

重庆大学《软件测试》课程结课报告



题 目：基于 space 的软件测试实验

学 生：_____

专 业：软件工程 2021 级 X 班

教 师：_____XXX_____

重庆大学 大数据与软件学院

二〇二四年 四 月 八 日

目录

一、引言.....	6
1.1 实验简介.....	6
1.1.1 项目来源.....	6
1.1.2 实验目的.....	6
1.2 测试环境.....	7
1.2.1 硬件环境.....	7
1.2.2 软件环境.....	7
1.3 测试团队信息.....	8
二、实验原理.....	9
2.1 白盒测试.....	9
2.2 语句覆盖.....	9
2.3 分支覆盖.....	10
2.4 函数覆盖.....	11
2.5 实验流程.....	12
2.5.1 整体概述.....	12
2.5.2 源代码修正.....	13
2.5.3 GCC 编译.....	14
2.5.4 选择测试组.....	15
2.5.5 生成覆盖率报告.....	16
2.5.6 Diff 命令比较输出结果.....	18
三、实验过程记录.....	19
3.1 环境配置.....	19
3.2 前期准备.....	19
3.2.1 源代码修正脚本.....	19
3.2.2 准备源代码.....	20
3.2.3 选择测试集.....	20
3.3 编写测试脚本 tScript.sh.....	22
3.3.1 对正确版本进行测试.....	22
3.3.2 对错误版本进行测试.....	24
3.3.3 比较输出结果.....	26
3.4 编写代码覆盖率统计脚本 getCovInfo.sh.....	26
3.5 测试过程.....	27
3.5.1 运行脚本.....	27
3.5.2 运行过程.....	28
3.5.3 运行结束.....	30
四、结果及分析.....	33
4.1 覆盖率结果及分析.....	33
4.1.1 覆盖率结果.....	33
4.1.2 分析.....	46
4.2 测试用例通过结果及分析.....	47
4.2.1 测试用例通过结果.....	47
4.2.2 分析.....	50

结论.....	52
参考文献.....	53

图索引

图 1 语句覆盖与分支覆盖.....	11
图 2 整体流程图.....	13
图 3 space 源代码修改前.....	13
图 4 space 源代码修改前.....	13
图 5 测试用例总览.....	15
图 6 测试集总览.....	15
图 7 测试集示意图.....	16
图 8 源代码修正脚本.....	19
图 9 正确版本.....	20
图 10 错误版本（共 38 个）.....	20
图 11 测试用例集合.....	20
图 12 测试集总览.....	21
图 13 测试集示意图.....	21
图 14 选择测试集.....	22
图 15 正确版本测试脚本.....	23
图 16 错误版本测试脚本.....	25
图 17 比较输出结果脚本.....	26
图 18 代码覆盖率统计脚本.....	26
图 19 CovInfo 文件内容示意图.....	27
图 20 脚本调用.....	27
图 21 运行脚本 tScript.sh.....	28
图 22 脚本正常运行过程.....	29
图 23 测试用例使程序无法正常运行时.....	30
图 24 覆盖率信息汇总文件.....	31
图 25 各版本覆盖率信息.....	31
图 26 供比对的程序输出结果.....	32
图 27 对比结果文件.....	32
图 28 suite0 部分内容.....	33
图 29 覆盖率折线图.....	44
图 30 源代码运行情况示例 1.....	45
图 31 源代码运行情况示例 2.....	46
图 32 测试用例通过结果.....	48
图 33 通过率折线图.....	50

表索引

表 1 硬件环境.....	7
表 2 软件环境.....	7
表 3 GCC 编译器添加选项.....	14
表 4 lcov 命令解析.....	16
表 5 Genhtml 命令解析.....	17
表 6 常用 diff 命令解析.....	18
表 7 覆盖率表格.....	43
表 8 通过率表格.....	47

一、引言

1.1 实验简介

1.1.1 项目来源

Software-artifact Infrastructure Repository (简称 SIR) 是一个在软件工程领域具有重要影响力的项目,旨在为程序分析和软件测试技术提供严格的控制实验支持,同时也是软件测试实验教育的重要资源。其对软件工程领域,特别是软件测试和分析方面产生了显著影响,它使研究人员能够进行控制实验,从而发展和完善软件测试技术。SIR 的标准化方法还有助于研究成果的可重复性,这对于该领域的科学知识进步至关重要。

SIR 项目起源于 2005 年发表在《Empirical Software Engineering》期刊上的一篇文章,该文章提出了建立标准化基础设施以促进软件测试和分析的控制实验的需求。为了满足这一需求, SIR 收集了一系列软件工件,供研究人员测试和评估各种软件测试技术及工具。

SIR 包含了多种编程语言编写的软件系统,如 Java、PHP、C#以及 C/C++,这些软件系统以多个版本提供,便于研究软件演化和维护的不同方面。除了软件系统, SIR 还包括了测试套件、故障数据和脚本等辅助工件,这些都是进行实验必不可少的。本次实验使用的是 C 语言程序 space。

space 由 9564 行 C 代码(6218 行可执行文件)组成,充当数组定义语言(ADL)的解释器。该程序读取包含多个 ADL 语句的文件,并检查文件内容是否符合 ADL 语法和特定的一致性规则。如果 ADL 文件正确, space 输出一个数组数据文件,其中包含数组元素、位置和激励的列表;否则程序输出错误消息。本次实验中共用到 space 程序的 1 个正确版本和 38 个错误版本。

1.1.2 实验目的

这个实验基于 SIR 项目中的 space 程序,目的是验证和评估软件测试过程中的覆盖率分析技术。通过这个实验,我们需要了解并掌握以下几点:

(1) 评估测试覆盖率:通过使用 Gcov 工具编译源代码并生成可执行程序,可以收集程序执行过程中的覆盖率数据,这包括哪些代码行被执行,哪些没有被执行,以及执行的次数等信息;

(2) 发现潜在缺陷：通过运行所有测试用例并使用 Gcov 收集覆盖率信息，可以分析软件中未被测试用例覆盖到的代码区域，这些区域可能隐藏着未被发现的缺陷；

(3) 比较正确与错误版本的差异：通过比较含有缺陷的版本与无缺陷的正确版本的输出结果，可以确定测试用例是否能够准确地检测出软件中的错误，这对于评估测试用例的有效性和完整性至关重要；

(4) 自动化测试流程：编写 Shell 脚本来自动执行测试用例，可以提高测试过程的效率和可重复性，自动化脚本可以确保每次测试都按照相同的步骤和条件进行，从而获得一致和可靠的测试结果；

(5) 提高软件质量：通过这个实验，我们可以更好地理解软件测试过程中覆盖率的重要性，并探索如何利用覆盖率信息来提高软件测试的效果，最终目的是提高软件的质量和可靠性。

总体而言，这个实验旨在通过实际的测试活动，深入理解覆盖率分析在软件测试中的作用，并通过自动化测试流程来提高测试的效率和有效性，从而为软件质量保证提供支持。

1.2 测试环境

1.2.1 硬件环境

表 1 硬件环境

硬件名称	规格型号
处理器	11th Gen Intel(R) Core(TM) i5-11260H @2.60GHz
显卡	NVIDIA GeForce RTX 3060 Laptop GPU
内存	Samsung DDR4 3200MHz 16GB
硬盘	SKHynix_HFS512GDE9X084N 476GB

1.2.2 软件环境

表 2 软件环境

软件名称	版本号	备注说明
VMware	17.5.0	虚拟机软件

Ubuntu	20.04.6	操作系统
gcc	11.4.0	编译器套件
gcov	11.4.0	测试代码覆盖率的工具
lcov	1.14	gcov 的图形化前端工具

1.3 测试团队信息

- (1) 测试内容：软件测试课程实验；
- (2) 测试项目：Software-artifact Infrastructure Repository（简称 SIR 项目）中的 C 语言程序 space；
- (3) 测试地点：软件工程实验室-DS1502；
- (4) 测试时间：2024.3.15-2024.4.8；
- (5) 测试人员：

二、实验原理

2.1 白盒测试

白盒测试 (White-box testing) 是一种软件测试方法，其目的是检查和评估程序内部结构、设计和实现的质量。在白盒测试中，测试人员对被测系统的内部逻辑、代码和数据流进行分析，并基于这些信息设计测试用例。

在白盒测试中，测试人员通常具有对被测系统的源代码和内部结构的访问权限。这使他们能够深入了解系统的内部工作原理，并根据代码路径、条件语句、循环结构等设计测试用例。白盒测试旨在覆盖不同的路径和分支，以验证程序在不同情况下的行为是否符合预期。

