华东交通大学信息工程学院

实验报告册

专　　业　计算机科学与技术

班　　级

姓　　名

学号（用户名）

课程名称　软件测试

教　　师　刘瑞康

学　　期　2025—2026 学年第 一 学期

2025 年　**11**　月　**21**　日

一、实验名称

黑盒测试—边界值分析与等价类划分

二、实验目的

本实验旨在通过对典型逻辑程序（三角形判别系统）的测试实践，提升软件测试的理论认知与工程实践能力：

1. 深入掌握黑盒测试方法，包括边界值分析法与等价类划分法，并理解输入域探索的重要性。
2. 培养严密的测试设计思维，从需求规格说明书中提取测试需求，构建测试模型，以尽可能少的测试用例实现更大覆盖率。
3. 掌握自动化测试流程，包括使用 Python（Pytest 框架）实现自动化测试脚本、断言与测试夹具，以及利用回归测试验证缺陷修复并进行根因分析。

三、实验内容

实验对象为两个功能相同但实现不同的三角形判别程序（triangle_program1.py 与 triangle_program2.py）。两者模拟真实开发场景中常见的逻辑错误。

被测对象功能规范：

- 输入接口：接收三个整数变量 a, b, c。
- 业务逻辑：基于欧几里得几何原理判断三条边是否能构成三角形，并进一步识别等边、等腰或一般三角形。
- 输出定义：返回“等边三角形”“等腰三角形”“一般三角形”“非三角形”或“无效输入”。

实验核心任务：

1. Program 1：应用边界值分析法设计不少于 10 个测试用例，覆盖最小值、最大值及几何临界情况。
2. Program 2：应用等价类划分法设计不少于 8 个测试用例，覆盖所有有效与无效等价类。
3. 工程化实现与分析：构建 Python 测试工程，批量执行并比对测试结果，对缺陷进行逆向分析并提出重构建议。

四、实验仪器与环境

1. 操作系统：`Linux 6.17.8-2-cachyos`
2. 软件开发环境：
    - `Python 3.14.0`

- `Visual Studio Code 1.106.1`
- `pytest==9.0.1`

五、实验原理

黑盒测试基于用户视角，不关注程序内部结构。本实验涉及以下两种核心技术。

1. 边界值分析法

理论基础指出大量逻辑错误出现在输入输出边界区域。常见原因包括对比较运算符的使用失误。

操作原则：识别输入边界，采用"五点法"或"七点法"测试最小值、略大于最小值、正常值、略小于最大值、最大值等典型点。

在本实验中，关键边界包括数值边界（如 1）以及几何边界（如 $a + b = c$）。例如 $(1, 2, 3)$ 比 $(1, 2, 5)$ 更容易暴露错误使用 $\geq$ 的逻辑缺陷。

2. 等价类划分法

理论基础认为输入域可划分为若干互不交叠的子集（等价类），在同一等价类中选择的任意输入具有相同的测试效果。

分类策略：

- 有效等价类：满足需求的合法输入
- 无效等价类：不符合需求的输入，包括负数、零、浮点数、字符等

六、实验过程及实验结果

第一部分：测试用例设计

本阶段通过逻辑分析将需求转化为可执行的测试数据。

1. Program 1 的边界值分析

设计策略：确定边长范围为 $[1, \infty)$，基于单故障假设一次主要变动一个边界值，或关注几何临界点。

边界关注点：

- 数值边界：1
- 几何临界：$a + b = c$

- 逻辑组合：等腰三角形中的不同相等组合

测试用例如下：

测试类别	输入 (a, b, c)	预期输出	设计意图
最小值边界	$(1, 1, 1)$	等边三角形	检查下限处理能力
正常值边界	$(3, 4, 5)$	一般三角形	典型用例
最大值边界	$(100, 100, 100)$	等边三角形	检查大数处理
几何临界值	$(1, 2, 3)$	非三角形	检查是否误用 $\geq$
几何临界邻域	$(2, 3, 4)$	一般三角形	临界附近测试
混合边界	$(1, 1, 2)$	非三角形	检查形状优先级
逻辑组合边界	$(5, 5, 8)$	等腰三角形	覆盖 $a = b$
逻辑组合边界	$(5, 8, 8)$	等腰三角形	覆盖 $b = c$
逻辑组合陷阱	$(5, 8, 5)$	等腰三角形	覆盖 $a = c$
无效数值边界	$(0, 1, 1)$	无效输入	检查零值
无效类型边界	$(-1, 5, 5)$	无效输入	检查负数合法性

