****

**“软件质量保证与测试”课程**

**作业（实验）报告**

组长姓名 朱盛 学号\_\_\_16201532 \_

成员姓名 朱天保 学号 16201533

成员姓名 王文涛 学号 16201534

成员姓名 王慎瑜 学号 16201535

专 业 \_\_\_软件工程 班级 162015

作业名称 “软件质量保证与测试”课程测试报告

指导教师 段喜龙

开课学期 2018 至 2019 学年 1 学期

完成时间 2019 年 1 月 4 日

**目 录**

[1 前言 1](#_Toc15848)

[2 软件测试概述 2](#_Toc23155)

[2.1 软件测试的概念 2](#_Toc18776)

[2.2 测试的目的 2](#_Toc136)

[2.3 测试的原则 2](#_Toc6924)

[2.4 黑盒测试概述 2](#_Toc26681)

2.5 白盒测试概述 3

2.6 场景测试概述 4

[3 电梯结构简介 6](#_Toc21432)

[3.1电梯定义 6](#_Toc534289598)

[3.2 电梯的简单使用方法（不考虑紧急情况和复杂情况）： 6](#_Toc534289599)

[3.3 电梯系统基本描述 6](#_Toc534289600)

[3.4 电梯基本运行规则 7](#_Toc534289601)

[4 制定测试计划 9](#_Toc23703)

[4.1 测试内容 9](#_Toc21141)

[4.1.1 黑盒测试 9](#_Toc2084)

[4.1.2 白盒测试 1](#_Toc20972)0

[4.1.3 场景测试 1](#_Toc2084)1

[4.2 测试环境 1](#_Toc9920)1

[4.3 测试方法 1](#_Toc4098)2

[4.4 测试通过标准 1](#_Toc19042)1

[5 测试用例的设计与实施 1](#_Toc637)2

[5.1 黑盒测试 1](#_Toc1896)2

[5.1.1 边界值测试 1](#_Toc2084)2

[5.1.2 等价类测试 1](#_Toc20972)3

[5.1.3 决策表测试 1](#_Toc2084)7

[5.2 白盒测试 2](#_Toc27918)2

[5.3 场景测试 6](#_Toc22326)0

[5.3.1 场景分析 6](#_Toc2084)0

[5.3.2 设计测试用例 6](#_Toc20972)0

[5.3.3 缺陷截图 6](#_Toc2084)2

[5.4 小组分工 6](#_Toc22326)2

[6 缺陷管理与分析 63](#_Toc24556)

[6.1 缺陷概述 6](#_Toc6852)3

[6.2 缺陷内容 6](#_Toc24070)3

[6.3 缺陷分析 6](#_Toc10125)4

[6.4 总结 6](#_Toc1282)4

[结论与展望 6](#_Toc8882)5

# 1 前言

随着计算机技术的日益普及和软件开发水平的不断提高以及人们对软件测试重要性的认识越来越深刻，软件测试阶段在整个软件开发周期中所占的比重日益增大。

软件测试是产品最终交付到用户之前的最后一道防线，有着举足轻重的地位。软件测试的效果直接关系到软件产品的质量。软件测试所追求的是通过系统的测试方法，发现软件中的错误，提供丰富的错误诊断信息，便于改正错误，以达到预防错误的发生，减少软件开发费用的目标。软件测试就显得尤为重要。下面主要通过黑盒测试，白盒测试以及场景测试方面对软件进行测试。

黑盒测试又被称为功能测试、数据驱动测试或基于规格说明的测试，是通过使用整个软件或某种软件的功能来严格的测试，而并没有通过检查程序的源代码来测试。白盒测试则主要是通过程序的源代码进行测试而不使用用户界面。这种类型的测试需要从代码中发现在算法、溢出、路径、条件等等中的缺点或者错误，进而加以修正。场景测试是通过运用场景来对系统的功能点或业务流程的描述，从而提高测试的效果。

随着社会的进步及人们对物质生活的需要自动化的东西渐渐的进入到每个的生活，还有一个原因就是我国老龄化越来越严重，这类人更需要电梯的存在。电梯运行软件保障电梯无障碍运行。

大量的体育馆、地铁、机场、商场、写字楼、宾馆、住宅小区等好多都加装了电梯。方便人们出行。电梯的出现相对应的电梯执行软件就蓄势待发一个一个升起。这个软件平台系统提高了电梯的运行效率，降低了企业运营的成本和风险，让企业有更好的发展。本次测试将对电梯运行的各个模块进行测试，重点测试其中的捎带功能模块。

# 2 软件测试概述

## 2.1 软件测试的概念

软件测试是指在规定条件下对程序进行操作，以发现错误，对软件质量进行评估。目前,国际上已对软件测试形成一个共识[[1]](#footnote-0):软件测试就是在软件投入使用前，对软件的需求分析、设计规格说明和系统编码的最终复审。由此可见,软件测试的对象不仅仅是程序，需求规格说明、概要设计、详细设计、源程序等都是软件测试的对象。首先，测试不能表明软件中不存在错误，它只能说明软件中存在错误。从用户的角度来看，希望通过测试发现错误和缺陷，从而考虑是否可接受该产品。而从软件开发者的角度来看，则希望通过测试，表明软件产品中不存在错误。换句话说，测试的目的是以最少的时间和人力，系统地找出软件中潜在的各种错误和缺陷。

## 2.2 测试的目的

基于不同的立场，也会有不同的测试目的，从用户角度出发，普遍希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可接受该产品但从软件开发者的角度出发，则希望测试能成为表明软件产品中不存在错误的过程，验证该产品已确实地实现了用户的要求，确立人们对软件质量的信心。

## 2.3 测试的原则

基于测试是为了寻找软件的错误和缺陷，评估与提高软件质量，我们在测试的时候遵循以下原则[[2]](#footnote-1)：

(1)所有的软件测试都应追溯到用户需求。

(2)应当把“尽早地不断地进行软件测试”作为软件测试者的座右铭。

(3)完全测试是不可能的，测试需要终止。

(4)测试无法显示软件潜在的缺陷。

(5)充分注意测试中的群集现象。

(6)程序员应避免检查自己的程序。

(7)尽量避免测试的随意性。

## 2.4 黑盒测试概述

从理论上讲，黑盒测试只有采用穷举输入测试，把所有可能的输入都作为测试情况考虑，才能查出[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但可能的输入进行测试。这样看来，完全测试是不可能的，所以我们要进行有针对性的测试，通过制定测试案例指导测试的实施，保证[软件测试](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)有组织、按步骤，以及有计划地进行。黑盒测试行为必须能够加以量化，才能真正保证[软件质量](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E8%B4%A8%E9%87%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)，而[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)就是将测试行为具体量化的方法之一。具体的黑盒[测试用例设计](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B%E8%AE%BE%E8%AE%A1" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)方法包括等价类划分法、边界值分析法、错误推测法、[因果图法](https://baike.baidu.com/item/%E5%9B%A0%E6%9E%9C%E5%9B%BE%E6%B3%95" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、判定[表驱动](https://baike.baidu.com/item/%E8%A1%A8%E9%A9%B1%E5%8A%A8" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)法、正交试验设计法、功能图法、[场景](https://baike.baidu.com/item/%E5%9C%BA%E6%99%AF" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)法等。

首先，根据用户需求报告中关于功能要求和性能指标的规格说明书，定义相应的测试需求报告，即制订黑盒测试的最高标准，以后所有的测试工作都将围绕着测试需求来进行，符合测试需求的[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)即是合格的，反之即是不合格的；同时，还要适当选择测试内容，合理安排测试人员、测试时间及测试资源等。将[测试计划](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E8%AE%A1%E5%88%92" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)阶段制订的测试需求分解、细化为若干个可执行的[测试过程](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E8%BF%87%E7%A8%8B" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)，并为每个测试过程选择适当的[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)（测试用例选择的好坏将直接影响到测试结果的有效性）。执行测试开发阶段建立的自动[测试过程](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E8%BF%87%E7%A8%8B" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)，并对所发现的[缺陷](https://baike.baidu.com/item/%E7%BC%BA%E9%99%B7" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)进行[跟踪管理](https://baike.baidu.com/item/%E8%B7%9F%E8%B8%AA%E7%AE%A1%E7%90%86" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)。测试执行一般由[单元测试](https://baike.baidu.com/item/%E5%8D%95%E5%85%83%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、组合测试、[集成测试](https://baike.baidu.com/item/%E9%9B%86%E6%88%90%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、系统联调及[回归测试](https://baike.baidu.com/item/%E5%9B%9E%E5%BD%92%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)等步骤组成，测试人员应本着科学负责的态度，[一步一个脚印](https://baike.baidu.com/item/%E4%B8%80%E6%AD%A5%E4%B8%80%E4%B8%AA%E8%84%9A%E5%8D%B0" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)地进行测试。

 黑盒测试的优点是基本上不用人管着，如果[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)停止运行了一般就是被[测试程序](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)crash了；设计完[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)之后，下来的工作就是爽了，当然更苦闷的是确定crash原因。缺点是结果取决于[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)的设计，测试用例的设计部分优势来源于经验，OUSPG的东西很值得借鉴；没有状态转换的概念，一些成功的例子基本上都是针对PDU来做的，还做不到针对被[测试程序](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)的状态转换来实现；就没有状态概念的测试来说，寻找和确定造成[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/%E9%BB%91%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)crash的测试例是个麻烦事情，必须把周围可能的测试例单独确认一遍。而就有状态的测试来说，就更麻烦了，尤其不是一个单独的testcase造成的问题。这些在堆的问题中表现的更为突出。

## 2.5 白盒测试概述

白盒测试又称结构测试、透明盒测试、逻辑驱动测试或基于[代码](https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)的测试。白盒测试是一种[测试用例设计](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B%E8%AE%BE%E8%AE%A1/9391582" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)方法，盒子指的是被测试的[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6/12053" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)，白盒指的是盒子是可视的，你清楚盒子内部的东西以及里面是如何运作的。"白盒"法全面了解程序内部逻辑结构、对所有逻辑路径进行测试。"白盒"法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数是天文数字。常用的[软件测试](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)方法有两大类：[静态测试](https://baike.baidu.com/item/%E9%9D%99%E6%80%81%E6%B5%8B%E8%AF%95/1916468" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)方法和[动态测试](https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E6%B5%8B%E8%AF%95/1916633" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)方法。其中软件的静态测试不要求在计算机上实际执行所测程序，主要以一些人工的模拟技术对软件进行分析和测试；而软件的动态测试是通过输入一组预先按照一定的测试准则构造的实例数据来动态运行程序，而达到发现程序错误的过程。在动态分析技术中,最重要的技术是路径和分支测试。

白盒测试的测试方法有代码检查法、静态结构分析法、静态质量度量法、[逻辑覆盖](https://baike.baidu.com/item/%E9%80%BB%E8%BE%91%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)法、基本[路径测试](https://baike.baidu.com/item/%E8%B7%AF%E5%BE%84%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)法、[域测试](https://baike.baidu.com/item/%E5%9F%9F%E6%B5%8B%E8%AF%95" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、符号测试、[路径覆盖](https://baike.baidu.com/item/%E8%B7%AF%E5%BE%84%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)和程序变异。

白盒测试法的覆盖标准有[逻辑覆盖](https://baike.baidu.com/item/%E9%80%BB%E8%BE%91%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、循环覆盖和基本路径测试。其中逻辑覆盖包括[语句覆盖](https://baike.baidu.com/item/%E8%AF%AD%E5%8F%A5%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、[判定覆盖](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、[条件覆盖](https://baike.baidu.com/item/%E6%9D%A1%E4%BB%B6%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)、判定/条件覆盖、[条件组合覆盖](https://baike.baidu.com/item/%E6%9D%A1%E4%BB%B6%E7%BB%84%E5%90%88%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)和[路径覆盖](https://baike.baidu.com/item/%E8%B7%AF%E5%BE%84%E8%A6%86%E7%9B%96" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)。六种覆盖标准发现错误的能力呈由弱到强的变化：

1.语句覆盖每条语句至少执行一次。

2.判定覆盖每个判定的每个分支至少执行一次。

3.条件覆盖每个判定的每个条件应取到各种可能的值。

4.判定/条件覆盖同时满足判定覆盖条件覆盖。

5.条件组合覆盖每个判定中各条件的每一种组合至少出现一次。

6.路径覆盖使程序中每一条可能的路径至少执行一次。

白盒测试中要求保证一个模块中的所有独立路径至少被使用一次；对所有逻辑值均需测试 true 和 false；在上下边界及可操作范围内运行所有循环；检查内部数据结构以确保其有效性。