白盒测试的主要目标包括：

(1) 确保代码的逻辑正确性：通过测试每个代码路径和分支，以检查程序是否按照预期进行计算和操作；

(2) 改进代码质量：通过发现和修复代码中的错误和缺陷，提高代码的可靠性、可维护性和性能；

(3) 最大限度地覆盖代码：通过设计测试用例，尽可能地覆盖代码中的每一行、每一个分支和每一种情况，以提高测试的全面性。

白盒测试技术包括语句覆盖、分支覆盖、路径覆盖、条件覆盖等。这些技术用于评估测试用例对代码的覆盖程度，并帮助测试人员确定是否需要进一步改进测试套件。

2.2 语句覆盖

语句覆盖 (Statement Coverage)，又称为语句执行覆盖，是软件测试中的一种基本覆盖标准，其核心目标是确保程序中的每一条语句至少被执行一次。它是代码覆盖率中的一种形式，用于评估测试用例对程序代码的覆盖程度。

在进行语句覆盖时，测试人员会设计测试用例，使得程序中的每个可执行语句至少执行一次。这样可以帮助发现那些未被执行到的代码中可能存在的缺陷。例如，如果一个条件分支中的所有路径都没有被测试到，那么这些路径中可能存在的逻辑错误就无法被发现。

计算语句覆盖率的方式是通过统计已执行的语句数量，然后除以总语句数，最后乘以 100 来得到百分比。计算语句覆盖率的公式如下：

$$\text{语句覆盖率} = (\text{已执行的语句数} / \text{总语句数}) * 100\%$$

语句覆盖的实现通常依赖于特定的测试工具，如 Gcov 或其他覆盖率分析工具，这些工具可以在程序运行时追踪代码的执行情况，并生成覆盖率报告。报告中会显示每个语句是否被执行，以及执行的次数，从而帮助测试人员判断测试用例的充分性和有效性。

需要注意的是，尽管语句覆盖是一个重要的测试目标，但它并不能保证找到所有的缺陷。例如，即使所有语句都被执行了，仍然可能存在逻辑错误、边界条件错误或者并发问题等。因此，在实际的软件测试中，通常会结合其他类型的覆盖标准，如分支覆盖、条件覆盖、路径覆盖等，来更全面地评估软件质量。

2.3 分支覆盖

分支覆盖率（Branch Coverage），又称为分支执行覆盖，是软件测试中用于评估测试用例对程序中所有可能的分支路径的覆盖情况的一种度量。在程序中，分支通常是指由条件语句（如 if-else、switch-case 等）产生的两种或多种执行路径。

分支覆盖率的核心目标是确保程序中的每个分支至少被执行一次，这样可以发现潜在的条件逻辑错误。通过设计测试用例来触发每个分支的执行，可以验证程序在不同条件下的行为是否符合预期。

分支覆盖率可以通过以下几种方式来进一步细分：

（1）简单分支覆盖：确保每个分支（如 if 语句中的 true 和 false 路径）至少被执行一次。

（2）复合分支覆盖：对于复杂的分支结构，确保每个可能的分支组合至少被执行一次。例如，在嵌套的 if-else 语句中，需要测试所有可能的 true/false 组合。

（3）条件覆盖：确保每个条件表达式的每个子条件（如 $a > b$ 中的 a 和 b ）都至少取到 true 和 false 两种情况。

（4）路径覆盖：这是分支覆盖的一个更广泛的形式，它要求测试用例覆盖程序中所有可能的执行路径。

计算分支覆盖率的方式是通过统计已执行的分支数量，然后除以总分支数，最后乘以 100 来得到百分比。计算分支覆盖率的公式如下：

分支覆盖率 = (已执行的分支数 / 总分支数) * 100%

这个比例值表示了测试用例对程序中所有分支的覆盖程度。理想情况下，分支覆盖率应该尽可能接近 100%，但实际上，由于分支数量可能非常庞大，达到完全的分支覆盖可能是非常困难的。需要注意的是，即使分支覆盖率达到 100%，也不能保证找到所有的缺陷，因为有些缺陷可能只有在特定的分支组合下才会显现。此外，分支覆盖率也不考虑循环和递归等复杂结构中的覆盖情况。因此，在实际的软件测试中，通常会结合其他类型的覆盖标准和测试方法，以更全面地评估软件质量。

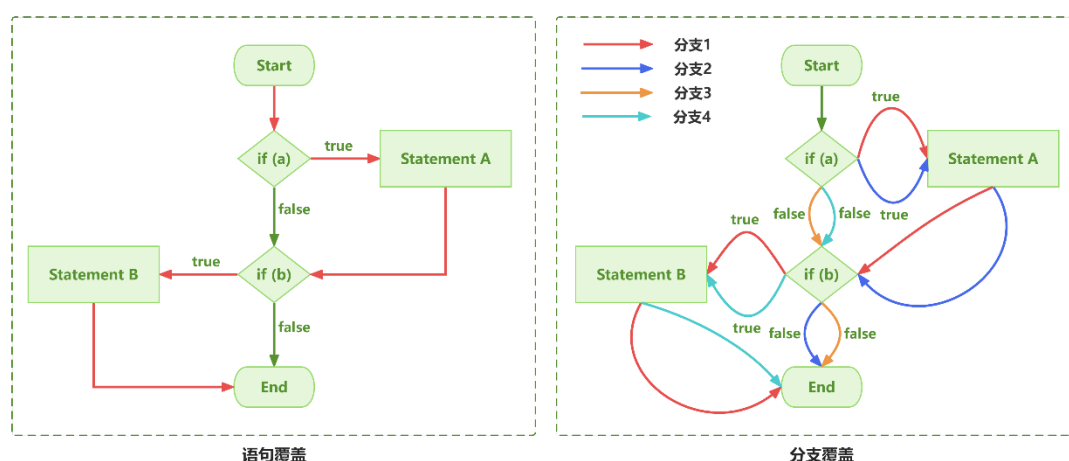


图 1 语句覆盖与分支覆盖

2.4 函数覆盖

函数覆盖 (Function Coverage) 是软件测试中的一种覆盖标准，它关注的是程序中各个函数的执行情况。函数覆盖的目标是确保在测试过程中，程序中的每个函数至少被调用一次。这种覆盖标准的目的是检测函数内部的代码质量，包括函数的实现逻辑和可能存在的内部错误。

函数覆盖的主要特点和目的包括：

- (1) 函数调用：确保测试用例能够触发程序中的每个函数至少一次调用，这样可以检查函数的入口和出口行为，以及函数内部的逻辑；
- (2) 局部逻辑：通过函数覆盖，可以发现函数内部的潜在问题，例如错误的计算、不正确的返回值、异常处理不当等；
- (3) 模块化测试：函数覆盖支持模块化测试，允许测试人员独立地测试程序的各个模块或组件，而不需要关心其他模块的实现细节；
- (4) 辅助其他覆盖标准：函数覆盖通常与其他覆盖标准（如语句覆盖、分支覆盖等）结合使用，以提供更全面的测试覆盖。

计算函数覆盖率的方式是通过统计已执行的函数数量，然后除以总函数数，最后乘以 100 来得到百分比。计算函数覆盖率的公式如下：

$$\text{函数覆盖率} = (\text{已执行的函数数} / \text{总函数数}) * 100\%$$

这个比例值表示了测试用例对程序中所有函数的覆盖程度。理想情况下，函数覆盖率应该达到 100%，即所有的函数都被至少调用了一次。然而，在实际的软件测试中，可能会因为某些函数难以被直接调用或者没有合适的测试输入而无法达到完全的覆盖。因此，测试人员需要分析未被覆盖的函数，并考虑是否需要设计额外的测试用例或测试策略来提高覆盖率。

需要注意的是，尽管函数覆盖是一个重要的测试目标，但它并不能保证找到所有的缺陷。例如，即使所有函数都被调用了，也可能存在逻辑错误或边界条件问题。因此，在进行软件测试时，通常需要结合其他类型的测试和覆盖标准，以更全面地评估软件质量。

2.5 实验流程

2.5.1 整体概述

本次实验测试流程主要由五部分组成：

- 1、对 space 文件中的程序版本进行修正；
- 2、使用 gcc 对正确版本以及错误版本源代码进行编译；
- 3、选择测试组进行测试；
- 4、使用 gcov 工具得到测试代码的覆盖率，使用 lcov 和 genhtml 对覆盖率结果进行可视化，得到覆盖率报告；
- 5、对比程序正确版本与不同错误版本的输出结果，判断错误版本是否通过测试。

