**《软件测试》**

**实验报告五 ——基路径测试方法**

**姓 名： 曹倜源 学 号： 2020112805**

**院 系： 计算机与信息学院 专 业：计算机科学与技术**

**实 验 室： J2301 实验日期： 2023/3/12**

**总评成绩： 审阅教师： 杨青**

### 一、实验目的：

1. 学习逻辑覆盖准则
2. 学习绘制程序图
3. 学习基路径测试
4. 学习测试用例评估方法

### 二、实验环境：

IntelliJ IDEA2023,JUnit

### 三、基路径测试步骤：

1.根据程序代码创建程序图

2.计算圈复杂度

3.获取基路径集

4.构造测试数据，覆盖基路径集

### 四、实验步骤与内容

以下代码是针对JDK8 LinkedList的remove方法实现，完成以下题目要求。具体实现可以参考[OpenJDK](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/LinkedList.java)

1. /\*\*
2. \* Removes the first occurrence of the specified element from this list,
3. \* if it is present. If this list does not contain the element, it is
4. \* unchanged. More formally, removes the element with the lowest index
5. \* {@code i} such that
6. \* <tt>(o==null ? get(i)==null : o.equals(get(i)))</tt>
7. \* (if such an element exists). Returns {@code true} if this list
8. \* contained the specified element (or equivalently, if this list
9. \* changed as a result of the call).
10. \*
11. \* @param o element to be removed from this list, if present
12. \* @return {@code true} if this list contained the specified element
13. \*/
14. public boolean remove(Object o) {
15. 01. if (o == null) {
16. 02. for (Node<E> x = first; x != null; x = x.next) {
17. 03. if (x.item == null) {
18. 04. unlink(x);
19. 05. return true;
20. }
21. }
22. 06. } else {
23. 07. for (Node<E> x = first; x != null; x = x.next) {
24. 08. if (o.equals(x.item)) {
25. 09. unlink(x);
26. 10. return true;
27. }
28. }
29. }
30. 11. return false;
31. }

针对上面的remove方法，设计尽可能少的测试用例，要求达到 100% 语句覆盖率，参考remove方法的文档说明

注：为了便于Jacoco 计算覆盖率，可以把linkedlist源码复制一份到自己工程中的一个包下（如mimic），然后针对该源码测试。

创建remove方法的程序图；

根据创建的程序图，计算圈复杂度

采用基路径测试方法，构造基路径集

根据基路径方法，构造测试数据

使用Junit5框架实现自动化测试，提交到码云仓库

配置插件Pitest，评估测试用例有效性

针对实验中的经验和教训，编写实验总结

按照实验报告模板 编写实验报告，以“学号-姓名-软件测试实验五”命名，提交到雨课堂“软件测试实验五”

**代码实现：**

|  |
| --- |
| **实现代码：**  package cn.edu.ctgu;  import org.junit.jupiter.api.Test;  import java.util.List;  import static org.junit.jupiter.api.Assertions.\*;  class LinkedListTest {  @Test  void testRemoveEmptyList() {  // 初始化空列表  List<Object> list = new LinkedList<>();  // 移除不存在的元素  boolean result = list.remove("element");  // 断言返回结果为false，且列表为空  assertFalse(result);  assertTrue(list.isEmpty());  }  @Test  void testRemoveNonExistingElement() {  // 初始化包含元素的列表  List<String> list = new LinkedList<>();  list.add("element1");  list.add("element2");  list.add("element3");  // 移除不存在的元素  boolean result = list.remove("nonExistingElement");  // 断言返回结果为false，且列表不变  assertFalse(result);  assertEquals(3, list.size());  }  @Test  void testRemoveExistingElement() {  // 初始化包含元素的列表  List<Integer> list = new LinkedList<>();  list.add(1);  list.add(2);  list.add(3);  // 移除存在的元素  boolean result = list.remove(Integer.valueOf(2));  // 断言返回结果为true，且列表中不包含被移除的元素  assertTrue(result);  assertFalse(list.contains(2));  }  @Test  void testRemoveExistingNullElement() {  // 初始化包含null元素的列表  List<String> list = new LinkedList<>();  list.add("element1");  list.add(null);  list.add("element2");  // 移除null元素  boolean result = list.remove(null);  // 断言返回结果为true，且列表中不包含null元素  assertTrue(result);  assertFalse(list.contains(null));  }  @Test  void testRemoveDuplicateElement() {  // 初始化包含重复元素的列表  List<String> list = new LinkedList<>();  list.add("element1");  list.add("element2");  list.add("element1");  list.add("element3");  // 移除第一个出现的重复元素  boolean result = list.remove("element1");  // 断言返回结果为true，且列表中只剩下一个"element1"元素  assertTrue(result);  assertEquals(3, list.size());  assertTrue(list.contains("element1"));  }  }测试结果：    程序图： |
| **圈复杂度：2**  **基路径：**  基路径 1: o == null 分支（包括循环内和循环外的路径） o 为 null 遍历链表中所有节点，找到匹配的节点 未找到匹配的节点 基路径 2: o != null 分支（包括循环内和循环外的路径） o 不为 null 遍历链表中所有节点，找到匹配的节点 未找到匹配的节点 |

### 五、结论分析与体会

通过本次实验，我对逻辑覆盖准则、绘制程序图、基路径测试和测试用例评估方法有了更深入的了解。我认识到在软件测试过程中，合理选择适当的逻辑覆盖准则、绘制程序图并进行基路径测试，可以帮助我们设计高效的测试用例，从而提高测试的效果。同时，合理的测试用例评估方法可以帮助我们评估测试用例的质量和有效性，从而优化测试策略和提高测试效率。

在实验中，我学会了如何根据实际需求选择适合的逻辑覆盖准则，并通过绘制程序图和进行基路径测试，设计了测试用例。同时，我也学习了如何使用不同的测试用例评估方法，如覆盖率评估、等效类划分、边界值分析等，来评估测试用例的质量和充分性。

然而，我也注意到在实际应用中，选择适当的逻辑覆盖准则和进行基路径测试时可能会面临一定的挑战和复杂性。不同的程序结构和复杂度可能需要不同的测试策略和方法。同时，测试用例评估方法也需要根据实际情况进行选择和调整。因此，在进行软件测试时，需要综合考虑测试目标、测试需求、程序结构和复杂度等因素，合理选择适合的测试方法和评估方法。

总的来说，本次实验让我对软件测试中的逻辑覆盖准则、程序图、基路径测试和测试用例评估方法有了更深入的认识，并通过实际操作获得了实际经验。这对我的软件测试能力提升和实际工作应用具有积极的意义。

### 六、仓库地址

https://github.com/Ctyring/work5.git