白盒测试的实施步骤：.[测试计划](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E8%AE%A1%E5%88%92" \t "https://baike.baidu.com/item/%E7%99%BD%E7%9B%92%E6%B5%8B%E8%AF%95/_blank)阶段：根据需求说明书，制定测试进度；测试设计阶段：依据程序设计说明书，按照一定规范化的方法进行软件结构划分和设计测试用例；测试执行阶段：输入测试用例，得到测试结果；测试总结阶段：对比测试的结果和代码的预期结果，分析错误原因，找到并解决错误。

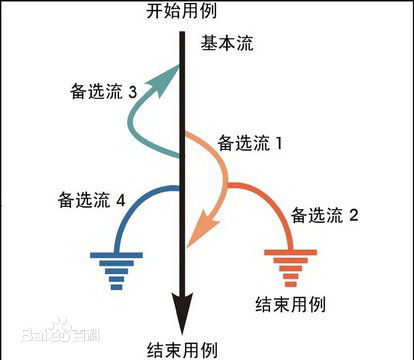
白盒测试的优点是迫使测试人员去仔细思考软件的实现；可以检测代码中的每条分支和路径；揭示隐藏在代码中的错误；对代码的测试比较彻底；最优化。缺点为测试昂贵；无法检测代码中遗漏的路径和数据敏感性错误；不验证规格的正确性。

## 2.6 场景测试概述

通过运用场景来对系统的功能点或业务流程的描述，从而提高测试效果的一种方法。[用例](https://baike.baidu.com/item/%E7%94%A8%E4%BE%8B/163511" \t "https://baike.baidu.com/item/%E5%9C%BA%E6%99%AF%E6%B3%95/_blank)场景来测试需求是指模拟特定场景边界发生的事情，通过事件来触发某个动作的发生，观察事件的最终结果，从而用来发现需求中存在的问题。我们通常以正常的用例场景分析开始，然后再着手其他的场景分析。场景法一般包含基本流和备用流，从一个流程开始，通过描述经过的路径来确定的过程，经过遍历所有的基本流和备用流来完成整个场景。场景主要包括4种主要的类型：正常的用例场景，备选的用例场景，异常的用例场景，假定推测的场景。

基本流：采用直黑线表示，是经过用例的最简单的路径（无任何差错，程序从开始直接执行到结束）

备选流：采用不同颜色表示，一个备选流可能从基本流开始，在某个特定条件下执行，然后重新加入基本流中，也可以起源于另一个备选流，或终止用例，不在加入到基本流中（各种错误情况）。



设计步骤：

1. 根据说明，描述出程序的基本流及各项备选流。

2. 根据基本流和各项备选流生成不同的场景。

3. 对每一个场景生成相应的[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B/1928697" \t "https://baike.baidu.com/item/%E5%9C%BA%E6%99%AF%E6%B3%95/_blank)。

4. 对生成的所有测试用例重新复审，去掉多余的测试用例，测试用例确定后，对每一个测试用例确定测试数据值。

**3 电梯结构简介**

## 3.1电梯定义

电梯是服务于规定楼层的固定式升降设备。垂直升降电梯具有一个轿厢，运行在至少两列垂直的或倾斜角小于15°的刚性导轨之间。台阶式自动扶梯和滚梯不在本次系统考虑范围之内。

## 3.2 电梯的简单使用方法（不考虑紧急情况和复杂情况）：

载人电梯都是微机控制的智能化、自动化设备，不需要专门的人员来操作驾驶，普通乘客只要按下列程序乘坐和操作电梯即可。

i) 在乘梯楼层电梯入口处，根据自己上行或下行的需要，按向上或向下的箭头按钮，只要按钮上的灯亮，就说明你的呼叫已被记录，只要等待电梯到来即可。

ii) 电梯到达目标楼层停下开门后，先让轿厢内人员走出电梯，然后呼梯者再进入电梯轿厢。进入轿厢后，根据你需要到达的楼层，按下轿厢内操纵盘上相应的数字按钮。同样，只要该按钮灯亮，则说明你的目标楼层已被记录；此时不用进行其他任何操作（在到达目标楼层之前，反复按该目标楼层的按钮为无效操作），一旦电梯门关闭之后，电梯就会开始运动，只要等电梯到达你的目标层停靠开门即可。

iii) 电梯行驶到你的目标层后会自动开门，此时按顺序走出电梯即结束了一次乘梯过程。

## 3.3 电梯系统基本描述

1) 本系统的电梯为一个运行在10层楼的电梯，电梯可在每层经停。楼层计数采用中国式楼层计数，即1层显示为1，2层为2，依次类推，直至顶层显示为10。

2) 每个楼层都有一组电梯运行请求按钮（下面简称为楼层按钮），即上行请求按钮和下行请求按钮。如果乘客按上行请求按钮时，则表示想搭乘电梯去往上面的楼层；反之，则表示想搭乘电梯去往下面的楼层。规定最底层（1层）只有上行请求按钮，最顶层（10层）只有下行请求按钮，中间楼层同时有上行和下行两个方向请求按钮。

3) 一个电梯箱体（剩余文档简称电梯）内有对应每个楼层的目的地请求按钮（下面简称目标楼层），一般显示相应楼层的层数。

## 3.4 电梯基本运行规则

本次系统的电梯控制系统具备“捎带”功能，即电梯在去往目标楼层过程中，如果在电梯前进方向上有新的同向乘梯请求，控制系统需要判断是否响应该请求。

1)程序运行开始或重置时设置电梯停靠在一层；

2)电梯的状态定义：

a)电梯运行状态：从电梯启动时刻,到电梯运动停止时的运行状态,包括上行（UP）和下行（DOWN）状态。此状态电梯速度大于0。（启动时刻，运行停止时刻）。

b)电梯开关门状态：电梯静止时，从门打开时刻（门开始动作），到门完全关闭时刻（门刚刚停止动作）时的状态。此状态电梯速度等于0。[门打开时刻，门关闭时刻]。

c)电梯停留状态（WFS状态，WaitforService）：电梯停在某层，且门长时间处于关闭状态。此状态电梯速度等于0，且请求队列为空。（门关闭时刻，门准备打开时刻或电梯准备启动时刻]。

d)STILL状态：此时电梯处于运行速度为0的状态。[电梯运行停止时刻，电梯准备启动时刻]。（注意STILL状态仅用于输出表达）

3)一个楼层同一时刻只能发出一个上行或下行请求。电梯未到本楼层的时候，某个请求按钮被按下变亮后，再按不会产生实际效果（即不被响应）。但是发出上行请求后可以再发出下行请求，反之亦可，这视为两个不同的请求，执行完一个后另一个仍需执行。在电梯到达某楼层处于开关门状态时，该楼层的多个同向请求只认为是一个请求。当电梯关门动作完成后（含门完全静止的那一刻），可以再产生新的上下行请求；

4)电梯内的一个目标楼层按钮被按下后只能发出前往某个目标楼层的请求，一旦发出某个目标楼层请求后，在电梯到达该楼层并完成关门动作前（包括关门完毕时刻），目标楼层与该按钮楼层相同的多个电梯内请求被认为是一个请求。当电梯关门结束后，可以再发出任意目标楼层请求。

5)所有请求按照时间上的先来先服务策略（FirstArrivedFirstServed，FAFS）作为基本调度原则，具体含义是，在没有其它策略时，按照FAFS来响应。

6)如果电梯同时收到了电梯内请求和楼层请求时，则按照输入时的请求排列顺序执行。

7)本次系统的电梯系统采用ALS\_Schedule(ALittleSmartSchedule)调度策略：

(1)只要队列不为空，每次都取出队列头请求来调度（同傻瓜调度策略）；

(2)电梯在运动过程中不能突然改变运动方向；

(3)在调度电梯完成一个（或一组执行时间有重叠）请求的过程中，电梯要响应所有满足“顺路捎带”条件的请求。

# 4 制定测试计划

## 4.1 测试内容

在电梯程序中，主要从以下几个方面进行：黑盒测试，白盒测试和场景测试。

**4.1.1 黑盒测试**

黑盒测试是为了检验系统是否满足功能需求说明中的功能需求。根据功能性测试用例，逐项检查，检查产品是否达到用户要求功能。对于软件的功能性测试主要针对每种功能设计若干典型测试用例，软件测试过程中运行测试用例，然后将得到的结果与已知标准答案进行比较，我们工作中的软件产品登记测试，主要是对软件进行的功能性测试。功能性测试的主要方法有;

(1)等价类划分测试

等价类划分是一种典型的黑盒测试方法，该方法完全不考虑程序的内部结构，只根据对软件的要求和说明，即需求规格说明，把程序输入域划分成若干个部分，然后从每个部分中选取少数有代表性的数据作为测试输入。使用等价类划分方法设计测试用例，必须在分析需求规格说明的基础上划分等价类。

(2)边界值测试

使用边界值分析方法设计测试用例，首先应确定边界情况。输入等价类与输出等价类的边界，就是着重测试的边界情况。

边界值分析方法的基本思想是：选取正好等于，刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中典型值或任意值作为测试数据。

边界值分析方法是一种最有效的黑盒测试方法，但当边界情况和复杂的时候，要找出适当的边界测试用例还需针对问题的输入域、输出域边界，耐心仔细的逐个进行考虑。

(3)决策表测试

决策表通常由4部分组成：条件桩、条件项、动作桩和动作项。

条件桩分列出问题的所有条件，除了某些问题对条件的先后次序有特定的要求外，通常在这里列出的条件其先后次序无关紧要。条件项部分对条件桩给出的条件列出所有可能的取值。动作桩则给出了问题规定的可能采取的操作，这些操作的排列顺序一般没有什么约束。动作项指出在条件项的各组取值情况下应采取的动作。

把任何一个条件组合的特定取值及相应要执行的动作称为一条规则，在决策表中贯穿条件项和动作项的一列就是一条规则。显然，决策表中列出多少组条件取值，就有多少条规则。

(4)因果图测试

因果图方法能够帮助测试人员按照一定的步骤，高效率地开发测试用例，以检测程序输入条件的各种组合情况。它是将自然语言规格说明转化成形式语言规格说明的一种严格的方法，可以指出规格说明中存在的不完整性和二义性。

因果图中使用了简单的逻辑符号，以直线连接左右结点。左结点表示输入状态，右结点表示输出状态。因果图用4种符号分别表示规格说明中的4种因果关系。

在实际问题中，输入状态相互之间还可能存在某些依赖关系，称之为：“约束”。比如，某些输入条件本身不可能同时出现。输出状态之间也往往存在约束。

**4.1.2 白盒测试**

白盒测试又称透明盒测试、逻辑驱动测试或基于代码的测试。白盒测试是一种测试用例设计方法，盒子指的是被测试的软件， 白盒指的是盒子是可视的，你清楚盒子内部的东西以及里面是如何运作的。"白盒"法全面了解程序内部逻辑结构、对所有逻辑路径进行测试。"白盒"法是穷举路 径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数是天文数字。

采用什么方法对软件进行测试呢？常用的软件测试方法有两大类：静态测试方法和动态测试方 法。其中软件的静态测试不要求在计算机上实际执行所测程序，主要以一些人工的模拟技术对软件进行分析和测试；而软件的动态测试是通过输入一组预先按照一定 的测试准则构造的实例数据来动态运行程序，而达到发现程序错误的过程。在动态分析技术中,最重要的技术是路径和分支测试。下面要介绍的六种覆盖测试方法属 于动态分析方法。

白盒测试的测试方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、域测试、符号测试、路径覆盖和程序变异。

白盒测试法的覆盖标准有逻辑覆盖、循环覆盖和基本路径测试。其中逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。六种覆盖标准发现错误的能力呈由弱到强的变化：

1.语句覆盖每条语句至少执行一次。

2.判定覆盖每个判定的每个分支至少执行一次。

3.条件覆盖每个判定的每个条件应取到各种可能的值。

4.判定/条件覆盖同时满足判定覆盖条件覆盖。

5.条件组合覆盖每个判定中各条件的每一种组合至少出现一次。

6.路径覆盖使程序中每一条可能的路径至少执行一次。

**4.1.3 场景测试**

我们可以理解为功能测试中的用例，即性能测试的场景就是性能测试的用例。性能测试的场景是为了要实现特定的测试目标而对应用执行的压测活动。性能测试场景的设计与执行是整个性能测试活动的核心与灵魂，没有完整的场景设计就无法达到我们的测试目的，没有合理的场景设计就不会发现系统的性能缺陷。我们所开发的测试脚本，所预埋的测试数据都是为了实现特定场景所准备的。