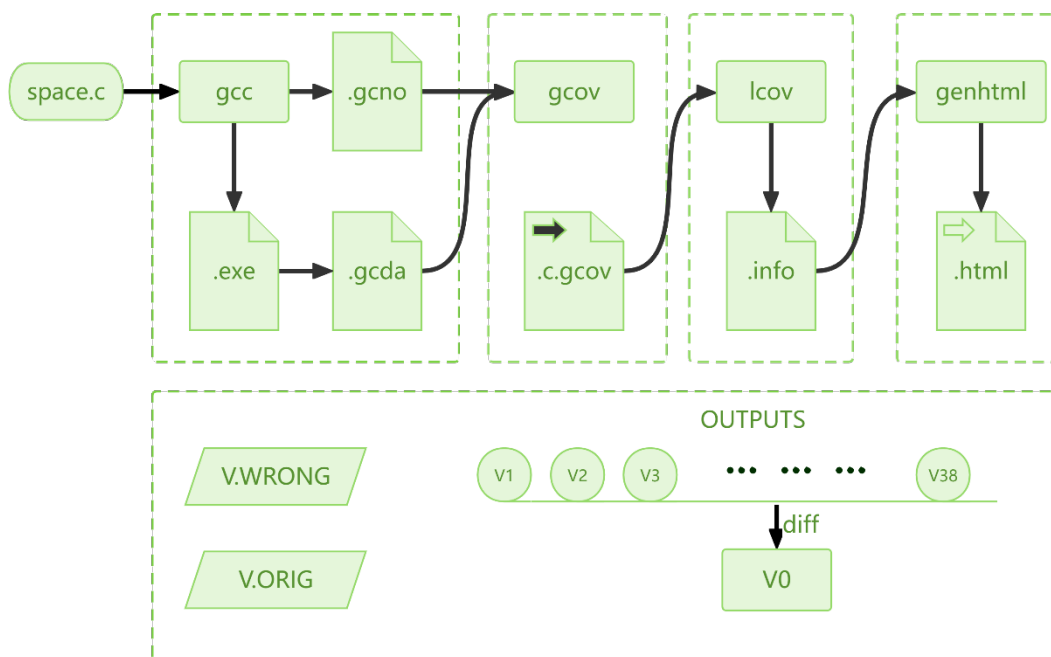


图 2 整体流程图

2.5.2 源代码修正

space 程序源代码无法通过 gcc 编译，经检查发现原因是 gcc 已经不支持 gets() 函数。因此，我们编写了一个 revise.sh 脚本将所有版本源代码中的 gets() 函数替换为 fgets() 函数，使之可以被 gcc 正确编译，实现效果如下所示。

修改前：

```

1367     printf("\n%s (Y/N): ", domanda);
1368     gets(s);
1369     if ((s[0] == 'N') || (s[0] == 'n'))

```

图 3 space 源代码修改前

修改后：

```

1367     printf("\n%s (Y/N): ", domanda);
1368     fgets(s,2,stdin);
1369     if ((s[0] == 'N') || (s[0] == 'n'))

```

图 4 space 源代码修改后

2.5.3 GCC 编译

GCC (GNU Compiler Collection) 是一个功能强大的编译器套件，由 GNU 项目开发，广泛用于多种编程语言的编译工作，包括但不限于 C、C++、Objective-C、Fortran、Ada 和 Go 等。作为一个开源项目，GCC 遵循 GNU 通用公共许可证 (GPL)，允许用户自由使用、修改和分发。它支持跨平台编译，能够在包括 Linux、macOS 和 Windows 在内的多种操作系统上运行，这使得 GCC 成为全球开发者广泛采用的编译工具之一。

GCC 以其出色的优化能力和对编程语言标准的遵循而闻名，能够生成高效的目标代码，并且提供了丰富的调试和剖面分析功能。此外，GCC 的可扩展性设计使得开发者可以为其添加新的语言支持和架构，确保了其长期的适应性和生命力。作为 GNU 工具链的核心组件，GCC 与其他工具如 GDB 调试器、Binutils 二进制工具集等紧密协作，为用户提供了一整套软件开发工具。

得益于其活跃的开发社区，GCC 持续得到改进和更新，以适应不断演进的软件和硬件技术。无论是在嵌入式系统开发、操作系统内核编译，还是日常的应用程序开发中，GCC 都证明了其作为一个可靠、高效和灵活的编译器的实力。因此，GCC 不仅是开源运动的一个重要成果，也是全球软件开发领域的一个重要支柱。

本实验中，将待测版本中的源代码和头文件复制到 source 文件夹下，我们使用 gcc 工具对程序的源代码进行编译，在编译前加入 GCC 编译器的多个选项：

```
gcc -fprofile-arcs -ftest-coverage -o space space.c -w -lm
```

表 3 GCC 编译器添加选项

-fprofile-arcs -ftest-coverage	用于生成 .gcda 和 .gcov 文件，编译结束之后，gcov 会结合 .gcon .gcda 和源文件产生 .c.gcov 代码覆盖率统计文件。
-o	用于输出编译得到的可执行文件。
-w	用于关闭编译器产生的警告信息。
-lm	用于指示编译器链接数学库。通常，数学库包含了一系列数学函数的实现，如三角函数、对数函数、指数函数等。在使用这些数学函数时，需要链接数学库以使得程序能够正确编译和执行。