2. Program 2 的等价类划分

将输入划分为以下等价类：

- 有效：等边、等腰、一般
- 无效：非三角形、零值、负值、浮点、字符
- 需特别关注特殊非三角形（如 $(1, 1, 3)$）

测试用例如下：

等价类	输入 (a, b, c)	预期输出	设计意图
有效：等边	$(5, 5, 5)$	等边三角形	标准逻辑
有效：等腰	$(5, 5, 3)$	等腰三角形	标准逻辑
有效：一般	$(3, 4, 5)$	一般三角形	标准逻辑
无效：非三角形	$(1, 2, 5)$	非三角形	不满足三角不等式
无效：零值	$(0, 5, 5)$	无效输入	非正整数
无效：负值	$(-1, 5, 5)$	无效输入	非正整数
无效：浮点	$(2.5, 3, 4)$	无效输入	类型校验
无效：字符	$(”a”, 3, 4)$	无效输入	类型校验
缺陷探测专用	$(10, 5, 2)$	非三角形	检查是否仅验证 $a + b > c$
流程探测专用	$(1, 1, 3)$	非三角形	检查流程顺序

第二部分：测试程序实现

使用 pytest 编写自动化测试脚本

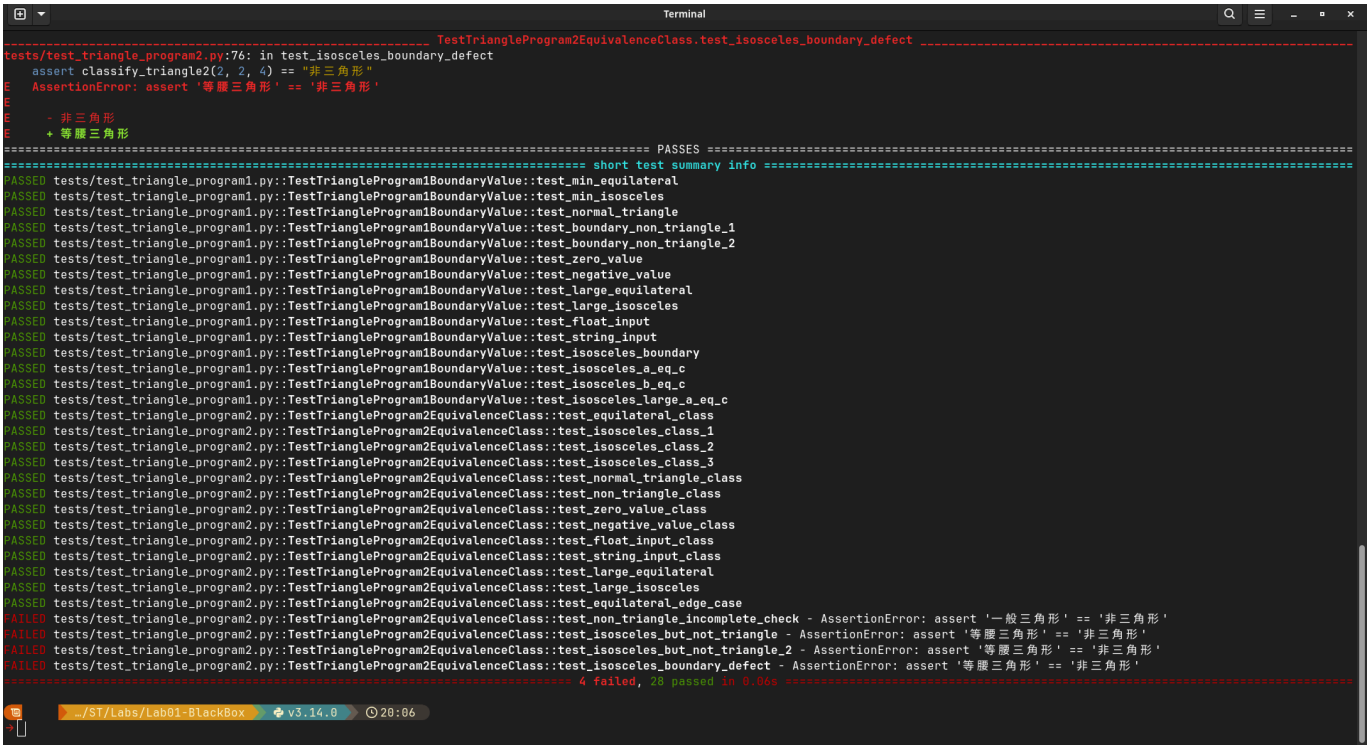
```python
# tests/test_triangle_program1.py
import sys
import os
sys.path.insert(0, os.path.join(os.path.dirname(__file__), '..', 'src'))

from src.triangle_program1 import classify_triangle1
import pytest

class TestTriangleProgram1BoundaryValue:

    # 测试最小正整数等边三角形
    def test_min_equilateral(self):
        assert classify_triangle1(1, 1, 1) == "等边三角形"

    # 测试最小等腰三角形
    def test_min_isosceles(self):
        assert classify_triangle1(2, 2, 1) == "等腰三角形"

    # 测试一般三角形
    def test_normal_triangle(self):
        assert classify_triangle1(3, 4, 5) == "一般三角形"

    # 测试a+b=c边界情况
    def test_boundary_non_triangle_1(self):
        assert classify_triangle1(1, 1, 2) == "非三角形"

    # 测试a+c=b边界情况
    def test_boundary_non_triangle_2(self):
        assert classify_triangle1(2, 5, 3) == "非三角形"

    # 测试零值输入
    def test_zero_value(self):
        assert classify_triangle1(0, 1, 1) == "无效输入"

    # 测试负值输入
    def test_negative_value(self):
        assert classify_triangle1(-1, 2, 3) == "无效输入"

    # 测试大数值等边三角形
    def test_large_equilateral(self):
        assert classify_triangle1(1000, 1000, 1000) == "等边三角形"

    # 测试大数值等腰三角形
    def test_large_isosceles(self):
        assert classify_triangle1(1000, 1000, 1) == "等腰三角形"

    # 测试浮点数输入
    def test_float_input(self):
```

```
    assert classify_triangle1(1.5, 2, 3) == "无效输入"

# 测试字符串输入
def test_string_input(self):
    assert classify_triangle1("a", 1, 1) == "无效输入"

# 测试等腰三角形(a==b)
def test_isosceles_boundary(self):
    assert classify_triangle1(5, 5, 8) == "等腰三角形"

# 测试等腰三角形(a==c)
def test_isosceles_a_eq_c(self):
    assert classify_triangle1(5, 4, 5) == "等腰三角形"

# 测试等腰三角形(b==c)
def test_isosceles_b_eq_c(self):
    assert classify_triangle1(3, 5, 5) == "等腰三角形"