核心思想是把自己当成最终的用户，使用软件，设计出在使用软件过程中重要的操作，一般包括两类：

模拟用户完成正常功能、核心业务逻辑的动作，以验证功能的正确性

模拟用户操作中出现的主要错误，以验证程序的异常处理能力

## 4.2 测试环境

(1)硬件环境：中央处理器:Inter(R) Core(TM) i5-6200U CPU @ 2.30 GHz 2.40 GHz

内存:8G

(2)软件环境：操作系统Micosoft Windows 10

编译器器 eclipse 4.5.2

## 4.3 测试方法

采用黑盒测试、白盒测试、场景测试设计测试用例。

测试工具：eclipse

## 4.4 测试通过标准

(1) 如果测试结果与测试用例中预期结果一致测试通过否则不通过。

(2) 数据安全、正确，测试通过，否则测试不通过。

(3) 系统崩溃，无法测试，测试不通过。

**5 测试用例的设计与实施**

## 5.1 黑盒测试

**5.1.1 边界值测试**

(1) 边界值分析：

标识符∈{FR,ER},m∈[1,10],方向∈{UP,DOWN},n∈[0,10]

min,min+,nom,max-,max5种组合测试

表5.1 边界值测试用例

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | **输入** | **预期输出** | **实际输出** | **测试结果** | **缺陷优先级** | **缺陷优先程度** | **备注** |
|  | (FR,1,UP,0)  (ER,1,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0) | Passed |  |  |  |
|  | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | Passed |  |  |  |
|  | (FR,1,UP,0)  (ER,5,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,5,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,5,1]/(5,UP,3.0) | Passed |  |  |  |
|  | (FR,1,UP,0)  (ER,9,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,9,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,9,1]/(9,UP,5.0) | Passed |  |  |  |
|  | (FR,1,UP,0)  (ER,10,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,10,1]/(10,UP,5.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,10,1]/(10,UP,5.5) | Passed |  |  |  |
|  | (FR,1,UP,0)  (FR,1,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  SAME[FR,1,UP,1] | [FR,1,UP,0]/(1,STILL,1.0)  SAME[FR,1,UP,1] | Passed |  |  |  |
|  | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | Passed |  |  |  |
|  | (FR,1,UP,0)  (FR,5,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,5,UP,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,5,UP,1]/(5,UP,3.0) | Passed |  |  |  |
|  | (FR,1,UP,0)  (FR,9,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,1]/(9,UP,5.0) | Passed |  |  |  |
|  | (FR,1,UP,0)  (ER,10,UP,1)  RUN | INVALID[ER,10,UP,1]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[ER,10,UP,1]  [FR,1,UP,0]/(1,STILL,1.0) | Passed |  |  |  |

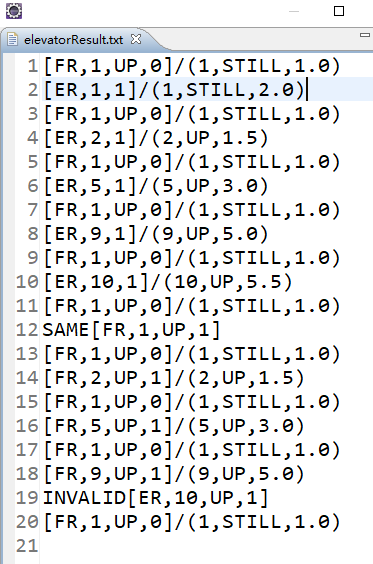


图5.1.1 边界值测试截图

**5.1.2 等价类测试**

(1)等价类分析;

程序的输入为楼层请求(FR,m,UP/DOWN,T)，电梯内请求(ER,n,T)；标识符是FR或者ER；m∈[1,10]；上下行为UP或DOWN；T0<=T1（前一次请求的时刻不大于后一次请求）；n∈[1,10]

得出等价类：

1. FR 2. ER 3. m∈[1,10] 4. 0<=T0<=T1 5. UP 6. DOWN 7. n∈[1,10] 8.m<1 9.m>10 10.T0<0 11.T0>T1 12.标识符不是ER或者FR 13.上下行不是UP或DOWN 14.n<1 15.n>10

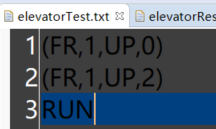
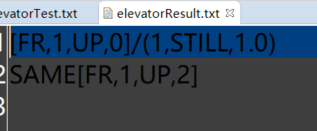
（2）电梯外的请求等价类生成测试用例如下：

等价类测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 | 备注 |
| 1 | (FR,1,UP,0) (FR,11,UP,0) RUN | INVALID[FR,11,UP,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,11,UP,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |
| 2 | (FR,1,UP,0) (FR,11,DOWN,1) RUN | INVALID[FR,11,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,11,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |
| 3 | (FR,1,UP,0) (FR,10,UP,1) RUN | INVALID[FR,10,UP,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,10,UP,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |
| 4 | (FR,1,UP,0) (FR,10,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,10,DOWN,1]/(10,UP,5.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,10,DOWN,1]/(10,UP,5.5) | Passed | - | - | - |
| 5 | (FR,1,UP,0) (FR,9,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,1]/(9,UP,5.0) | Passed | - | - | - |
| 6 | (FR,1,UP,0) (FR,9,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,DOWN,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,DOWN,1]/(9,UP,5.0) | Passed | - | - | - |
| 7 | (FR,1,UP,0) (FR,5,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,UP,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,UP,1]/(5,UP,3.0) | Passed | - | - | - |
| 8 | (FR,1,UP,0) (FR,5,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,DOWN,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,DOWN,1]/(5,UP,3.0) | Passed | - | - | - |
| 9 | (FR,1,UP,0) (FR,2,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,UP,1]/(2,UP,1.5) | Passed | - | - | - |
| 10 | (FR,1,UP,0) (FR,2,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,DOWN,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,DOWN,1]/(2,UP,1.5) | Passed | - | - | - |
| 11 | (FR,1,UP,0) (FR,1,UP,2) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,1,UP,2]/(1,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) SAME[FR,1,UP,2] | Failed | 5 | 5 | 截图1 |
| 12 | (FR,1,UP,0) (FR,1,DOWN,1) RUN | INVALID[FR,1,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,1,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |
| 13 | (FR,1,UP,0) (FR,0,UP,2) RUN | INVALID[FR,0,UP,2] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,0,UP,2] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |
| 14 | (FR,1,UP,0) (FR,0,DOWN,2) RUN | INVALID[FR,0,DOWN,2] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,0,DOWN,2] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |

**缺陷截图**

**截图1**

程序说明书中说了，相同楼层的请求应该输出关门之后的时间，但是这里却把它当成相同请求，这是一个比较严重的缺陷。

（3）电梯内的请求等价类生成测试用例如下：

黑盒测试（电梯内部请求等价类生成测试用例）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 | 备注 |
| 1 | (FR,1,UP,0) (ER,11,1) RUN | INVALID[ER,11,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[ER,11,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |
| 2 | (FR,1,UP,0) (ER,10,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,10,1]/(10,UP,5.5) | [FR,1,UP,0]/(1,STILL,1.0) [ER,10,1]/(10,UP,5.5) | Passed | - | - | - |
| 3 | (FR,1,UP,0) (ER,9,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,1]/(9,UP,5.0) | Passed | - | - | - |
| 4 | (FR,1,UP,0) (ER,5,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,5,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,5,1]/(5,UP,3.0) | Passed | - | - | - |
| 5 | (FR,1,UP,0) (ER,2,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0) [ER,2,1]/(2,UP,1.5) | Passed | - | - | - |
| 6 | (FR,1,UP,0) (ER,1,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,1,1]/(1,STILL,2.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,1,1]/(1,STILL,2.0) | Passed | - | - | - |
| 7 | (FR,1,UP,0) (ER,0,1) RUN | INVALID[ER,0,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[ER,0,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - | - |

**5.1.3 决策表测试**

**设计条件桩**

1. 请求格式是否合法
2. 请求是否含有非法字符
3. 请求是否重复
4. 请求时刻是否为非负整数
5. 最后一个输入请求的下一行是否带有有 RUN 指令
6. 是否为向上请求
7. 电梯前进方向上有新的同向乘梯请求
8. 同一时刻是否有多个同向请求
9. 同一时刻是否有多个异向请求
10. 是否为“顺路捎带”请求

**设计动作桩**

1. 提示请求格式非法
2. 提示请请求格式含非法字符
3. 提示请求重复
4. 提示请求时刻非负整数
5. 向上
6. 向下
7. 过滤相同请求
8. 顺路捎带

其中A、B、C、D、E可以合并为Z请求格式非法，

**决策表 1代表‘是’ 0代表不是，以下每一列代表一个测试用例。**

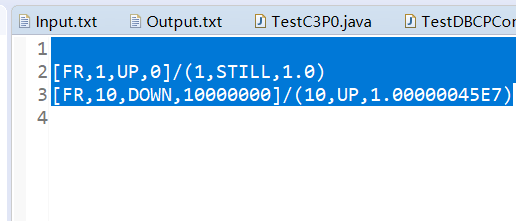
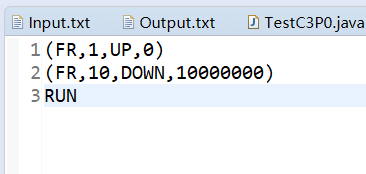
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **序号** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **条件桩** | **Z** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **0** |
| **F** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| **G** | **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **1** |
| **H** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** |
| **I** | **0** | **0** | **1** | **0** | **0** | **1** | **0** | **0** | **1** | **0** | **0** | **1** | **0** | **0** | **0** |
| **J** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** |
| **动作** | **K** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** |
| **L** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **1** |
| **M** | **1** | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **0** | **0** |
| **N** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** |
| **O** | **1** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **1** | **0** |
| **P** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** | **1** | **1** | **0** | **1** |
| **Q** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **1** | **0** |
| **R** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**根据决策表设计测试用例：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 |
| 1 | (FR,1,UP,0) (FR,11,DOWN,1) RUN | INVALID[FR,11,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,11,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - |
| 2 | ((FR,1,UP,0)  (ER,8,1)  (FR,1,UP,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,10,DOWN,1]/(10,UP,6.5)  [FR,1,UP,0]/(1,STILL,1.0)  [FR,3,UP,1]/(3,UP,2.0)  [FR,10,DOWN,1]/(10,UP,6.5) | INVALID[FR,11,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - |
| 3 | (FR,1,UP,0)  (FR,10,DOWN,10000000)  RUN | INVALID[FR,11,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,10,DOWN,10000000]/(10,UP,1.00000045E7) | Filed | - | - |
| 4 | (FR,1,UP,0) (FR,10,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,10,DOWN,1]/(10,UP,5.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,10,DOWN,1]/(10,UP,5.5) | Passed | - | - |
| 5 | (FR,1,UP,0) (FR,9,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,1]/(9,UP,5.0) | Passed | - | - |
| 6 | (FR,1,UP,0) (FR,9,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,DOWN,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,DOWN,1]/(9,UP,5.0) | Passed | - | - |
| 7 | (FR,1,UP,0) (FR,5,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,UP,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,UP,1]/(5,UP,3.0) | Passed | - | - |
| 8 | (FR,1,UP,0) (FR,5,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,DOWN,1]/(5,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,DOWN,1]/(5,UP,3.0) | Passed | - | - |
| 9 | (FR,1,UP,0) (FR,2,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,UP,1]/(2,UP,1.5) | Passed | - | - |
| 10 | (FR,1,UP,0) (FR,2,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,DOWN,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,2,DOWN,1]/(2,UP,1.5) | Passed | - | - |
| 11 | (FR,1,UP,0) (FR,1,UP,2) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,1,UP,2]/(1,UP,3.0) | [FR,1,UP,0]/(1,STILL,1.0) SAME[FR,1,UP,2] | Failed | 5 | 5 |
| 12 | (FR,1,UP,0) (FR,1,DOWN,1) RUN | INVALID[FR,1,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,1,DOWN,1] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - |
| 13 | (FR,1,UP,0) (FR,0,UP,2) RUN | INVALID[FR,0,UP,2] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,0,UP,2] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - |
| 14 | (FR,1,UP,0) (FR,0,DOWN,2) RUN | INVALID[FR,0,DOWN,2] [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,0,DOWN,2] [FR,1,UP,0]/(1,STILL,1.0) | Passed | - | - |
| 15 | (FR,1,UP,0)  (ER,1,1)  (ER,2,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | passed |  |  |

