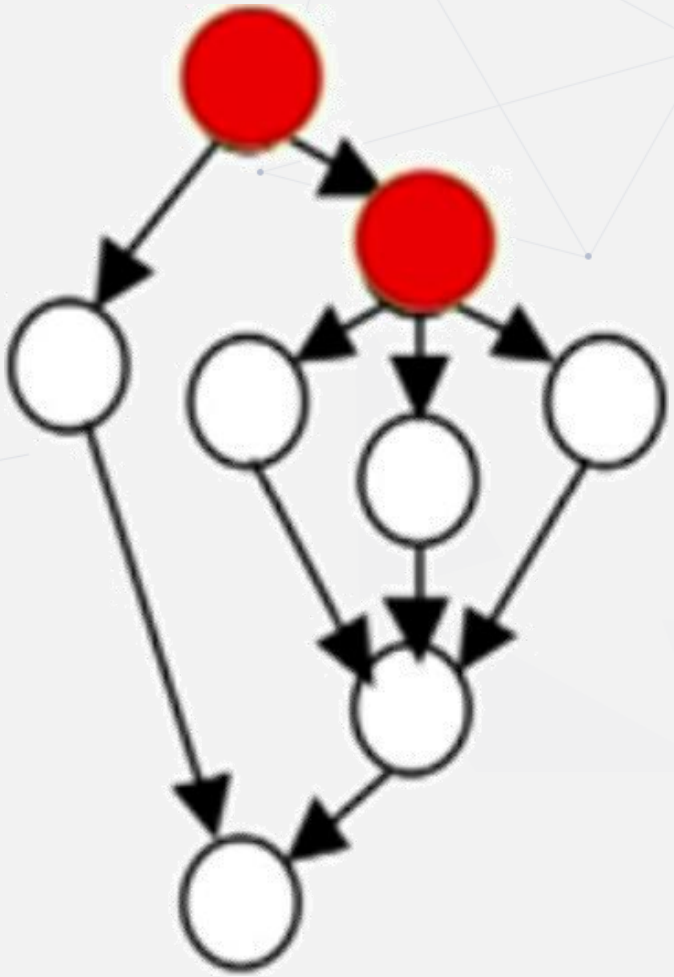
目录

多判定节点串联和存在循环的独立路径测试

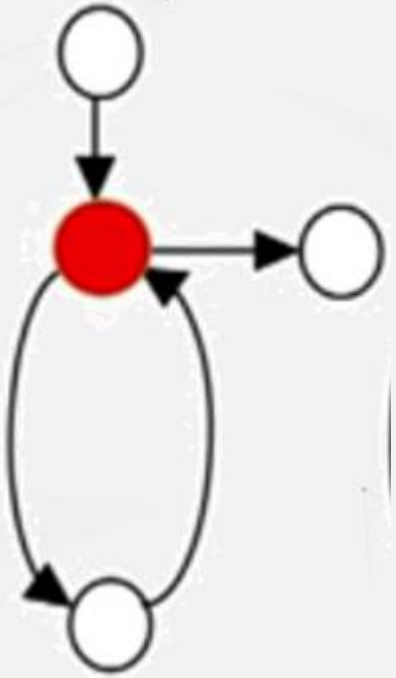
- 存在多个判定节点串联和存在循环的情况：



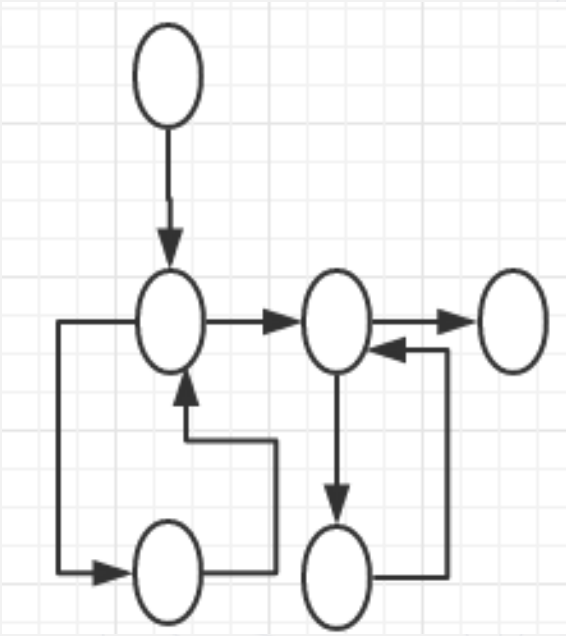
a判定节点串联



b 判定节点嵌套



c 单个循环节点



d 循环节点串联

多判定节点串联和存在循环的独立路径测试

- 如图a所示，假设程序仅包含串联的m（m > 0）个条件判定节点，其中第i个（i∈[1,m]）判定节点具有n_i个分支，路径总数是

path = $\prod_{i=1}^{m} n_i$，独立路径总数V（G） = 1+ $\sum_{i=1}^{m}(n_i-1)$

- 对于条件判定节点的嵌套情况，如b所示，假设程序中仅包含嵌套的m（m > 0）个条件判定节点，其中第i个（i∈[1,m]）判定节点有n_i个分支，则路径总数path及独立路径总数均为V(G) = 1 + $\sum_{i=1}^{m}(ni-1)$