2.5.4 选择测试组

在 inputs 文件中，space 程序共提供了 13526 个测试用例。

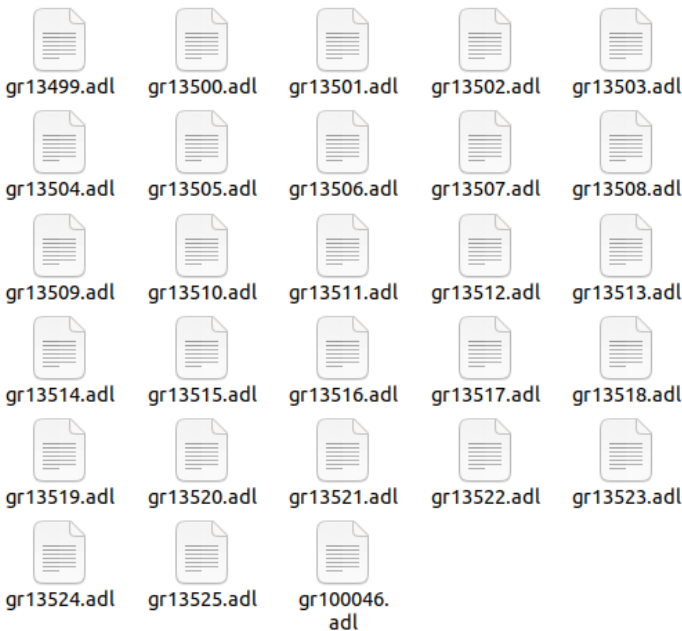


图 5 测试用例总览

而 Testplans.alt 文件夹中的 suite 文件按照特定顺序组合了测试用例，使之成为一个测试集。

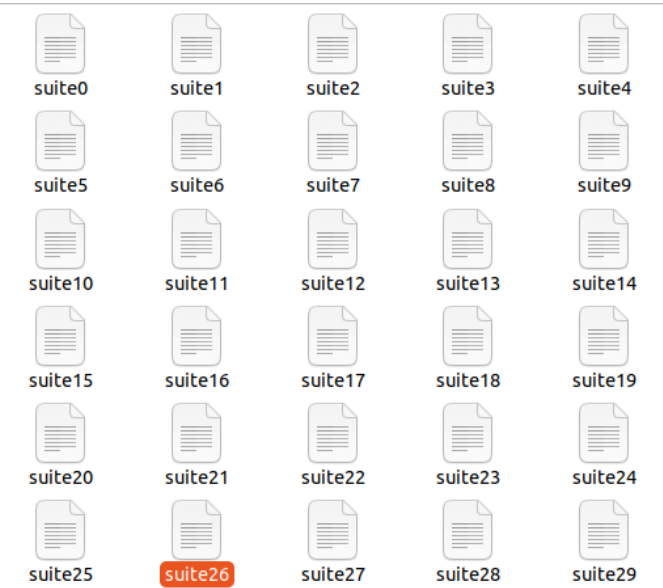


图 6 测试集总览

我们从中随机选取一个测试集（以 suite0 为例），分别用正确版本和 38 个错误版本进行测试。

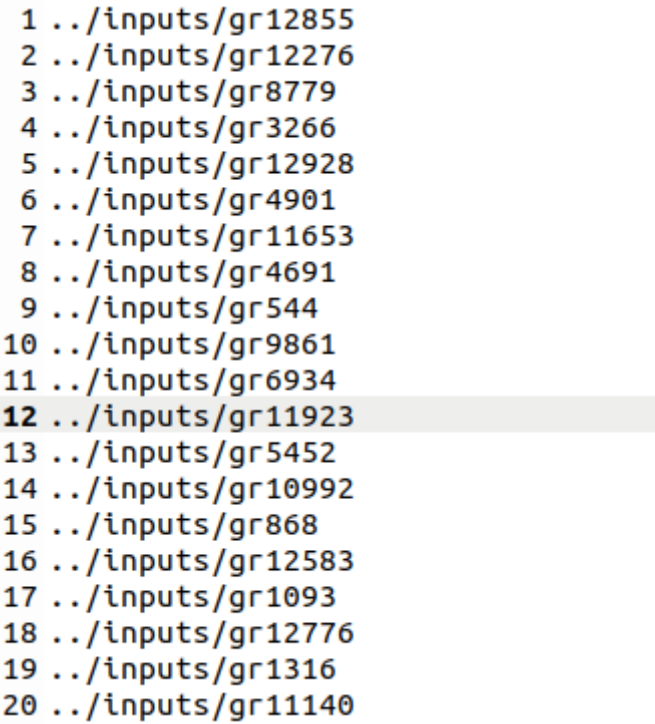


图 7 测试集示意图

2.5.5 生成覆盖率报告

使用 `gcov` 工具生成代码覆盖率统计文件：

```
gcov space.c
```

使用 `lcov` 工具生成包含语句覆盖、函数覆盖、分支覆盖率的 `.info` 文件：

```
lcov --rc lcov_branch_coverage=1 -c -d . -o $out2/v0.info
```

表 4 `lcov` 命令解析

<code>--rc lcov_branch_coverage=1</code>	收集分支覆盖率。
<code>-c</code>	告诉 <code>lcov</code> 创建一个新的覆盖率数据集。
<code>-d</code>	指定 <code>lcov</code> 应该递归地搜索当前目录下的所有 <code>.gcda</code> 和 <code>.gcno</code> 文件，这些

	文件包含了由 gcov 生成的覆盖率数据。
-o	指定输出文件的路径和生成的覆盖率数据文件名。

最后利用 generate_html 函数在指定目录下运行：

```
genhtml $out2/v0.info --branch-coverage --function-coverage --
output-directory $out2/html > $out2/outputs
```

以此来读取 lcov 生成的.info 文件并生成可视化的 html 文件，将 html 文件以及其他输出结果存放再 outputs 文件夹下。

表 5 Genhtml 命令解析

genhtml	表示调用的程序名，它会根据给定的输入文件生成 HTML 格式的覆盖率报告。
\$out2/v0.info	表示 genhtml 工具的输入文件路径，其中包含了由 GCC 在编译时生成的覆盖率数据信息。\$out2 是一个变量，通常在构建系统中使用，代表输出目录的路径，v0.info 是存储覆盖率数据的文件。
--branch-coverage	指示 genhtml 包括分支覆盖率信息。
--function-coverage	指示 genhtml 包括函数覆盖率信息。
--output-directory \$out2/html	指定了输出目录的路径，genhtml 将在此目录下生成 HTML 格式的覆盖率报告。
> \$out2/outputs	重定向操作，它将 genhtml 命令的标准输出（stdout）重定向到一个文件中，文件路径由变量\$out2 和 outputs 指定。这意味着任何由 genhtml 打印到控制台的输出都会被写入到这个文件中。

2.5.6 Diff 命令比较输出结果

Diff 命令是 Unix 和类 Unix 操作系统中的一个标准工具，用于比较两个文件或文本流之间的差异。这个命令的核心功能是输出两个输入源之间的差异，通常用于检查代码变更、文档修改或其他任何文本文件的更新。

我们将不同版本的程序的输出结果存放在 newoutputs 文件夹对应版本号文件夹里，使用 diff 命令，将错误版本的每个测试用例的输出结果与正确版本的输出结果进行比较，如果相同，则认为测试用例通过，如果存在不同则认为测试用例不通过，得到错误版本的测试用例通过率。最后将所有错误版本的测试通过率统计存放到 compareInfo 文件中。

表 6 常用 diff 命令解析

-c 或 --context	输出上下文格式的差异，显示行号和周围的未改变的行，以及具体的变更行。
-u 或 -U 或 --unified	输出统一格式的差异，这是默认的输出格式，它只显示行号和变更行。
-b 或 --ignore-space-change	忽略空格和制表符的差异，只关注实际的文本差异。
-i 或 --ignore-case	忽略大小写的差异，将大写和小写视为相同。
-r 或 --recursive	递归地比较目录中的所有文件。
--brief 或 -t	只输出哪些文件有差异，而不显示具体的差异内容。
-e 或 --enhanced	输出增强的统一格式差异，包含更多信息，便于其他工具解析。
-w 或 --word-diff	按单词级别显示差异，而不是按行。

三、实验过程记录

3.1 环境配置

小组成员预先在自己电脑上下载虚拟机软件 VMware (Version 17.5.0)，在其中安装好操作系统 Ubuntu (Version 20.04.6)，再准备好软件测试所需要工具，包括测试对象软件 space (Version 1)、编译器套件 gcc (Version 11.4.0)、测试代码覆盖率工具 gcov (Version 11.4.0)、gcov 的图形化前端工具 lcov (Version 1.14) 等。

3.2 前期准备

3.2.1 源代码修正脚本

在 source 文件夹下编写 revise.sh 脚本，读取源代码文件，将 gets() 函数替换为 fgets() 函数。



```
1 #修改版本文件中的错误
2 #修改正确版本
3 sed -i 's/gets(s)/fgets(s,2,stdin)/g' ../source.alt/source.orig/space.c
4
5 #修改错误版本
6 num=1
7 while [ $num -lt 39 ]
8 do
9     filename=../versions.alt/versions.orig/v${num}/space.c
10    sed -i 's/gets(s)/fgets(s,2,stdin)/g' $filename
11    ((num++))
12 done
13
14 echo "正确版本和38个错误版本修改完成"
```