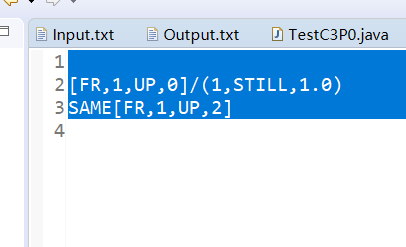
**错误截图如下：**

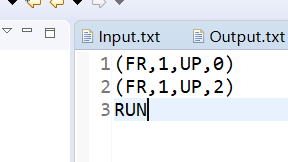
**缺陷一：可以看出当时间很长时，就会出现缺陷，第十层出现向上的状态，而本程序电梯最高只能运行到第十层，所以第十层不可能再向上。**

****

**缺陷二**

**从以下可以看出不同时刻发出的请求，而程序在此被视为同一请求。**

****

****

## 5.2 白盒测试

**5.2.1 测试的类有如下类**

**ValidateRequestion类**

isLegalRequestion

**Request类：**

isFloorRequestCompetition，isElevatorRequestCompetition

**NewRequest类：**

executeRequest，handleIncidentalRequest，getIncidentalRequest，handleIncidentalRequest

**isLegalRequestion方法：**

public static boolean isLegalRequestion(String str){

String[] strs;

strs = str.split(",");

1 for (int i = 0; i < regexStr.length; i ++){

Pattern pattern = Pattern.compile(regexStr[i]);

Matcher matcher = pattern.matcher(str);

2 if(matcher.matches() == true){

3 if (Integer.valueOf(strs[1]) < 1 || Integer.valueOf(strs[1]) > 10){

return false;

4 } else if (strs[0].equals("(FR")){

switch (strs[2]){

5 case "UP":

6 if (strs[1].equals("10")){

return false;

7 } else

return true;

8 case "DOWN":

9 if (strs[1].equals("1")){

return false;

}

10 else

return true;

}

11 } else {

return true;

}

}

}

12 return false;

}

**isLegalRequestion方法：**

public static boolean isLegalRequestion(String str){

String[] strs;

strs = str.split(",");

1 for (int i = 0; i < regexStr.length; i ++){

Pattern pattern = Pattern.compile(regexStr[i]);

Matcher matcher = pattern.matcher(str);

2 if(matcher.matches() == true){

3 if (Integer.valueOf(strs[1]) < 1 || Integer.valueOf(strs[1]) > 10){

return false;

4 } else if (strs[0].equals("(FR")){

switch (strs[2]){

5 case "UP":

6 if (strs[1].equals("10")){

return false;

7 } else

return true;

8 case "DOWN":

9 if (strs[1].equals("1")){

return false;

}

10 else

return true;

}

11 } else {

return true;

}

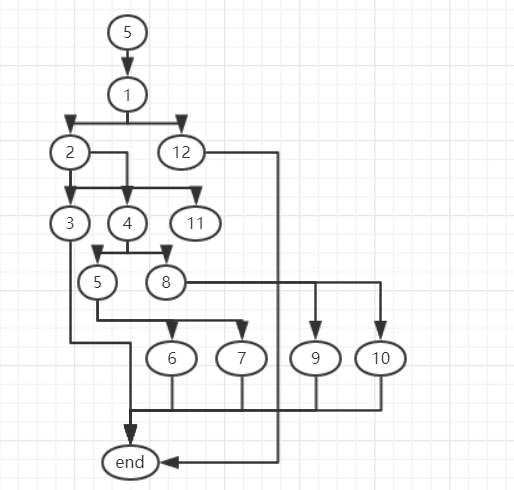
}

}

12 return false;

}

**程序节点图如下：**

****

**条件判定：**

判定1：matcher.matches() == true T1F1

判定2：Integer.valueOf(strs[1]) < 1 || Integer.valueOf(strs[1]) > 10 T2F2

判定3：strs[0].equals("(FR") T3F3

判定4：case "UP"/"DOWN" T4F4

判定5：strs[1].equals("10") T5F5

判定6：strs[1].equals("1") T6F6

**条件判定覆盖测试用例：**

IslegaRequest判定测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖判定 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,-1,DOWN,2)  RUN | INVALID[FR,-1,DOWN,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,-1,DOWN,2]  [FR,1,UP,0]/(1,STILL,1.0) | T2 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,10,UP,2)  RUN | INVALID[FR,10,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,10,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | T1F2T3T4T5 | Passed | - | - |
| 3 | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | T1F2T3T4F5 | Passed | - | - |
| 4 | (FR,1,UP,0)  (FR,1,DOWN,1)  RUN | INVALID[FR,1,DOWN,1]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,1,DOWN,1]  [FR,1,UP,0]/(1,STILL,1.0) | T1F2T3F4T6 | Passed | - | - |
| 5 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | T1F2T3F4F6 | Passed | - | - |
| 6 | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | T1F2F3 | Passed | - | - |
| 7 | (FR,1,UP,0)  (FR,2,UP,0,0)  RUN | INVALID[FR,2,UP,0,0]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,2,UP,0,0]  [FR,1,UP,0]/(1,STILL,1.0) | F1 | Passed | - | - |

**路径覆盖测试用例：**

IslegaRequest路径测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖路径 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,-1,DOWN,2)  RUN | INVALID[FR,-1,DOWN,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,-1,DOWN,2]  [FR,1,UP,0]/(1,STILL,1.0) | 1、2、3 | Passed |  | - |
| 2 | (FR,1,UP,0)  (FR,10,UP,2)  RUN | INVALID[FR,10,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,10,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | 1、2、4、5、6 | Passed | - | - |
| 3 | (FR,1,UP,0)  (FR,9,UP,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | 1、2、4、5、7 | Passed | - | - |
| 4 | (FR,1,UP,0)  (FR,1,DOWN,1)  RUN | INVALID[FR,1,DOWN,1]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,1,DOWN,1]  [FR,1,UP,0]/(1,STILL,1.0) | 1、2、4、8、9 | Passed | - | - |
| 5 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | 1、2、4、8、10 | Passed | - | - |
| 6 | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | 1、2、11 | Passed | - | - |
| 7 | (FR,1,UP,0)  (FR,2,UP,0,0)  RUN | INVALID[FR,2,UP,0,0]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,2,UP,0,0]  [FR,1,UP,0]/(1,STILL,1.0) | 1、12 | Passed | - | - |

**条件：**

条件1：matcher.matches() == true T1F1

条件2：Integer.valueOf(strs[1]) < 1 || Integer.valueOf(strs[1]) > 10 T2.1F2.1/T2.2F2.2

条件3：strs[0].equals("(FR") T3F3

条件4：case "UP"/"DOWN" T4F4

条件5：strs[1].equals("10") T5F5

条件6：strs[1].equals("1") T6F6

**条件覆盖测试用例：**

IslegaRequest条件测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖条件 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,-1,DOWN,2)  RUN | INVALID[FR,-1,DOWN,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,-1,DOWN,2]  [FR,1,UP,0]/(1,STILL,1.0) | T2.1 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,11,UP,2)  RUN | INVALID[FR,11,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,11,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | T2.2 | Passed | - | - |
| 3 | (FR,1,UP,0)  (FR,10,UP,2)  RUN | INVALID[FR,10,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,10,UP,2]  [FR,1,UP,0]/(1,STILL,1.0) | T1F2.1F2.2T3T4T5 | Passed | - | - |
| 4 | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,9,UP,2]/(9,UP,6.0) | T1F2.1F2.2T3T4F5 | Passed | - | - |
| 5 | (FR,1,UP,0)  (FR,1,DOWN,1)  RUN | INVALID[FR,1,DOWN,1]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,1,DOWN,1]  [FR,1,UP,0]/(1,STILL,1.0) | T1F2.1F2.2T3F4T6 | Passed | - | - |
| 6 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | T1F2.1F2.2T3F4F6 | Passed | - | - |
| 7 | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | T1F2.1F2.2F3 | Passed | - | - |
| 8 | (FR,1,UP,0)  (FR,2,UP,0,0)  RUN | INVALID[FR,2,UP,0,0]  [FR,1,UP,0]/(1,STILL,1.0) | INVALID[FR,2,UP,0,0]  [FR,1,UP,0]/(1,STILL,1.0) | F1 | Passed | - | - |

**isFloorRequestCompetition方法：**

public void isFloorRequestCompetition(){

1 for (int i = 0; i < arrayListFLR.size(); i ++){

2 if ((FRTime - Double.valueOf(arrayListFLR.get(i)[0])) > Double.valueOf(arrayListFLR.get(i)[1])){

3 floor[Integer.valueOf(arrayListFLR.get(i)[3]) - 1].setButtonOff(arrayListFLR.get(i)[2]);

4 arrayListFLR.remove(i); //

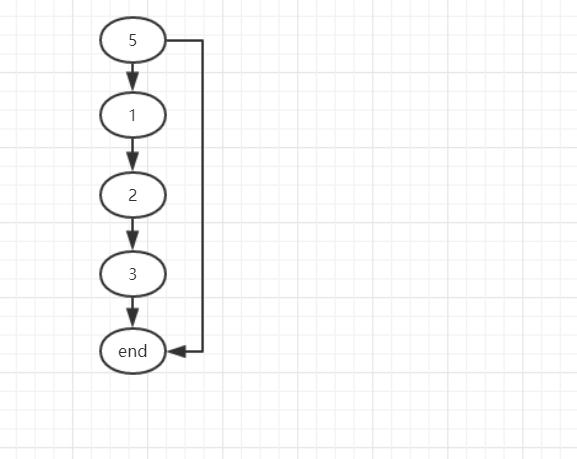
5 i--; //上面删除后，后面的元素会前移一个，所以 i--一次

}

}

}

**程序节点图如下**：



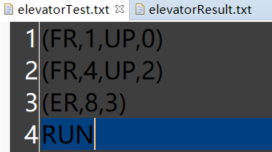
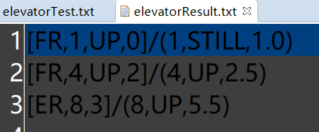
**语句覆盖测试用例：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 路径 覆盖 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 | 备注 |
| 1 | (FR,1,UP,0)  (FR,4,UP,2)  (ER,8,3)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,3.5)  [ER,8,3]/(8,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,2.5)  [ER,8,3]/(8,UP,5.5) | 1-2-3 -5 -8-9 | Falid | 4 | 4 | 截图2 |
| 2 | (FR,1,UP,0) (FR,9,UP,2) (FR,6,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | 1-2-3 -4-6-7 -8-9 | Passed | - | - | - |
| 3 | (FR,1,UP,0)  (ER,8,3)  (ER,2,10)  (ER,6,10)  RUN | FR,1,UP,0]/(1,STILL,1.0)  [ER,8,3]/(8,UP,6.5)  [ER,6,10]/(6,UP,11.0)  [ER,2,10]/(2,DOWN,13.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,3]/(8,UP,6.5)  [ER,2,10]/(2,DOWN,13.0)  [ER,6,10]/(6,UP,16.0) | 1-2-11-12 -14-16-17 | Failed | 5 | 4 | 截图3 |
| 4 | (FR,1,UP,0) (ER,9,2) (ER,6,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | 1-2-10-11 -13-14-16 -17 | Passed | - | - | - |
| 5 | (FR,1,UP,0) (ER,9,2) (ER,6,5) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) [ER,6,5]/(6,DOWN,8.5) | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) [ER,6,5]/(6,DOWN,8.5) | 1-2-10- 11-15-16-17 | Passed | - | - | - |
| 6 | (FR,1,UP,0)  (ER,6,2)  (FR,4,DOWN,4)  (FR,10,DOWN,6)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,6,2]/(6,UP,4.5)  [FR,4,DOWN,4]/(4,DOWN,6.5)  [FR,10,DOWN,6]/(10,UP,11.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,6,2]/(6,UP,4.5)  [FR,4,DOWN,4]/(4,DOWN,6.5)  [ER,6,2]/(6,UP,8.5)  [FR,10,DOWN,6]/(10,UP,11.5) | 1-2-13- 15-16-17 | Falied | 4 | 4 | 截图4 |