多判定节点串联和存在循环的独立路径测试

- 对于单个循环的情况，如c图所示，若循环次数为n,则完整路径为n+1，环复杂度为2，循环次数越多，基于独立路径的测试优势越明显
- 对于循环节点的串联情况，如d图所示，假设程序中有m个（m>0）串联节点，其中第i个（i∈[1,m]）节点的循环次数为n_i,则路径总数为path = $\prod_{i=1}^{m}(n_i+1)$,独立路径总数为V(G) = m + 1
- 总之，有多判定节点串联和循环节点导致独立路径数量爆增的问题，应基于独立路径进行测试来降低测试用例的规模

目录

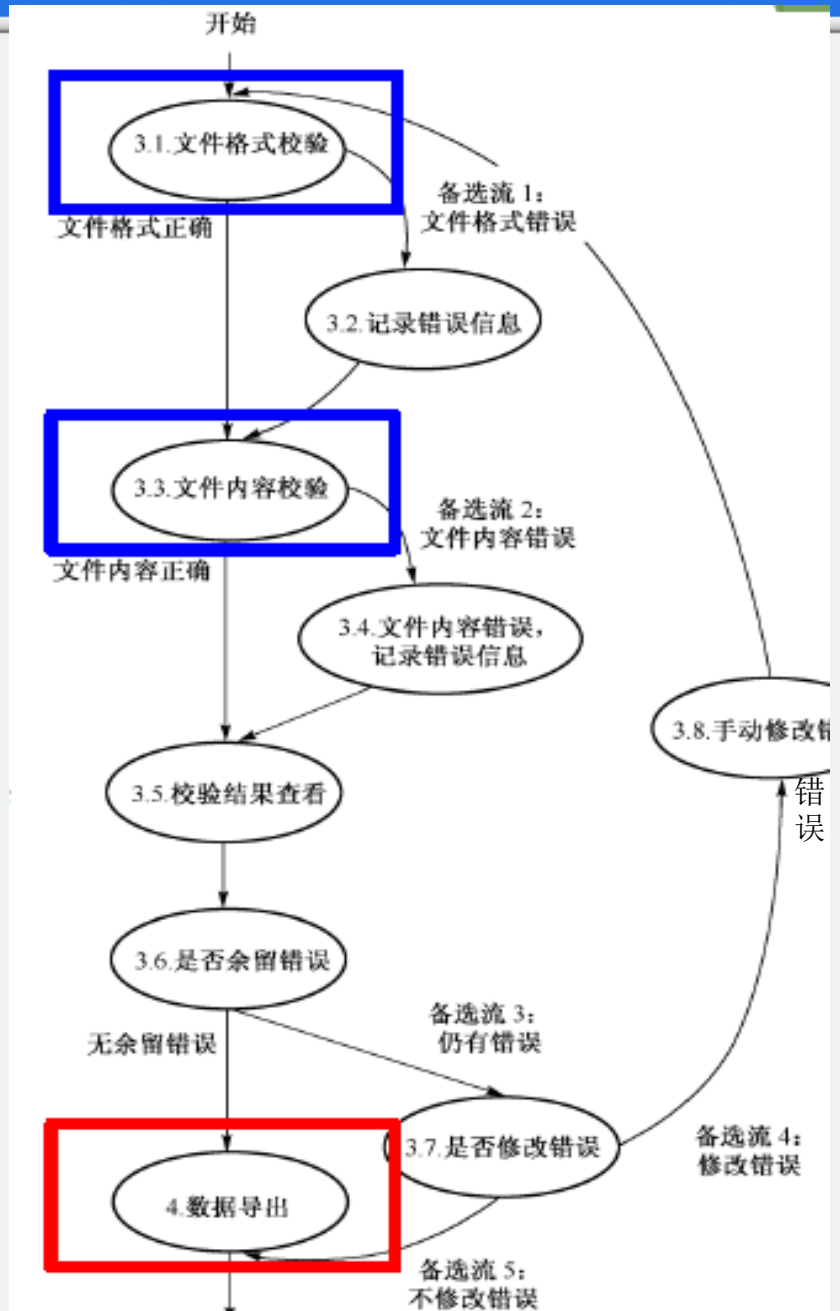
独立路径测试注意事项总结

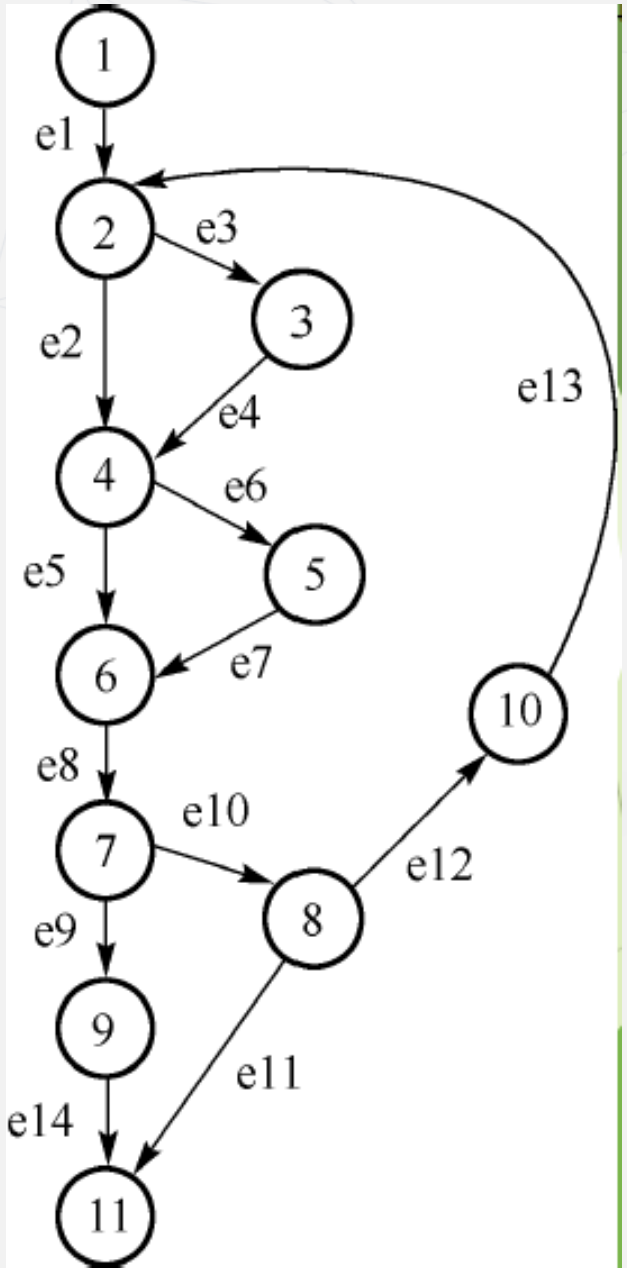
- 避免引入不可行路径
  - 程序设计时应避免不可行路径的引入
- 基于程序图和环复杂度的独立路径测试仅关注结构的测试覆盖
  - 程序图是压缩的程序流图，不考虑串行语句的长度，即忽略了源代码文本复杂性（即代码中数据变量的类型、数量导致 的复杂性）
  - 忽略了循环次数对程序造成的复杂度
  - 忽略了对串行语句的长度，以及每条路径所涉及的针对各数据变量的行为
- 所以，引用独立路径测试，有效降低测试用例数量，提高测试用例有效性，但是也有一定的局限性

路径测试 实践1

- ATM机取款问题的路径测试
  - 根据ATM机取款流程画出程序图，并编号注明编号代表的实际步骤，然后使用路径测试进行测试用例设计
- 做好后，总结独立路径测试与场景法测试的关系

路径测试 实践2





- 信息采集系统

路径测试实践2