图 8 源代码修正脚本

3.2.2 准备源代码

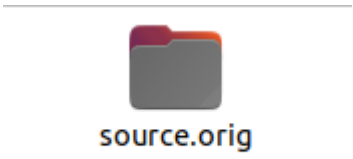


图 9 正确版本



图 10 错误版本（共 38 个）

3.2.3 选择测试集

inputs 文件夹中包含共 13526 个测试用例。

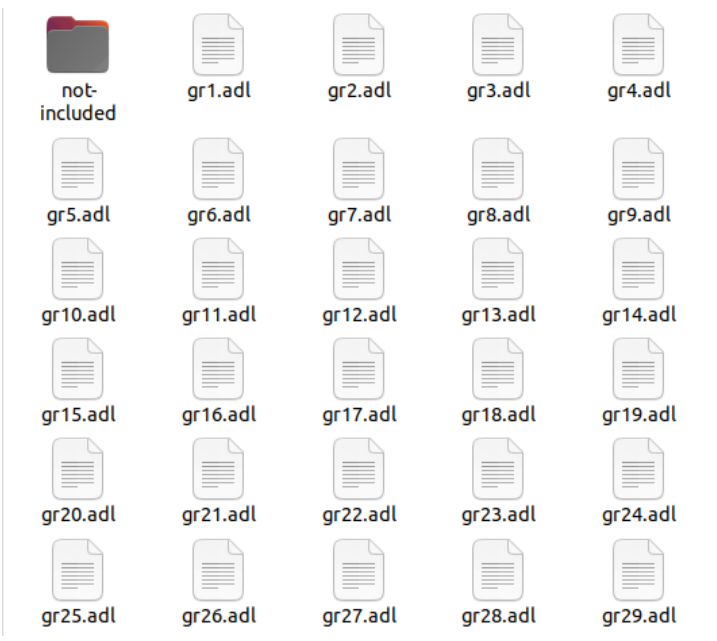


图 11 测试用例集合

testplans.alt/testplans.cov 中包含 1000 个测试集。

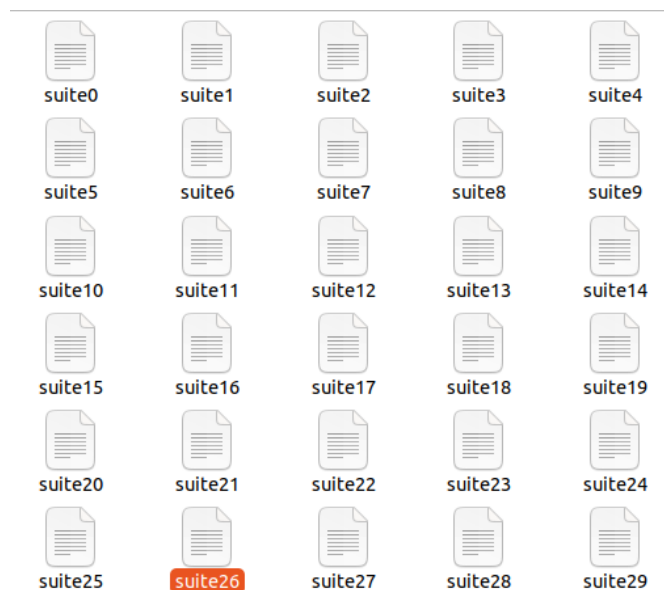


图 12 测试集总览

每一个测试集包含大约 150 个测试用例的路径，以 suite0 为例。

```
1 ../inputs/gr12855
2 ../inputs/gr12276
3 ../inputs/gr8779
4 ../inputs/gr3266
5 ../inputs/gr12928
6 ../inputs/gr4901
7 ../inputs/gr11653
8 ../inputs/gr4691
9 ../inputs/gr544
10 ../inputs/gr9861
11 ../inputs/gr6934
12 ../inputs/gr11923
13 ../inputs/gr5452
14 ../inputs/gr10992
15 ../inputs/gr868
16 ../inputs/gr12583
17 ../inputs/gr1093
18 ../inputs/gr12776
19 ../inputs/gr1316
20 ../inputs/gr11140
```

图 13 测试集示意图

选择 suite0 作为第一次测试的测试集。

```
#选择测试组
suite=" ../testplans.alt/testplans.cov/suite0"
total_lines=$(wc -l < "$suite") #测试组中输入的行数+1
```

图 14 选择测试集

3.3 编写测试脚本 tScript.sh

3.3.1 对正确版本进行测试

将正确版本的源代码复制到 source 文件夹中，使用 gcc 工具将 space.c 文件编译为可执行文件 space；创建输出文件夹../outputs/v0（用于存放覆盖率报告）和../newoutputs/v0（用于存放程序输出结果）；使用 space 可执行文件运行每一个输入文件（测试组的每一行或者说每一个测试用例）并用 gcov 统计代码覆盖率；使用 lcov 和 genhtml 将代码覆盖率可视化，生成覆盖率报告。最后将 source 中的源代码以及相关文件删除。具体如下：

(1) echo "0"：这一行输出字符串"0"，可能用于初始化或测试某个过程。

(2) cp ../source.alt/source.orig/* ./：将../source.alt/source.orig/目录下的所有文件复制到当前目录下。这可能是为了将原始的源代码文件复制到测试环境中。

(3) gcc -fprofile-arcs -ftest-coverage -o space space.c -w -lm：使用 GCC 编译器编译 space.c 文件，生成可执行文件 space。这里使用了-fprofile-arcs 和-ftest-coverage 选项来生成用于代码覆盖率分析的额外信息。-w 选项关闭了所有警告信息，-lm 链接了数学库。

(4) 接下来的两段代码创建了两个目录\$out1 和\$out2，分别用于存放测试输出和覆盖率报告。

(5) current_line=1：初始化变量 current_line，用于记录当前读取的行号。

(6) while IFS= read -r line; do ... done < "\$suite"：这是一个循环，它从文件\$suite 中逐行读取输入。IFS=保证了行的首尾不会有额外的空白字符，-r 选项防止反斜杠转义字符被解释。

(7) if [\$current_line -lt \$total_lines]; then ... fi：这个条件判断当前行号是否小于总行数\$total_lines。如果是，执行以下命令：

- ./space \$line > \$out1/\$current_line: 运行编译后的程序 space, 并将输入 \$line 传递给程序, 输出结果到 \$out1 目录下对应的文件中。

① gcov -b space.c > /dev/null 2>&1: 使用 gcov 工具分析 space.c 的覆盖率信息, 并将输出重定向到 /dev/null, 这样就不会在终端显示任何输出。

② ((current_line++)): 增加 current_line 的值, 以便在下一次循环中读取下一行。

(8) lcov --rc lcov_branch_coverage=1 --capture --directory . --output-file \$out2/v0.info > /dev/null 2>&1: 使用 lcov 工具收集当前目录下的覆盖率数据, 并输出到 \$out2/v0.info 文件中。--rc lcov_branch_coverage=1 启用分支覆盖率分析。输出同样被重定向到 /dev/null。

(9) genhtml \$out2/v0.info --branch-coverage --function-coverage --output-directory \$out2/html > \$out2/outputs: 将 \$out2/v0.info 文件转换成 HTML 格式的覆盖率报告, 并存放到 \$out2/html 目录下。

(10) rm space.c strutt.h space space.gcno space.gcda space.c.gcov: 最后, 删除所有生成的文件, 包括源代码文件、可执行文件和覆盖率分析文件, 清理测试环境。

```
8 #对正确版本进行测试
9     echo "0"
10    #将测试版本复制到source文件
11    cp ../source.alt/source.orig/* ./
12    #对被测版本源文件进行编译
13    gcc -fprofile-arcs -ftest-coverage -o space space.c -w -lm
14
15    #输出结果路径
16    out1="../newoutputs/r0"
17    if [ ! -d "$out1" ]; then
18        mkdir $out1
19    fi
20
21    #覆盖率报告输出路径
22    out2="../outputs/v0"
23    if [ ! -d "$out2" ]; then
24        mkdir $out2
25    fi
26
27    current_line=1
28    #读取文件每一行的输入
29    while IFS= read -r line;
30    do
31        if [ $current_line -lt $total_lines ]
32        then
33            ./space $line > $out1/$current_line
34            gcov -b space.c > /dev/null 2>&1
35            ((current_line++))
36        fi
37    done < "$suite"
38
39    #将结果输入到v0
40
41    lcov --rc lcov_branch_coverage=1 --capture --directory . --output-file $out2/v0.info > /dev/null 2>
42    genhtml $out2/v0.info --branch-coverage --function-coverage --output-directory $out2/html > $out2/outp
43
44
45    #运行程序 更新gcda文件
46    #输出结果 outputs html v0.info
47    # /dev/null 2>&1 丢弃输出信息
48
49    #从source中移除相关文件
50    rm space.c strutt.h space space.gcno space.gcda space.c.gcov
51
52 #对错误版本进行测试
```

图 15 正确版本测试脚本

3.3.2 对错误版本进行测试

对 version.alt 中的每一个错误版本 vi，将其的源代码复制到 source 文件夹中，使用 gcc 工具将 space.c 文件编译为可执行文件 space；创建输出文件夹../outputs/vi（用于存放覆盖率报告）和../newoutputs/vi（用于存放程序输出结果）；使用 space 可执行文件运行每一个输入文件（测试组的每一行或者说每一个测试用例）并用 gcov 统计代码覆盖率；使用 lcov 和 genhtml 将代码覆盖率可视化，生成覆盖率报告。最后将 source 中的源代码以及相关文件删除。具体如下：

(1) `Vnum=`find ../versions.alt/versions.orig/ -mindepth 1 -type d | wc -l``：这行代码使用 find 命令查找../versions.alt/versions.orig/目录下的所有子目录（错误版本），并通过 wc -l 计算它们的数量。这个数量被存储在变量 Vnum 中。

(2) `for ((i=1; i<=Vnum; i++))`：这是一个循环，从 1 开始，直到 Vnum 的值，用于依次处理每个错误版本。

(3) `echo "$i"`：在每次循环中，打印当前处理的版本号。

(4) `cp ../versions.alt/versions.orig/v$i/* ./`：将第 i 个错误版本的所有文件复制到当前目录下，以便进行编译和测试。