**存在缺陷截图**

截图2：

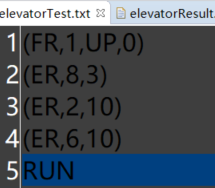
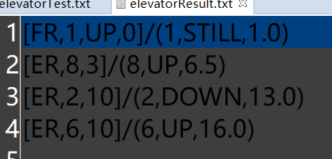
输入 输出

截图3：

图5.5.3 缺陷3截图

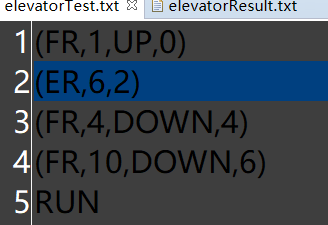
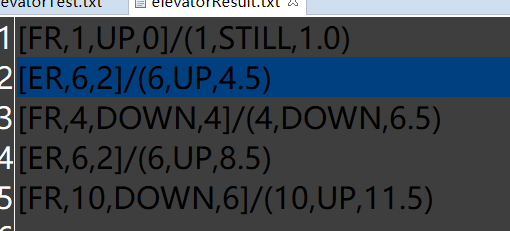
输入 输出

截图4：

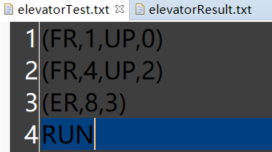
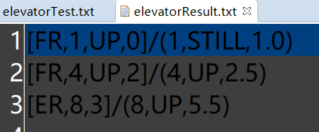
图5.5.3 缺陷3截图

输入 输出

截图2：

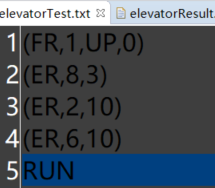
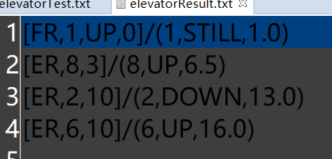
输入 输出

截图3：

图5.5.3 缺陷3截图

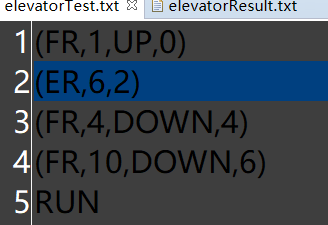
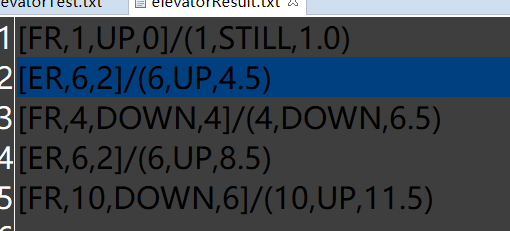
输入 输出

截图4：

图5.5.3 缺陷3截图

输入 输出

**路径覆盖测试用例：**

isFloorRequestCompetition路径测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖路径 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,4,UP,2)  (FR,7,UP,03)  RUN | (FR,1,UP,0)  (FR,4,UP,2)  (FR,7,UP,03)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,3.5)  [FR,7,UP,03]/(7,UP,6.0) | 1、2、3 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,7,UP,1)  (FR,4,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,1]/(4,UP,2.5)  [FR,7,UP,1]/(7,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,1]/(4,UP,2.5)  [FR,7,UP,1]/(7,UP,5.0) |  | Passed | - | - |

**条件-判定**：

(FRTime-Double.valueOf(arrayListFLR.get(i)[0]))>Double.valueOf(arrayListFLR.get(i)[1])

**条件-判定覆盖测试用例：**

isFloorRequestCompetition判定测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖判定 | 测试结果 | 缺陷优先级 | 缺陷严重程度 | 备注 |
| 1 | (FR,1,UP,0)  (FR,4,UP,2)  (FR,7,UP,03)  RUN | (FR,1,UP,0)  (FR,4,UP,2)  (FR,7,UP,03)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,3.5)  [FR,7,UP,03]/(7,UP,6.0) | T1 | Passed | - | - | - |
| 2 | (FR,1,UP,0)  (FR,7,UP,1)  (FR,4,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,1]/(4,UP,2.5)  [FR,7,UP,1]/(7,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,1]/(4,UP,2.5)  [FR,7,UP,1]/(7,UP,5.0) | F1 | Passed | - | - | - |

条件：

(FRTime-Double.valueOf(arrayListFLR.get(i)[0]))>Double.valueOf(arrayListFLR.get(i)[1]) T1F1

条件覆盖测试用例：

表 5.2.3 isFloorRequestCompetition条件测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖条件 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,4,UP,2)  (FR,7,UP,03)  RUN | (FR,1,UP,0)  (FR,4,UP,2)  (FR,7,UP,03)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,3.5)  [FR,7,UP,03]/(7,UP,6.0) | T1 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,7,UP,1)  (FR,4,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,1]/(4,UP,2.5)  [FR,7,UP,1]/(7,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,1]/(4,UP,2.5)  [FR,7,UP,1]/(7,UP,5.0) | F1 | Passed | - | - |

**isElevatorRequestCompetition方法：**

public void isElevatorRequestCompetition(){

1 for (int i = 0; i < arrayListELR.size(); i ++){

2 if ((ERTime - Double.valueOf(arrayListELR.get(i)[0])) > Double.valueOf(arrayListELR.get(i)[1])){

3 elevator.setIndexFloor(Integer.valueOf(arrayListELR.get(i)[2]), false);

4 arrayListELR.remove(i);

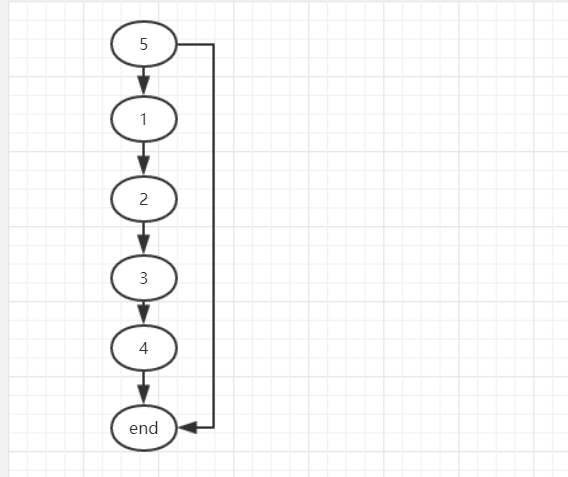
5 i--; //上面删除后，后面的元素会前移一个，所以 i--一次

}

}

}

**程序节点图：**

****

**路径覆盖测试用例：**

isElevatorRequestCompetition路径测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖路径 | 测试结果 | 缺陷优先级 | 缺陷严重程度 | 备注 |
| 1 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  (ER,1,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0)  [ER,1,2]/(1,DOWN,7.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0)  [ER,1,2]/(1,DOWN,7.0) | 1、2、3、4 | Passed | - | - | - |
| 2 | (FR,1,UP,0)  (ER,1,1)  (ER,2,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) |  | Passed |  |  |  |

**条件判定：**

**条件判定覆盖测试用例：**

(ERTime - Double.valueOf(arrayListELR.get(i)[0])) > Double.valueOf(arrayListELR.get(i)[1]) T1F1

表 5.3.3 isElevatorRequestCompetition判定测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖判定 | 测试结果 | 缺陷优先级 | 缺陷严重程度 | 备注 |
| 1 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  (ER,1,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0)  [ER,1,2]/(1,DOWN,7.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0)  [ER,1,2]/(1,DOWN,7.0) | T1 | Passed | - | - | - |
| 2 | (FR,1,UP,0)  (ER,1,1)  (ER,2,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | F1 | Passed | - | - | - |

**条件：**

(ERTime - Double.valueOf(arrayListELR.get(i)[0])) >Double.valueOf(arrayListELR.get(i)[1])

**条件覆盖测试用例：**

表 5.3.4 isElevatorRequestCompetition条件测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖判定 | 测试结果 | 缺陷优先级 | 缺陷严重程度 | 备注 |
| 1 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  (ER,1,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0)  [ER,1,2]/(1,DOWN,7.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0)  [ER,1,2]/(1,DOWN,7.0) | T1 | Passed | - | - | - |
| 2 | (FR,1,UP,0)  (ER,1,1)  (ER,2,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | F1 | Passed | - | - | - |

**executeRequest方法：**

public void executeRequest(LinkQueue que) throws NumberFormatException, IOException{

.1 Mrequest = que.poll();

2

switch(Mrequest[0]){

3 case "FR":

4 this.FRTime = Integer.valueOf(Mrequest[3]); //更新请求最新时间

5 isFloorRequestCompetition(); //判断执行中的请求是否完成 5

//判断该请求是否重复

6 if(control.getElevator().getFloor()[Integer.valueOf(Mrequest[1]) - 1].getfloorButton(Mrequest[2])){

FileIO.writeFile(pathW,"SAME[" +Mrequest[0]+","+Mrequest[1]+","+Mrequest[2]+","+Mrequest[3]+"]"); //输出重复请求

return ;

}

7 control.getElevator().getFloor()[Integer.valueOf(Mrequest[1]) - 1].setButtonOn(Mrequest[2]); //设置目标楼层按钮 7

8 elevator.setRequest(Integer.valueOf(Mrequest[1]) - 1,Mrequest[0] +","+Mrequest[1] + ","+Mrequest[2] + ","+Mrequest[3]);

1. runTime = control.getExpectedTime(Integer.valueOf(Mrequest[1]));
2. getIncidentalRequest(que);

11 if (a.size() != 0){

handleIncidentalRequest();

getResult(Integer.valueOf(Mrequest[1]),que);

}

12 else {

Time = (Time > Integer.valueOf(Mrequest[3])) ? Time : Integer.valueOf(Mrequest[3]);

getResult(Integer.valueOf(Mrequest[1]),que);

}

13 saveExecutedFR(Mrequest);

break;

14 case "ER":

15 this.ERTime = Integer.valueOf(Mrequest[2]);

16 isElevatorRequestCompetition();

17 if(elevator.getIndexFloor(Integer.valueOf(Mrequest[1]) - 1)){ //判断该请求是否重复

FileIO.writeFile(pathW,"SAME[" +Mrequest[0]+","+Mrequest[1]+","+Mrequest[2]+"]");

return ;

}

18 elevator.setIndexFloor(Integer.valueOf(Mrequest[1]) - 1, true);

19 elevator.setRequest(Integer.valueOf(Mrequest[1]) - 1,Mrequest[0] +","+Mrequest[1] + ","+Mrequest[2]);

20 runTime = control.getExpectedTime(Integer.valueOf(Mrequest[1]));

22 if (a.size() != 0){

handleIncidentalRequest();

getResult(Integer.valueOf(Mrequest[1]),que);

23 } else

{

Time = (Time > Integer.valueOf(Mrequest[2])) ? Time : Integer.valueOf(Mrequest[2]);

getResult(Integer.valueOf(Mrequest[1]),que);

}

24 saveExecutedER(Mrequest);

default:

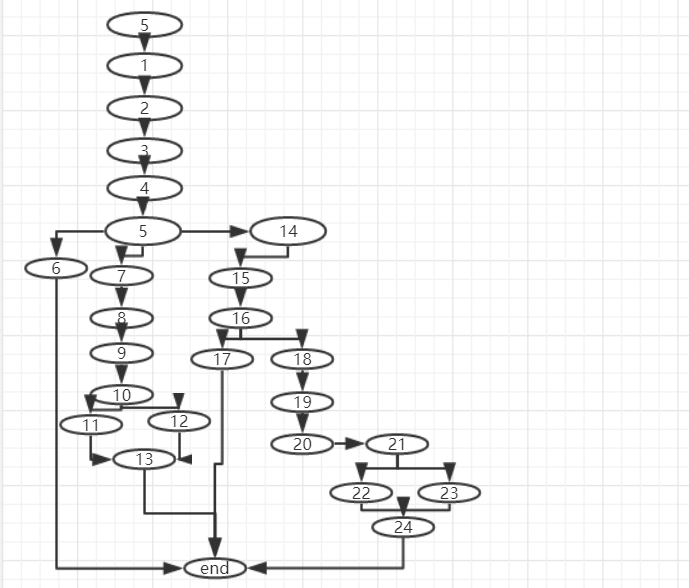
break;