# 测试大数值等腰三角形(a==c)
def test_isosceles_large_a_eq_c(self):
    assert classify_triangle1(10000, 500, 10000) == "等腰三角形"
```

```python
# tests/test_triangle_program2.py
import sys
import os
sys.path.insert(0, os.path.join(os.path.dirname(__file__), '..', 'src'))

from src.triangle_program2 import classify_triangle2
import pytest

class TestTriangleProgram2EquivalenceClass:

    # 测试等边三角形
    def test_equilateral_class(self):
        assert classify_triangle2(5, 5, 5) == "等边三角形"

    # 测试等腰三角形(a==b)
    def test_isosceles_class_1(self):
        assert classify_triangle2(5, 5, 3) == "等腰三角形"

    # 测试等腰三角形(b==c)
    def test_isosceles_class_2(self):
        assert classify_triangle2(3, 5, 5) == "等腰三角形"

    # 测试等腰三角形(a==c)
    def test_isosceles_class_3(self):
        assert classify_triangle2(5, 3, 5) == "等腰三角形"

    # 测试一般三角形
    def test_normal_triangle_class(self):
        assert classify_triangle2(3, 4, 5) == "一般三角形"

    # 测试非三角形
    def test_non_triangle_class(self):
        assert classify_triangle2(1, 2, 5) == "非三角形"

    # 测试三角形不等式验证不完整
    def test_non_triangle_incomplete_check(self):
        assert classify_triangle2(10, 5, 2) == "非三角形"

    # 测试零值输入
    def test_zero_value_class(self):
        assert classify_triangle2(0, 1, 1) == "无效输入"

    # 测试负值输入
    def test_negative_value_class(self):
        assert classify_triangle2(-1, 2, 3) == "无效输入"

    # 测试浮点数输入
    def test_float_input_class(self):
```

```
    assert classify_triangle2(1.5, 2, 3) == "无效输入"

# 测试字符串输入
def test_string_input_class(self):
    assert classify_triangle2("a", 1, 1) == "无效输入"

# 测试超大数值等边
def test_large_equilateral(self):
    assert classify_triangle2(9999, 9999, 9999) == "等边三角形"

# 测试超大数值等腰
def test_large_isosceles(self):
    assert classify_triangle2(9999, 9999, 1) == "等腰三角形"

# 测试等腰但不满足三角形不等式
def test_isosceles_but_not_triangle(self):
    assert classify_triangle2(1, 1, 3) == "非三角形"

# 测试等腰但不满足三角形不等式(另一种情况)
def test_isosceles_but_not_triangle_2(self):
    assert classify_triangle2(2, 5, 2) == "非三角形"

# 测试大数值等边三角形
def test_equilateral_edge_case(self):
    assert classify_triangle2(1000, 1000, 1000) == "等边三角形"

# 测试等腰且a+b=c边界
def test_isosceles_boundary_defect(self):
    assert classify_triangle2(2, 2, 4) == "非三角形"
```