(5) `gcc -fprofile-arcs -ftest-coverage -o space space.c -w -lm`：使用 GCC 编译器编译 space.c 文件，生成可执行文件 space。这里使用了 -fprofile-arcs 和 -ftest-coverage 选项来生成用于代码覆盖率分析的额外信息。-w 选项关闭了所有警告信息，-lm 链接了数学库。

(6) 接下来的两段代码创建了两个目录\$out1 和\$out2，分别用于存放测试输出和覆盖率报告。这些目录的名称包含了版本号\$i，以区分不同版本的输出结果。

(7) `current_line=1`：初始化变量 current_line，用于记录当前读取的行号。

(8) `while IFS= read -r line; do ... done < "$suite"`：这是一个循环，它从文件\$suite 中逐行读取输入。IFS=保证了行的首尾不会有额外的空白字符，-r 选项防止反斜杠转义字符被解释。

(9) `if [$current_line -lt $total_lines]; then ... fi`：这个条件判断当前行号是否小于总行数\$total_lines。如果是，执行以下命令：

① `./space $line > $out1/$current_line`：运行编译后的程序 space，并将输入\$line 传递给程序，输出结果到\$out1 目录下对应的文件中。

②`gcov -b space.c > /dev/null 2>&1`: 使用 `gcov` 工具分析 `space.c` 的覆盖率信息, 并将输出重定向到 `/dev/null`, 这样就不会在终端显示任何输出。

③`((current_line++))`: 增加 `current_line` 的值, 以便在下一次循环中读取下一行。

(10) `lcov --rc lcov_branch_coverage=1 --capture --directory . --output-file $out2/v$i.info`: 使用 `lcov` 工具收集当前目录下的覆盖率数据, 并输出到 `$out2/v$i.info` 文件中。`--rc lcov_branch_coverage=1` 启用分支覆盖率分析。输出被重定向到 `/dev/null`。

(11) `genhtml $out2/v$i.info --branch-coverage --function-coverage --output-directory $out2/html`: 将 `$out2/v$i.info` 文件转换成 HTML 格式的覆盖率报告, 并存放到 `$out2/html` 目录下。

(12) `rm space.c strutt.h space space.gcno space.gcda space.c.gcov`: 在每次循环结束时, 删除所有生成的文件, 包括源代码文件、可执行文件和覆盖率分析文件, 清理测试环境。

```
52 #对错误版本进行测试
53 #错误版本个数
54 Vnum=`find ../versions.alt/versions.orig/ -mindepth 1 -type d | wc -l`
55 #Vnum=2
56 for ((i=1;i<=Vnum;i++))
57 do
58     echo "$i"
59     #将测试版本复制到source文件
60     cp ../versions.alt/versions.orig/v$i/* ./
61     #对被测版本源文件进行编译
62     gcc -fprofile-arcs -ftest-coverage -o space space.c -w -lm
63
64     #输出结果路径
65     out1=../newoutputs/r$i
66     if [ ! -d "$out1" ]; then
67         mkdir $out1
68     fi
69
70     #覆盖率报告输出路径
71     out2=../outputs/v$i
72     if [ ! -d "$out2" ]; then
73         mkdir $out2
74     fi
75
76     current_line=1
77     #读取文件每一行的输入
78     while IFS= read -r line;
79     do
80         if [ $current_line -lt $total_lines ]
81         then
82             ./space $line > $out1/$current_line
83             gcov -b space.c > /dev/null 2>&1
84             ((current_line++))
85         fi
86     done < "$suite"
87
88     #将结果输入到vi
89     #lcov --rc lcov_branch_coverage=1 -c -d . -o $out2/v$i.info > /dev/null 2>&1
90     #genhtml -o $out2/html $out2/v$i.info > $out2/outputs
91     lcov --rc lcov_branch_coverage=1 --capture --directory . --output-file $out2/v$i.info > /dev/null 2>&1
92     genhtml $out2/v$i.info --branch-coverage --function-coverage --output-directory $out2/html > /dev/null 2>&1
93
94     #删除测试版本
95     rm space.c strutt.h space space.gcno space.gcda space.c.gcov
```

图 16 错误版本测试脚本

3.3.3 比较输出结果

程序输出结果保存在../newinputs/vi（对于版本编号的文件夹下，正确版本为v0）。将每一个错误版本的输出结果与正确版本的输出结果进行比较。计算测试通过率，将结果输出到compareInfo文件中。

```
3 #将正确版本的输出与错误版本的输出进行比较
3 echo -n "" > compareInfo
3 output=$((total_lines - 1))      #输出个数
1 for((i=1;i<=Vnum;i++))
2 do
3     wrongNum=`diff -rq ../newoutputs/r0 ../newoutputs/r$i | grep -c '^文件.*不同$'`
4     #通过率
5     printf "%4s %3s/$output\n" "v$i:" $((output-wrongNum)) >> compareInfo
5 done
7
```

图 17 比较输出结果脚本

3.4 编写代码覆盖率统计脚本 getCovInfo.sh

编写 getCovInfo.sh 脚本，统计所有版本的代码覆盖率，存放在 CovInfo 文件中。

```
num=`find ../outputs/ -maxdepth 1 -mindepth 1 -type d | wc -l`
echo -n "" > CovInfo
for ((i=0;i<num;i++))
do
    echo v$i:\n >> CovInfo
    tail -n 3 ../outputs/v$i/outputs >> CovInfo
done
```

图 18 代码覆盖率统计脚本

```
1 v0:
2   lines.....: 91.9% (3403 of 3701 lines)
3   functions...: 91.9% (125 of 136 functions)
4   branches...: 85.1% (1013 of 1190 branches)
5 v1:
6   lines.....: 91.9% (3403 of 3701 lines)
7   functions...: 91.9% (125 of 136 functions)
8   branches...: 85.1% (1013 of 1190 branches)
9 v2:
10  lines.....: 91.9% (3403 of 3701 lines)
11  functions...: 91.9% (125 of 136 functions)
12  branches...: 85.1% (1013 of 1190 branches)
13 v3:
14  lines.....: 91.9% (3403 of 3701 lines)
15  functions...: 91.9% (125 of 136 functions)
16  branches...: 85.1% (1013 of 1190 branches)
```

图 19 CovInfo 文件内容示意图

3.5 测试过程

3.5.1 运行脚本

我们事先在 `tScript.sh` 脚本文件中调用 `revise.sh` 脚本文件和 `getCovInfo.sh` 脚本文件（均保存在同一路径下），这样我们只需要运行 `tScript.sh` 脚本文件即可完成本次测试。

```
#修改源代码中的错误
/bin/bash revise.sh

#汇总覆盖率信息
/bin/bash getCovInfo.sh
```