}

return ;

}

**程序节点图：**



**路径覆盖测试用例：**

executeRequest 路径测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖路径 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,4,UP,2)  (ER,8,3)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,3.5)  [ER,8,3]/(8,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,2.5)  [ER,8,3]/(8,UP,5.5) | 1、2、3、4、5、6 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,2,UP,1)  (FR,5,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | 1、2、3、4、5、7、8、9、10、11、13 | Passed | - | - |
| 3 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | 1、2、3、4、5、7、8、9、10、12、13 | Passed | - | - |
| 4 | (FR,1,UP,0)  (ER,2,1)  (ER,10,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | 1、2、14、15、16、17 | Passed | - | - |
| 5 | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | 1、2、14、15、16、18、19、20、21、22、24 | Passed | - | - |
| 6 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | 1、14、15、16、18、19、20、21、23、24 | Passed | - | - |

**条件判定：**

①case "FR":

②control.getElevator().getFloor()[Integer.valueOf(Mrequest[1]) - 1].getfloorButton(Mrequest[2]):

③a.size() != 0

④case "ER":

⑤elevator.getIndexFloor(Integer.valueOf(Mrequest[1]) - 1)

⑥a.size() != 0

**条件判定覆盖测试用例：**

executeRequest 判定测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖判定 | 测试结果 | 缺陷优先级 | 缺陷严重程度 | 备注 |
| 1 | (FR,1,UP,0)  (FR,2,UP,1)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  SAME[FR,2,UP,1] | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  SAME[FR,2,UP,1] | T1T2F4 | Passed | - | - | - |
| 2 | (FR,1,UP,0)  (FR,2,UP,1)  (FR,5,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | T1F2T3F4 | Passed | - | - | - |
| 3 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | T1F2F3F4 | Passed | - | - | - |
| 4 | (FR,1,UP,0)  (ER,2,1)  (ER,10,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | F1T4T5 | Passed | - | - | - |
| 5 | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | F1T4F5T6 | Passed | - | - | - |
| 6 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | F1T4F5F6 | Passed | - | - | - |

**条件**：

①case "FR": T1F1

②control.getElevator().getFloor()[Integer.valueOf(Mrequest[1]) - 1].getfloorButton(Mrequest[2]):T2F2

③a.size() != 0 T3F3

④case "ER": T4F4

⑤elevator.getIndexFloor(Integer.valueOf(Mrequest[1]) - 1)

⑥a.size() != 0 T5F5

**条件覆盖测试用例：**

executeRequest 条件测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖条件 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,2,UP,1)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  SAME[FR,2,UP,1] | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  SAME[FR,2,UP,1] | T1T2F4 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,2,UP,1)  (FR,5,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | T1F2T3F4 | Passed | - | - |
| 3 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | T1F2F3F4 | Passed | - | - |
| 4 | (FR,1,UP,0)  (ER,2,1)  (ER,10,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | F1T4T5 | Passed | - | - |
| 5 | (FR,1,UP,0)  (ER,2,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5) | F1T4F5T6 | Passed | - | - |
| 6 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | F1T4F5F6 | Passed | - | - |

**handleIncidentalRequest方法：**

public void handleIncidentalRequest() throws IOException{

1. for (int i = 0; i < a.size(); i++){

2. switch(a.get(i)[0]){

3. case "FR":

4. if (control.getElevator().getFloor()[Integer.valueOf(a.get(i)[1]) - 1].getfloorButton(a.get(i)[2])){

5. FileIO.writeFile(pathW,"SAME[" + a.get(i)[0] +","+a.get(i)[1] + ","+a.get(i)[2]+ ","+a.get(i)[3]+"]");

6. } else{

7. control.getElevator().getFloor()[Integer.valueOf(a.get(i)[1]) - 1].setButtonOn(a.get(i)[2]);

elevator.setRequest(Integer.valueOf(a.get(i)[1]) - 1,a.get(i)[0] +","+a.get(i)[1] + ","+a.get(i)[2] + ","+a.get(i)[3]);

}

8. a.remove(i);

i--;

9. break;

10. case "ER":

11. if(Integer.valueOf(a.get(i)[1]) > Integer.valueOf(Mrequest[1]) && elevator.getState().equals("UP")

|| Integer.valueOf(a.get(i)[1]) < Integer.valueOf(Mrequest[1]) && elevator.getState().equals("DOWN"))

12. Mrequest[1] = a.get(i)[1];

13. if (control.getElevator().getIndexFloor(Integer.valueOf(a.get(i)[1]) - 1)){

14. FileIO.writeFile(pathW,"SAME[" + a.get(i)[0] +","+a.get(i)[1] + ","+a.get(i)[2]+"]");

} else{

15. control.getElevator().setIndexFloor(Integer.valueOf(a.get(i)[1]) - 1, true);

elevator.setRequest(Integer.valueOf(a.get(i)[1]) - 1,a.get(i)[0] +","+a.get(i)[1] + ","+a.get(i)[2]);

}

16. a.remove(i);

i--;

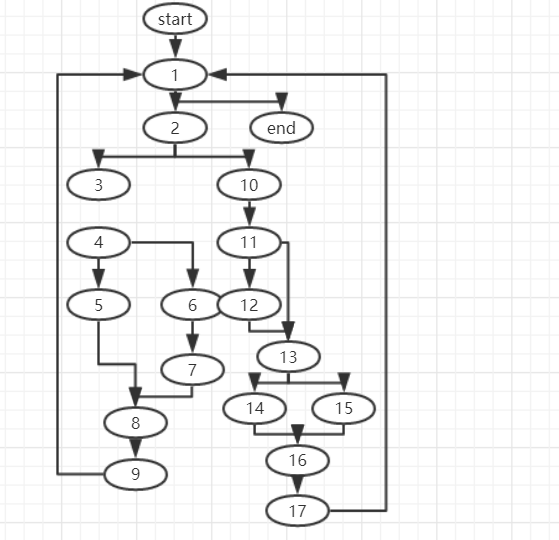
17. break;

}

}

**程序节点图节点图**

handleIncidentalRequest节点图



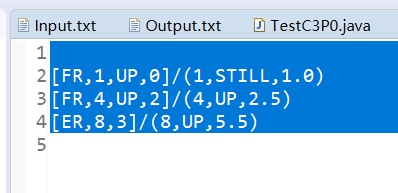
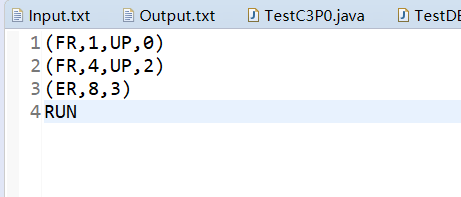
**路径测试有：**

handleIncidentalRequest路径测试

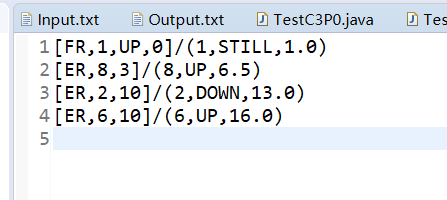
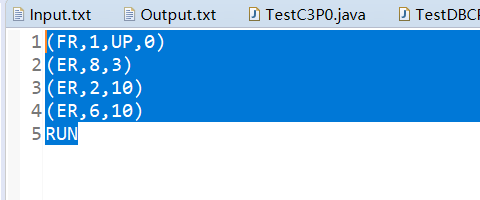
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 路径 覆盖 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 |
| 1 | (FR,1,UP,0)  (FR,4,UP,2)  (ER,8,3)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,3.5)  [ER,8,3]/(8,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,4,UP,2]/(4,UP,2.5)  [ER,8,3]/(8,UP,5.5) | 1-2-3 -5 -8-9 | Faild | 4 | 4 |
| 2 | (FR,1,UP,0) (FR,9,UP,2) (FR,6,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | 1-2-3 -4-6-7 -8-9 | Passed | - | - |
| 3 | (FR,1,UP,0)  (ER,8,3)  (ER,2,10)  (ER,6,10)  RUN | FR,1,UP,0]/(1,STILL,1.0)  [ER,8,3]/(8,UP,6.5)  [ER,6,10]/(6,UP,11.0)  [ER,2,10]/(2,DOWN,13.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,3]/(8,UP,6.5)  [ER,2,10]/(2,DOWN,13.0)  [ER,6,10]/(6,UP,16.0) | 1-2-11-12 -14-16-17 | Failed | 5 | 4 |
| 4 | (FR,1,UP,0) (ER,9,2) (ER,6,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | 1-2-10-11 -13-14-16 -17 | Passed | - | - |
| 5 | (FR,1,UP,0) (ER,9,2) (ER,6,5) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) [ER,6,5]/(6,DOWN,8.5) | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) [ER,6,5]/(6,DOWN,8.5) | 1-2-10- 11-15-16-17 | Passed | - | - |
| 6 | (FR,1,UP,0)  (ER,6,2)  (FR,4,DOWN,4)  (FR,10,DOWN,6)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,6,2]/(6,UP,4.5)  [FR,4,DOWN,4]/(4,DOWN,6.5)  [FR,10,DOWN,6]/(10,UP,11.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,6,2]/(6,UP,4.5)  [FR,4,DOWN,4]/(4,DOWN,6.5)  [ER,6,2]/(6,UP,8.5)  [FR,10,DOWN,6]/(10,UP,11.5) | 1-2-13- 15-16-17 | Falied | 4 | 4 |

**缺陷截图如下：**

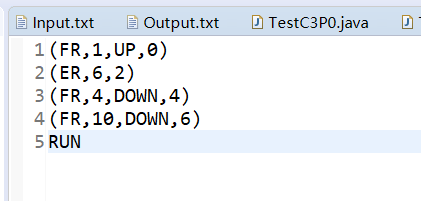
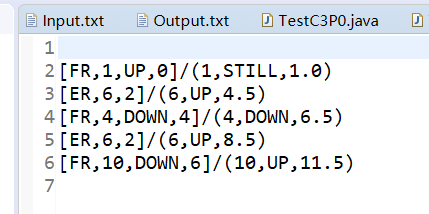
**缺陷一：**

****

**缺陷二：**

****

**缺陷三：**

****

**判定一** T1 F1

if(control.getElevator().getFloor()[Integer.valueOf(a.get(i)[1]) - 1].getfloorButton(a.get(i)[2]))

**判定二** T2 F2

if(Integer.valueOf(a.get(i)[1]) > Integer.valueOf(Mrequest[1]) && elevator.getState().equals("UP")

**判定三**T3 F3

if (control.getElevator().getIndexFloor(Integer.valueOf(a.get(i)[1]) - 1))

**条件判定覆盖有：**

handleIncidentalRequest路径测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 覆盖 条件 | 测试结果 | 缺陷 严重 程度 |
| 1 | (FR,1,UP,0) (FR,9,UP,2) (FR,9,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,2]/(9,UP,6.0) SAME[FR,9,UP,3] | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,2]/(9,UP,6.0) SAME[FR,9,UP,3] | T1 | Passed | - |
| 2 | (FR,1,UP,0) (FR,9,UP,2) (FR,6,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | F1 | Passed | - |
| 3 | (FR,1,UP,0) (ER,9,2) (ER,9,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | F2,T3 | Passed | - |
| 4 | (FR,1,UP,0) (ER,9,2) (ER,6,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | T2,F3 | Passed | - |

**条件1** T1 F1

control.getElevator().getFloor()[Integer.valueOf(a.get(i)[1]) - 1].getfloorButton(a.get(i)[2])

**条件2** T2 F2

Integer.valueOf(a.get(i)[1]) > Integer.valueOf(Mrequest[1])

**条件3** T3 F3

elevator.getState().equals("UP")

**条件4** T4 F4

control.getElevator().getIndexFloor(Integer.valueOf(a.get(i)[1]) - 1)

**条件覆盖有：**

**测试用例如下：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 条件 覆盖 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 |
| 1 | (FR,1,UP,0) (FR,9,UP,2) (FR,9,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,2]/(9,UP,6.0) SAME[FR,9,UP,3] | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,2]/(9,UP,6.0) SAME[FR,9,UP,3] | T1 | Passed | - | - |
| 2 | (FR,1,UP,0) (FR,9,UP,2) (FR,6,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | F1,T2 | Passed | - | - |
| 3 | (FR,1,UP,0) (ER,9,2) (ER,9,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | F1,F2,T3 | Passed | - | - |
| 4 | (FR,1,UP,0) (ER,9,2) (ER,6,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [ER,6,3]/(6,UP,3.5) [ER,9,2]/(9,UP,6.0) | F1,F2,F3,T4 | Passed | - | - |
| 5 | (FR,1,UP,0) (ER,9,2) (ER,6,5) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) [ER,6,5]/(6,DOWN,8.5) | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) [ER,6,5]/(6,DOWN,8.5) | F1,F2,F3,F4 | Passed | - | - |