Path	包含的边	备注	对应场景	是否可行
Path1	e1,e3,e4,e6,e7,e8,e10,e11	包含所有判定节点	场景6	可行
Path2	e1,e2,e6,e7,e8,e10,e11	在判定节点2处执行e2分支	场景3	可行
Path3	e1,e3,e4,e5,e8,e10,e11	在判定节点4处执行e5分支	场景2	可行
Path4	e1,e3,e4,e6,e7,e8,e9,e14	在判定节点7处执行e9分支	无	不可行
Path5	e1,e3,e4,e6,e7,e8,e10,e12,e13,e3,e4, e6,e7,e8,e10,e11	在判定节点8处执行e12分支	无	可行

基于路径测试的总结

- 适应场景：单元测试阶段，基于独立路径测试主要用于对程序源代码的执行测试，在集成测试阶段，该方法主要用于对业务流程、页面跳转等类似动态执行路径测试
- 使用步骤：

  1 从源代码生成程序图，去掉注释，不包含初始化数据变量声明和串行语句进行压缩

  2 根据程序图计算环复杂度，必要时进行改造

  3 根据环复杂度，以一条最复杂的基础路径为基准，通过覆盖该路径上的每个新的判定分支来抽取一组独立路径集合

  4 分析判定表达式的关联性，去除不可行路径

  5 补充其他高概率或需求（场景）未涵盖到的路径

  6 每条路径生成一条测试用例，对该路径设计合适的数据

基于路径测试的总结

- 基于独立路径测试更强调良好的程序设计，体现在如下方面：

  1 代码设计应尽量简单，保持环复杂度不超过10

  2 代码中应避免重复的判定条件或数据依赖，保持判定节点的独立性，以避免不可行路径

Question