图 20 脚本调用

在终端进入到 `source` 文件夹路径下，运行该脚本文件。

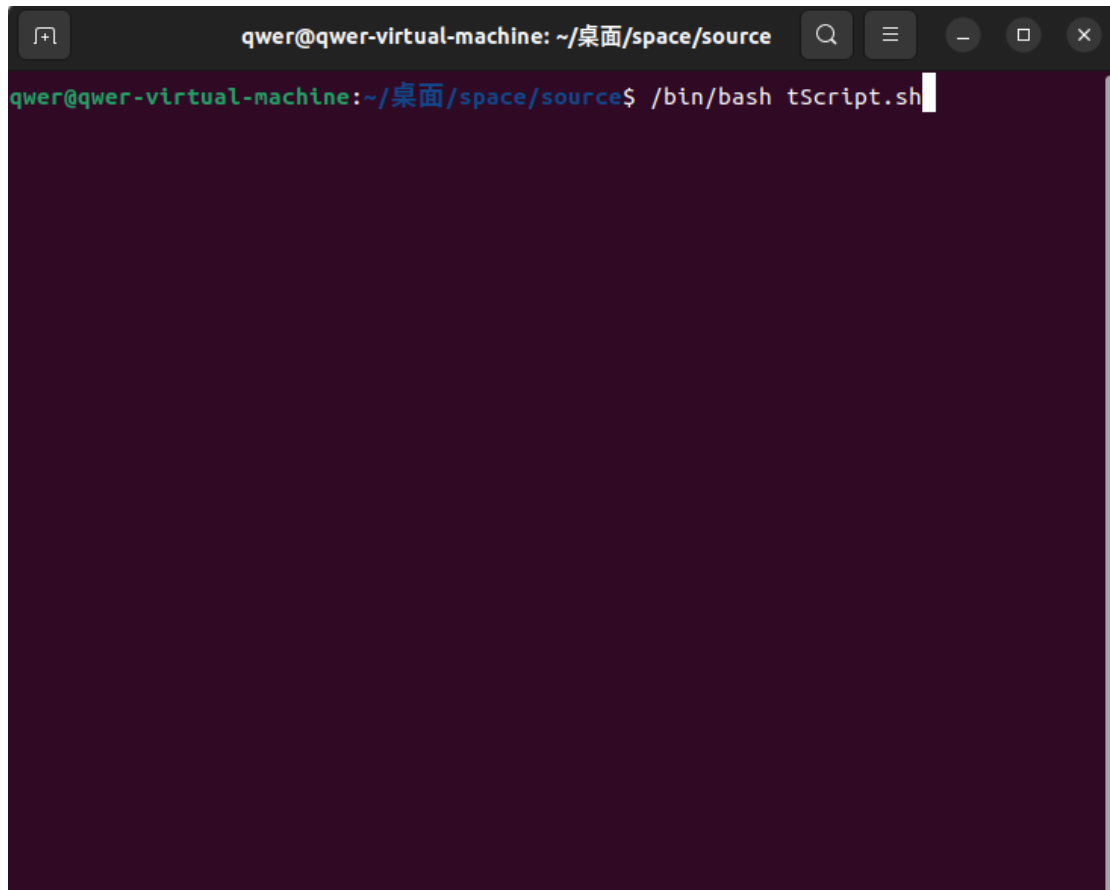


图 21 运行脚本 tScript.sh

3.5.2 运行过程

脚本文件首先对源代码进行修正处理，并打印出“正确版本和 38 个错误版本修改已完成”的信息，然后开始对正确版本和错误版本分别进行测试，测试途中按照执行顺序依次打印出对应的版本号（从 0 到 38，其中 0 代表正确版本）。

```
qwer@qwer-virtual-machine: ~/桌面/space/source
qwer@qwer-virtual-machine:~/桌面/space/source$ /bin/bash tScript.sh
正确版本和38个错误版本修改完成
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

图 22 脚本正常运行过程

```

32
33
34
35
tScript.sh: 第 96 行: 43612 段错误          (核心已转储) ./space $line >
  $out1/$current_line
tScript.sh: 第 96 行: 43645 段错误          (核心已转储) ./space $line >
  $out1/$current_line
tScript.sh: 第 96 行: 43714 段错误          (核心已转储) ./space $line >
  $out1/$current_line
tScript.sh: 第 96 行: 43805 段错误          (核心已转储) ./space $line >
  $out1/$current_line
36
tScript.sh: 第 96 行: 44097 段错误          (核心已转储) ./space $line >
  $out1/$current_line
tScript.sh: 第 96 行: 44114 段错误          (核心已转储) ./space $line >
  $out1/$current_line
tScript.sh: 第 96 行: 44153 段错误          (核心已转储) ./space $line >
  $out1/$current_line
37
38
tScript.sh: 第 96 行: 44799 段错误          (核心已转储) ./space $line >
  $out1/$current_line
qwer@qwer-virtual-machine:~/桌面/space/sources

```

图 23 测试用例使程序无法正常运行时

3.5.3 运行结束

对于每个测试用例，我们运行程序并使用 gcov 生成覆盖率数据，包括语句覆盖率、函数覆盖率和分支覆盖率等信息。脚本文件运行结束后，得到覆盖率信息，有了整合后的覆盖率数据文件，就可以使用 genhtml 工具生成一个可视化的 HTML 报告。这个报告将包含语句覆盖率和分支覆盖率，以及每个文件和整个项目的覆盖率概览。

总体文字概览存放在 source/CovInfo 文件中；

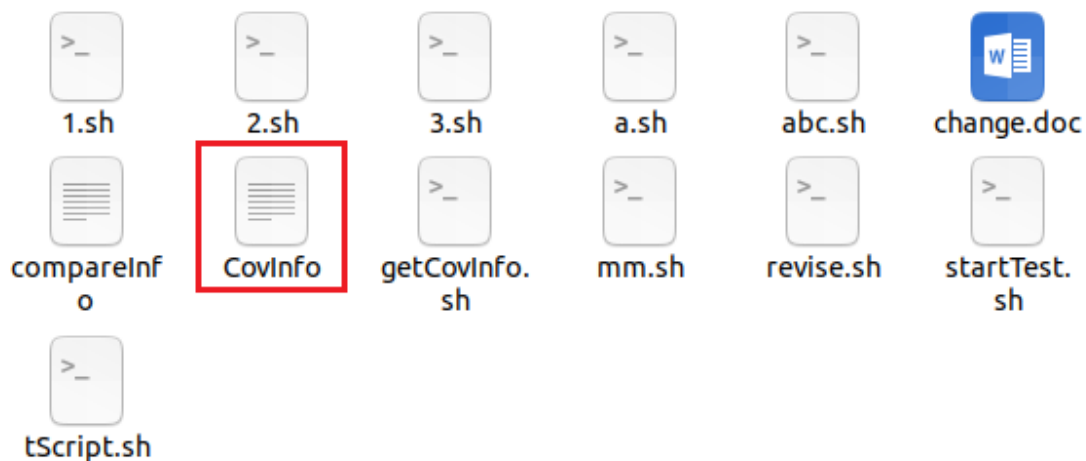


图 24 覆盖率信息汇总文件

各个版本的覆盖率信息及可视化 HTML 界面存放在 outputs 文件夹中；



图 25 各版本覆盖率信息

记录每个测试用例的输出成果并用以对比的结果存放在 newoutputs 文件夹中；



图 26 供比对的程序输出结果

各版本程序结果比对结果存放在 space/compareInfo 文件中。

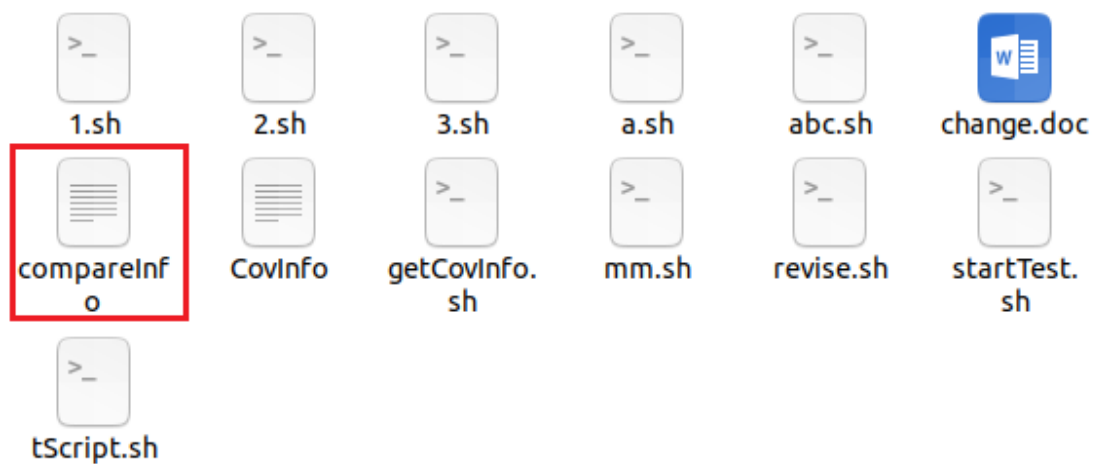


图 27 对比结果文件

四、结果及分析

4.1 覆盖率结果及分析

4.1.1 覆盖率结果

本次实验中，存在大量的测试用例，所幸官方提供了可使用的测试集索引，每个测试集包含约 150 个测试用例，在其中我们选取 suite0 作为测试集进行测试：（下为 suite0 部分内容）

```
1 |../inputs/gr12855
2 |../inputs/gr12276
3 |../inputs/gr8779
4 |../inputs/gr3266
5 |../inputs/gr12928
6 |../inputs/gr4901
7 |../inputs/gr11653
8 |../inputs/gr4691
9 |../inputs/gr544
10 |../inputs/gr9861
11 |../inputs/gr6934
12 |../inputs/gr11923
13 |../inputs/gr5452
14 |../inputs/gr10992
15 |../inputs/gr868
16 |../inputs/gr12583
17 |../inputs/gr1093
18 |../inputs/gr12776
19 |../inputs/gr1316
20 |../inputs/gr11140
21 |../inputs/gr8103
22 |../inputs/gr12961
23 |../inputs/gr6933
24 |../inputs/gr609
25 |../inputs/gr1559
26 |../inputs/gr4588
27 |../inputs/gr3353
28 |../inputs/gr12312
```

图 28 suite0 部分内容

得到语句覆盖率、函数覆盖率及分支覆盖率的结果如下所示，其中 V0 为正确版本。

(1) 覆盖率信息汇总（文字版本）