**getIncidentalRequest方法**

public void getIncidentalRequest(LinkQueue que){

1. for(int i = 0; i < que.queueSize(); i ++){

2. switch(que.get(i)[0]){

3. case "FR":

4. if (elevator.getState().equals(que.get(i)[2])

&& Integer.valueOf(que.get(i)[1]) < getMax(elevator.getCurrentFloor(),Integer.valueOf(Mrequest[1]))

&& Integer.valueOf(que.get(i)[1]) > getMin(elevator.getCurrentFloor(),Integer.valueOf(Mrequest[1]))

&& Integer.valueOf(que.get(i)[3]) - Time < control.getExpectedTime(Integer.valueOf(que.get(i)[1]))){

5. que.toHead(i);

a.add(que.poll());

i--;

}

6. break;

7. case "ER":

8. if ((elevator.getState().equals("UP") && Integer.valueOf(que.get(i)[1]) > elevator.getCurrentFloor()

|| elevator.getState().equals("DOWN") && Integer.valueOf(que.get(i)[1]) < elevator.getCurrentFloor())

&& Integer.valueOf(que.get(i)[2]) - Time < control.getExpectedTime(Integer.valueOf(que.get(i)[1]))

&& Integer.valueOf(que.get(i)[1]) != Integer.valueOf(Mrequest[1])){

9. que.toHead(i);

a.add(que.poll());

i--;

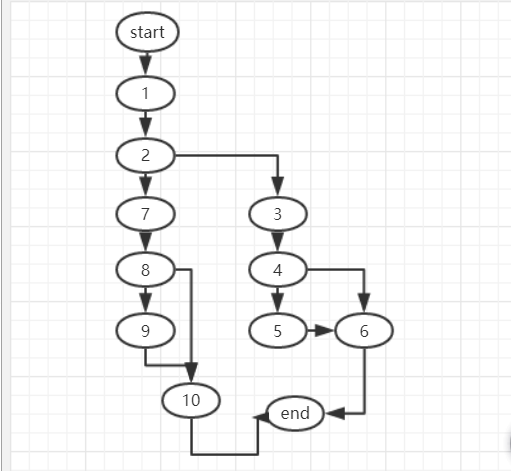
}

10. break;

}

}

**程序节点图如下：**



**基本路径测试**

getIncidentalRequest 路径测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预计输出 | 实际输出 | 路径 覆盖 | 测试结果 | 缺陷 优先级 | 缺陷 严重 程度 |
| 1 | (FR,1,UP,0) (ER,5,3) (ER,6,5) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,5,3]/(5,UP,5.0) [ER,6,5]/(6,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0) [ER,5,3]/(5,UP,5.0) [ER,6,5]/(6,UP,6.5) | 1-2-3 4-5-6 | Passed | - | - |
| 2 | (FR,1,UP,0) (FR,9,UP,2) (FR,6,UP,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,6,UP,3]/(6,UP,3.5) [FR,9,UP,2]/(9,UP,6.0) | 1-2-3 -4-6 | Passed | - | - |
| 3 | (FR,1,UP,0) (ER,9,2) (ER,9,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | 1-2-7-9-9-10 | Passed | - | - |
| 4 | (FR,1,UP,0) (FR,5,UP,3) (FR,4,UP,7) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,UP,3]/(5,UP,5.0) [FR,4,UP,7]/(4,DOWN,7.5) | [FR,1,UP,0]/(1,STILL,1.0) [FR,5,UP,3]/(5,UP,5.0) [FR,4,UP,7]/(4,DOWN,7.5) | 1-2-7-8-10 | Passed | - | - |

**条件判定覆盖测试：**

判定1：if (elevator.getState().equals(que.get(i)[2]) ) T1，F1

判定2：if ((elevator.getState().equals("UP") && Integer.valueOf(que.get(i)[1]) > elevator.getCurrentFloor()

**条件判定覆盖测试用例有**：

getIncidentalRequest 判定测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖判定 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (ER,2,1)  (ER,5,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,5,1]/(5,UP,4.0) | T1 | Passed | - | - |
| 2 | (FR,1,UP,0)  (ER,1,1)  (ER,2,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,1,1]/(1,STILL,2.0)  [ER,2,2]/(2,UP,2.5) | F1,T2 | Passed | - | - |
| 3 | (FR,1,UP,0) (ER,9,2) (ER,9,3) RUN | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | [FR,1,UP,0]/(1,STILL,1.0) [ER,9,2]/(9,UP,6.0) SAME[ER,9,3] | F2 | Passed | - | - |

**条件覆盖测试**

条件有：

**条件1** T1 F1

Integer.valueOf(que.get(i)[1]) < getMax(elevator.getCurrentFloor(),Integer.valueOf(Mrequest[1])

**条件2**  T2 F2

Integer.valueOf(que.get(i)[3]) - Time < control.getExpectedTime(Integer.valueOf(que.get(i)[1]))

**条件3** T3 F3

(elevator.getState().equals("UP") && Integer.valueOf(que.get(i)[1]) > elevator.getCurrentFloor()

**条件4** T4 F4

Integer.valueOf(que.get(i)[1]) != Integer.valueOf(Mrequest[1])

getIncidentalRequest 条件测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 输入 | 预期输出 | 实际输出 | 覆盖条件 | 测试结果 | 缺陷优先级 | 缺陷严重程度 |
| 1 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | T1T2 | Passed | - | - |
| 2 | (FR,1,UP,0)  (FR,2,UP,1)  (FR,5,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5)  [FR,5,UP,1]/(5,UP,4.0) | T1F2 | Passed | - | - |
| 3 | (FR,1,UP,0)  (FR,2,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,2,UP,1]/(2,UP,1.5) | F1F2T3 | Passed | - | - |
| 4 | (FR,1,UP,0)  (ER,2,1)  (ER,10,2)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,2,1]/(2,UP,1.5)  [ER,10,2]/(10,UP,6.5) | F1F2T3T4 | Passed | - | - |
| 5 | (FR,1,UP,0) (FR,9,UP,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,UP,1]/(9,UP,5.0) | T2T3T4 | Passed | - | - |
| 6 | (FR,1,UP,0) (FR,9,DOWN,1) RUN | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,DOWN,1]/(9,UP,5.0) | [FR,1,UP,0]/(1,STILL,1.0) [FR,9,DOWN,1]/(9,UP,5.0) | T2T3F4 | Passed | - | - |

#### 判定覆盖测试

1、 FileIO.java

在此类中写main方法进行测试：

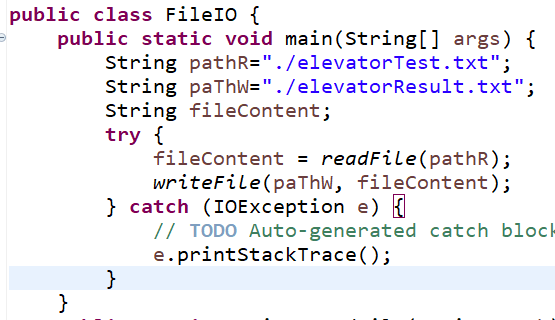


图5.2-1FileIO桩

由白盒测试判定覆盖可知：

该类有两个方法，每个方法中都有一个if判断，判定覆盖测试用例如下：

表5.2-1FileIO测试用例

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **编号** | **pathR** | **pathW** | **预计输出** | **实际输出** | **结果** | **缺陷优先级** | **缺陷优先程度** | **备注** |
| 1 | ./elevatorTest.txt | ./elevatorResult.txt | 123 | 123 | Passed |  |  |  |
| 2 | ./elevatorTest.txt |  | java.io.FileNotFoundException: File is not exist! | java.io.FileNotFoundException: File is not exist! | Passed |  |  |  |
| 3 |  | ./elevatorResult.txt | java.io.FileNotFoundException: File is not exist! | java.io.FileNotFoundException: File is not exist! | Passed |  |  |  |
| 4 |  |  | java.io.FileNotFoundException: File is not exist! | java.io.FileNotFoundException: File is not exist! | Passed |  |  |  |

2、 ValidateRequestion

这个类的作用是用正则表达式判断输入的信息格式与逻辑关系

设计桩格式：

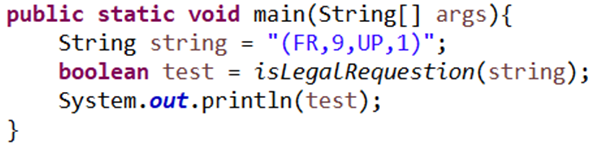


图5.2-2 ValidateRequestion桩

其中，判定表如下:

表5.2-2 ValidateRequestion判定表

|  |  |  |
| --- | --- | --- |
| matcher.matches() == **true** | T1 | F1 |
| Integer.*valueOf*(strs[1]) < 1 || Integer.*valueOf*(strs[1] | T2 | F2 |
| strs[0].equals("(FR") | T3 | F3 |
| strs[2].equals(“UP”) | T4 | F4 |
| strs[1].equals("10") | T5 | F5 |
| strs[2].equals(“DOWN”) | T6 | F6 |
| strs[1].equals("1") | T7 | F7 |

表5.2-3 ValidateRequestion测试用例

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **编号** | **string的值** | **覆盖判定** | **预期输出** | **实际输出** | **结果** | **缺陷优先级** | **缺陷优先程度** | **备注** |
| 1 | (FR,UP,1) | F1 | false | false | Passed |  |  |  |
| 2 | (FR,0,UP,1) | T1、T2 | false | false | Passed |  |  |  |
| 3 | (FR,10,UP,1) | T1、F2、T3、T4、T5、F6 | false | false | Passed |  |  |  |
| 4 | (FR,9,UP,1) | T1、F2、T3、T4、F5、F6 | false | false | Passed |  |  |  |
| 5 | (FR,1,DOWN,1) | T1、F2、T3、F4、T6、T7 | false | false | Passed |  |  |  |
| 6 | (FR,1,DOWN,1) | T1、F2、T3、F4、T6、F7 | false | false | Passed |  |  |  |

3、 NewElevator

这个类是一个关键类，需要重点测试，观察代码，排除一些基本的get、set方法，需要测试moveElevate()这个方法，这个方法中，返回的string值有四种：Bring、Arrived、OppenDoor、Pass。这四种输出在类NewRequest中有关键作用，四个返回值的表达意思分别是：需要去捎带、只需要上下一层、只需要开关门、在运行上下楼状态。

所以，根据情况做出如下的测试：

表5.2-3 NewElevator判定表

|  |  |  |
| --- | --- | --- |
| n - **this**.currentFloor > 0 | T1 | F1 |
| **this**.currentFloor == n | T2 | F2 |
| floor[**this**.currentFloor - 1].getfloorButton("UP") || indexFloor[**this**.currentFloor - 1] | T3 | F3 |
| n - **this**.currentFloor < 0 | T4 | F4 |
| floor[**this**.currentFloor - 1].getfloorButton("DOWN") || indexFloor[**this**.currentFloor - 1] | T5 | F5 |

表5.2-4 NewElevator测试用例：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **编号** | **n的值** | **覆盖判定** | **预期输出** | **实际输出** | **结果** | **缺陷优先级** | **缺陷严重程度** | **备注** |
| 1 | 5 | F1、F4 | OppenDoor | OppenDoor | Passed |  |  |  |
| 2 | 6 | T1、T2 | Arrived | Arrived | Passed |  |  |  |
| 3 | 7 | T1、F2、T3 | Bring | Bring | Passed |  |  |  |
| 4 | 3 | F1、T4、T5 | Bring | Bring | Passed |  |  |  |