测试程序结果：



第三部分：缺陷分析

1. Program 1：逻辑组合遗漏

缺陷复现：输入 $(5, 8, 5)$ 输出错误。

原因：代码仅检查 $a = b$ 或 $b = c$，遗漏 $a = c$。

修复：

```
elif a == b or b == c or a == c:
    return "等腰三角形"
```

2. Program 2：流程与算法问题

（1）控制流顺序错误 示例输入 $(1, 1, 3)$ 被错误判定为等腰三角形。原因是先判断形状再判断是否构成三角形。 应先验证三角形存在性。

（2）算法逻辑不完备 示例 $(10, 5, 2)$ 未检查所有三角形不等式。

修复：

```
elif (a + b > c) and (a + c > b) and (b + c > a):
    return "一般三角形"
```

七、讨论与结论

1. 故障分析与处理

通过测试，我们定位并分析了以下关键缺陷：

- **Program 1 (逻辑遗漏)**：输入 `(5, 8, 5)` 失败。原因是代码仅检查相邻边相等，遗漏了 `a==c`。需修改条件补全逻辑。
- **Program 2 (算法与流程错误)**：
  - **算法缺陷**：输入 `(10, 5, 2)` 失败。代码仅校验 `a+b>c`，未校验三边完整不等式。需完善为三边两两之和大于第三边。
  - **流程缺陷**：输入 `(1, 1, 3)` 失败。程序先判断"形状"后校验"存在性"。需重构顺序，优先进行合法性校验。

2. 实验反思与心得

- **边界值局限**：物理边界（如最大最小值）易于测试，但逻辑边界（如变量间的关系 `a=c`）更易被遗漏，需特别关注。
- **无效等价类**：需跳出"正常路径"思维，深挖"数值合法但逻辑非法"的隐蔽场景（如非法三角形）。
- **改进建议**：建议在开发中推行测试驱动开发，并结合自动化工具提升代码健壮性。

3. 结论

本实验验证了边界值分析与等价类划分在发现逻辑缺陷与架构漏洞方面的有效性。科学的测试方法结合自动化工具，是弥补开发者自测盲区、保障软件质量的关键手段。