## 5.3 场景测试

## 5.3.1 场景分析

先判断基本流与备选流如下：

|  |  |
| --- | --- |
| 基本流 | 乘客按下楼层按钮，电梯到达此楼层，乘客进入电梯，按下目标楼层，等待电梯到达目标楼层，离开电梯 |
| 备选流1 | 已有乘客在电梯内部时，出现捎带情况 |
| 备选流2 | 乘客按下向上楼层按钮，然后进入电梯后，按下的目标楼层低于乘客进入电梯的楼层 |
| 备选流3 | 乘客按下向下楼层按钮，然后进入电梯后，按下的目标楼层高于乘客进入电梯的楼层 |
| 备选流4 | 乘客按下向上楼层按钮后，进入电梯，错按下低于乘客进入电梯的楼层按钮，然后按下目标楼层按钮 |
| 备选流5 | 乘客按下向下楼层按钮后，进入电梯，错按下高于乘客进入电梯的楼层按钮，然后按下目标楼层按钮 |
| 备选流6 | 乘客按下楼层按钮，进入电梯后，按下本楼层按钮 |
| 备选流7 | 电梯运行过程中，出现捎带情况，但是捎带的乘客进入电梯后，按下与上一个目标楼层方向相反的新目标楼层 |

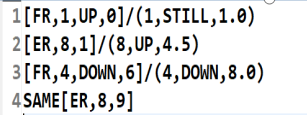
然后分析场景如下：

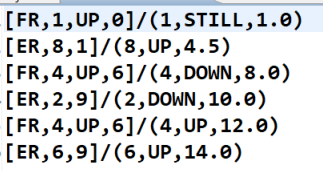
|  |  |  |
| --- | --- | --- |
| 场景1：电梯运行成功 | 基本流 |  |
| 场景2：电梯出现捎带情况 | 基本流 | 备选流1 |
| 场景3：方向向上，目标楼层低于此楼层 | 基本流 | 备选流2 |
| 场景4：方向向下，目标楼层高于此楼层 | 基本流 | 备选流3 |
| 场景5：方向向上，出现两个目标楼层，一个高于此楼层，一个低于此楼层 | 基本流 | 备选流4 |
| 场景6：方向向下，出现两个目标楼层，一个高于此楼层，一个低于此楼层 | 基本流 | 备选流5 |
| 场景7：进入电梯按下与进入电梯楼层相同的目标楼层 | 基本流 | 备选流6 |
| 场景8：捎带情况中新的目标楼层与原目标楼层方向相反 | 基本流 | 备选流7 |

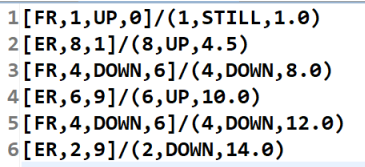
## 5.3.2 设计测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 场景 | 输入 | 预期输出 | 实际输出 | 测试结果 |
| 1 | 1 | (FR,1,UP,0)  (ER,8,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5) | Passed |
| 2 | 2 | (FR,1,UP,0)  (ER,8,1)  (FR,3,UP,1)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,3,UP,1]/(3,UP,2.0)  [ER,8,1]/(8,UP,5.5) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,3,UP,1]/(3,UP,2.0)  [ER,8,1]/(8,UP,5.5) | Passed |
| 3 | 3 | (FR,1,UP,0)  (ER,8,1)  (FR,5,UP,6)  (ER,3,9)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,5,UP,6]/(5,DOWN,7.5)  [ER,3,9]/(3,DOWN,10.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,5,UP,6]/(5,DOWN,7.5)  [ER,3,9]/(3,DOWN,10.0) | Passed |
| 4 | 4 | (FR,1,UP,0)  (ER,8,1)  (FR,4,DOWN,6)  (ER,8,9)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,DOWN,6]/(5,DOWN,8.0)  [ER,8,9]/(8,UP,11) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,DOWN,6]/(4,DOWN,8.0)  SAME[ER,8,9] | Failed |
| 5 | 5 | (FR,1,UP,0)  (ER,8,1)  (FR,4,UP,6)  (ER,2,9)  (ER,6,9)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,UP,6]/(4,DOWN,8.0)  [ER,6,9]/(6,UP,10.0)  [ER,2,9]/(2,DOWN,13.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,UP,6]/(4,DOWN,8.0)  [ER,2,9]/(2,DOWN,10.0)  [FR,4,UP,6]/(4,UP,12.0)  [ER,6,9]/(6,UP,14.0) | Failed |
| 6 | 6 | (FR,1,UP,0)  (ER,8,1)  (FR,4,DOWN,6)  (ER,6,9)  (ER,2,9)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,DOWN,6]/(4,DOWN,8.0)  [ER,2,9]/(2,DOWN,10.0)  [ER,6,9]/(6,UP,113.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,DOWN,6]/(4,DOWN,8.0)  [ER,6,9]/(6,UP,10.0)  [FR,4,DOWN,6]/(4,DOWN,12.0)  [ER,2,9]/(2,DOWN,14.0) | Failed |
| 7 | 7 | (FR,1,UP,0)  (ER,8,1)  (FR,4,DOWN,6)  (ER,4,9)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,DOWN,6]/(4,DOWN,8.0)  [ER,4,9]/(4,STILL,10.0) | [FR,1,UP,0]/(1,STILL,1.0)  [ER,8,1]/(8,UP,4.5)  [FR,4,DOWN,6]/(4,DOWN,8.0)  [ER,4,7]/(4,STILL,10.0) | Passed |
| 8 | 8 | (FR,1,UP,0)  (ER,8,1)  (FR,5,UP,1)  (ER,3,3)  RUN | [FR,1,UP,0]/(1,STILL,1.0)  [FR,5,UP,1]/(5,UP,3.0)  [ER,8,1]/(8,UP,5.5)  [ER,3,3]/(3,DOWN,9.0) | [FR,1,UP,0]/(1,STILL,1.0)  [FR,5,UP,1]/(5,UP,3.0)  [ER,8,1]/(8,UP,5.5)  [ER,3,3]/(3,DOWN,9.0) | Passed |

## 5.3.3 缺陷截图







**5.4 小组分工**

朱盛：黑盒测试：等价类测试；白盒测试：语句覆盖法。

朱天保：黑盒测试：边界值测试；白盒测试：判定覆盖法。

王文涛：场景测试。整合报告。

王慎瑜：黑盒测试：决策表测试；白盒测试：条件覆盖、判定条件覆盖、路径覆盖法。

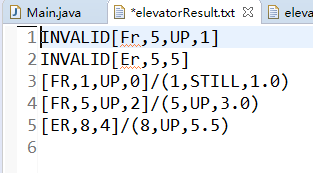
# 6 缺陷管理与分析

## 6.1 缺陷概述

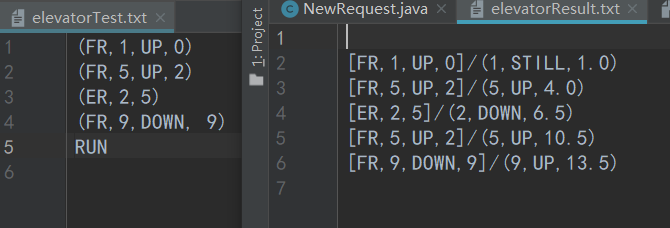
软件缺陷（Defect），常常又被叫做Bug。所谓软件缺陷，即为计算机软件或程序中存在的某种破坏正常运行能力的问题、错误，或者隐藏的功能缺陷。缺陷的存在会导致软件产品在某种程度上不能满足用户的需要。IEEE729-1983对缺陷有一个标准的定义：从产品内部看，缺陷是软件产品开发或维护过程中存在的错误、毛病等各种问题；从产品外部看，缺陷是系统所需要实现的某种功能的失效或违背。

## 6.2 缺陷内容

黑盒测试内容：输出文件第4行，请求在第2s发出，执行结果中的时间应该是4.0，输出文件第5行，电梯在1s时已经完成了第一条请求，而（FR,1,UP,2）却被认为是相同操作；输出文件第8行，在第9层请求的是DOWN，而第9条请求的执行结果，电梯的运动状态是UP;



白盒测试内容：本来只有四条输入指令，但是，最后结果出现了五条输出指令，最后检查发现是第二条指令被重复执行了一次；本来只有四条输入指令，但，第二条指令被重复执行了一次。导致出现了五条输出。



## 6.3 缺陷分析

对于同楼层的请求，假如前一个请求已经执行完毕，电梯开关门已经完成后，则之后的同楼层请求不应视为相同请求；电梯的状态应该设置为当前执行的请求的状态；在电梯外请求后下一个请求如果是电梯内请求结果可能会出错，不同数据结果会变，算法可能有问题。

指令被重复执行了，观察源代码后发现是因为在处理LinkQueue的时候，程序将电梯外的指令（FR）和电梯内指令（ER）分开处理，最后程序没有将队列中的FR项项清空，导致这条指令被保存在了队列中被再次执行了一次。本来不同的指令被程序判断为了相同指令，但是最后程序并不是没有做处理就扔掉了，而是执行了该指令但是在输出的时候还是说为相同指令，之所以出现这种错误是在程序判断为相同指令的时候没有考虑时时刻，但是后台却还是仍然执行了这条指令。

电梯本应该遵循“顺路捎带”原则捎带(ER,5,20)请求，可是电梯却先执行了(ER,1,20)请求。由此推测该缺陷可能是由于“顺路捎带”算法没有考虑到同一时刻发出的请求。电梯本应该遵循捎带原则在执行电梯内的请求(ER,1,20)后捎带电梯外的请求(FR,3,DOWN,20)，但是电梯却先执行了(ER,1,20)。同样推测改缺陷是由于“顺路捎带”算法没有考虑到同一时刻发出的请求而产生的。刚开始电梯外发出了(FR,1,UP,0)请求，在一段时间后电梯外又发出了请求(FR,1,UP,20)，但是却被电梯认为是同一请求。由此推测该缺陷产生的原因可能是因为算法没有考虑到不同时刻的同一请求的不同性。

## 6.4 总结

在黑盒测试中发现的缺陷，其中大部分是与边界值有关，所以在电梯程序中可能要对其边界值部分的代码进行再次的修改，以保证电梯边界功能的完整性。在场景测试与白盒测试中发现的缺陷大部分与电梯的调度算法有关，因为这部分算法又恰巧是整个程序代码的核心，所以需要开发者再次认真且谨慎的去完善这一部分代码，使得电梯的功能需求能正真的被完善。另外在白盒测试中发现部分代码存在冗余、缺少注释等，同样这也需要开发人员进行修改。

# 结论与展望

通过这次测试，我们可以看出电梯程序在结构性和功能性上还有些缺陷，具体缺陷如下：

一、黑盒测试缺陷：

(1)相同楼层不同时间请求会被程序视为相同请求 。

(2)假如电梯在接受请求，请求执行完成后的电梯状态可能会出现错误，如电梯在第 5层接收第九9层的下行请求，在输出执行结果中电梯状态是向上的

(3)在电梯初始化时，对于电梯外和电梯内的请求可能会出错，如缺陷1-3的截图所示

(4)‘顺路捎带’算法可能没有考虑到相同时刻发出的请求

二、白盒测试缺陷：

(1)指令会被重复执行。程序在处理LinkQueue的时候，将电梯外的指令（FR）和电梯内指令（ER）分开处理，最后程序没有将队列中的FR项清空，导致这条指令被保存在了队列中被再次执行了一次

(2) 本来不同的指令被程序判断为了相同指令。最后程序并不是没有做处理就扔掉了，而是执行了该指令但是在输出的时候还是说为相同指令，之所以出现这种错误是在程序判断为相同指令的时候没有考虑时时刻，但是后台却还是仍然执行了这条指令

对于电梯程序来说，单用黑盒测试或者白盒测试是无法保证测试的完备性的，所以本次测试采用了黑盒测试，白盒测试和场景测试，结合三种测试方法对电梯程序进行测试。由于本次测试程序是小组分工进行的，通过这次测试，对于我们小组的每个成员来说，都增加了对软件测试的认识，对测试的技术方法更加理解，让我们在软件测试方面积累了宝贵的经验。

软件测试是软件工程中非常重要的一部分，它决定了软件最终的质量。因此软件测试人员一定要认真熟读软件规格说明书，理解用户的需求，选择合适的测试技术，设计严谨的测试用例，保证测试的完整性。要确保软件是按照设计的要求运行的，以站在用户的角度进行安全性和可用性的测试。随着科技水平的提升，对软件的要求也越来越严格，必然会出现各种各样的软件开发的新技术，相应的软件测试也需要一定的技术更新，对于新技术的开发，需要我们不懈的努力。

1. [↑](#footnote-ref-0)
2. [↑](#footnote-ref-1)