```

1 v0:
2   lines.....: 91.9% (3403 of 3701 lines)
3   functions...: 91.9% (125 of 136 functions)
4   branches...: 85.1% (1013 of 1190 branches)
5 v1:
6   lines.....: 91.9% (3403 of 3701 lines)
7   functions...: 91.9% (125 of 136 functions)
8   branches...: 85.1% (1013 of 1190 branches)
9 v2:
10  lines.....: 91.9% (3403 of 3701 lines)
11  functions...: 91.9% (125 of 136 functions)
12  branches...: 85.1% (1013 of 1190 branches)
13 v3:
14  lines.....: 91.9% (3403 of 3701 lines)
15  functions...: 91.9% (125 of 136 functions)
16  branches...: 85.1% (1013 of 1190 branches)
17 v4:
18  lines.....: 30.2% (1118 of 3701 lines)
19  functions...: 33.1% (45 of 136 functions)
20  branches...: 24.1% (287 of 1190 branches)
21 v5:
22  lines.....: 88.1% (3260 of 3701 lines)
23  functions...: 91.2% (124 of 136 functions)
24  branches...: 78.5% (934 of 1190 branches)
25 v6:
26  lines.....: 28.4% (1051 of 3701 lines)
27  functions...: 33.1% (45 of 136 functions)
28  branches...: 21.6% (257 of 1190 branches)

29 v7:
30  lines.....: 91.9% (3399 of 3697 lines)
31  functions...: 91.9% (125 of 136 functions)
32  branches...: 85.1% (1009 of 1186 branches)
33 v8:
34  lines.....: 91.9% (3399 of 3697 lines)
35  functions...: 91.9% (125 of 136 functions)
36  branches...: 85.1% (1009 of 1186 branches)
37 v9:
38  lines.....: 91.9% (3403 of 3701 lines)
39  functions...: 91.9% (125 of 136 functions)
40  branches...: 85.1% (1013 of 1190 branches)
41 v10:
42  lines.....: 91.9% (3403 of 3701 lines)
43  functions...: 91.9% (125 of 136 functions)
44  branches...: 85.1% (1013 of 1190 branches)
45 v11:
46  lines.....: 91.9% (3403 of 3701 lines)
47  functions...: 91.9% (125 of 136 functions)
48  branches...: 85.1% (1013 of 1190 branches)
49 v12:
50  lines.....: 91.9% (3403 of 3701 lines)
51  functions...: 91.9% (125 of 136 functions)
52  branches...: 85.1% (1013 of 1190 branches)
53 v13:
54  lines.....: 91.8% (3399 of 3701 lines)
55  functions...: 91.2% (124 of 136 functions)
56  branches...: 85.0% (1012 of 1190 branches)

```

```

57 v14:
58   lines.....: 87.5% (3237 of 3701 lines)
59   functions...: 90.4% (123 of 136 functions)
60   branches...: 80.5% (958 of 1190 branches)
61 v15:
62   lines.....: 75.8% (2806 of 3701 lines)
63   functions...: 77.9% (106 of 136 functions)
64   branches...: 66.1% (787 of 1190 branches)
65 v16:
66   lines.....: 91.9% (3403 of 3701 lines)
67   functions...: 91.9% (125 of 136 functions)
68   branches...: 85.1% (1013 of 1190 branches)
69 v17:
70   lines.....: 91.9% (3402 of 3700 lines)
71   functions...: 91.9% (125 of 136 functions)
72   branches...: 85.1% (1013 of 1190 branches)
73 v18:
74   lines.....: 91.9% (3403 of 3701 lines)
75   functions...: 91.9% (125 of 136 functions)
76   branches...: 85.1% (1013 of 1190 branches)
77 v19:
78   lines.....: 92.0% (3405 of 3703 lines)
79   functions...: 91.9% (125 of 136 functions)
80   branches...: 85.2% (1015 of 1192 branches)
81 v20:
82   lines.....: 91.9% (3403 of 3701 lines)
83   functions...: 91.9% (125 of 136 functions)
84   branches...: 85.1% (1013 of 1190 branches)
85 v21:
86   lines.....: 91.9% (3403 of 3701 lines)
87   functions...: 91.9% (125 of 136 functions)
88   branches...: 85.1% (1013 of 1190 branches)
89 v22:
90   lines.....: 91.9% (3399 of 3697 lines)
91   functions...: 91.9% (125 of 136 functions)
92   branches...: 85.2% (1012 of 1188 branches)
93 v23:
94   lines.....: 91.9% (3403 of 3701 lines)
95   functions...: 91.9% (125 of 136 functions)
96   branches...: 85.1% (1013 of 1190 branches)
97 v24:
98   lines.....: 91.9% (3391 of 3689 lines)
99   functions...: 91.9% (125 of 136 functions)
100  branches...: 85.1% (1002 of 1178 branches)
101 v25:
102  lines.....: 90.0% (3331 of 3701 lines)
103  functions...: 89.7% (122 of 136 functions)
104  branches...: 82.9% (986 of 1190 branches)
105 v26:
106  lines.....: 91.1% (3373 of 3701 lines)
107  functions...: 91.9% (125 of 136 functions)
108  branches...: 83.7% (996 of 1190 branches)
109 v27:
110  lines.....: 91.9% (3403 of 3701 lines)
111  functions...: 91.9% (125 of 136 functions)
112  branches...: 85.1% (1013 of 1190 branches)

```

113 v28:
114 lines.....: 77.8% (2881 of 3701 lines)
115 functions...: 82.4% (112 of 136 functions)
116 branches...: 70.5% (839 of 1190 branches)
117 v29:
118 lines.....: 91.8% (3396 of 3701 lines)
119 functions...: 91.9% (125 of 136 functions)
120 branches...: 84.7% (1008 of 1190 branches)
121 v30:
122 lines.....: 43.5% (1610 of 3701 lines)
123 functions...: 47.8% (65 of 136 functions)
124 branches...: 37.8% (450 of 1190 branches)
125 v31:
126 lines.....: 87.2% (3226 of 3701 lines)
127 functions...: 90.4% (123 of 136 functions)
128 branches...: 80.7% (960 of 1190 branches)
129 v32:
130 lines.....: 91.9% (3403 of 3701 lines)
131 functions...: 91.9% (125 of 136 functions)
132 branches...: 85.1% (1013 of 1190 branches)
133 v33:
134 lines.....: 91.9% (3403 of 3701 lines)
135 functions...: 91.9% (125 of 136 functions)
136 branches...: 85.1% (1013 of 1190 branches)
137 v34:
138 lines.....: 91.9% (3403 of 3701 lines)
139 functions...: 91.9% (125 of 136 functions)
140 branches...: 85.1% (1013 of 1190 branches)
141 v35:
142 lines.....: 91.8% (3393 of 3695 lines)
143 functions...: 91.9% (125 of 136 functions)
144 branches...: 84.7% (1005 of 1186 branches)
145 v36:
146 lines.....: 91.9% (3400 of 3701 lines)
147 functions...: 91.9% (125 of 136 functions)
148 branches...: 84.7% (1008 of 1190 branches)
149 v37:
150 lines.....: 91.9% (3403 of 3701 lines)
151 functions...: 91.9% (125 of 136 functions)
152 branches...: 85.1% (1013 of 1190 branches)
153 v38:
154 lines.....: 91.9% (3400 of 3701 lines)
155 functions...: 91.9% (125 of 136 functions)
156 branches...: 85.0% (1012 of 1190 branches)

(2) 各版本覆盖率信息（数据可视化版本）
V0(正确版本):

LCOV - code coverage report									
Current view: top level				Hit		Total		Coverage	
Test: v0.info				Lines:		3403		3701	
Date: 2024-04-08 17:50:17				Functions:		125		136	
				Branches:		1013		1190	
								91.9 %	
								91.9 %	
								85.1 %	
Generated by: LCOV version 1.14									

V1:

LCOV - code coverage report

Current view: top level

Test: v1.info

Date: 2024-04-08 17:50:20

Lines:3403370191.9 %

Functions:12513691.9 %

Branches:1013119085.1 %

Directory	Line Coverage %	Functions %	Branches %
SOURCE	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V2:

LCOV - code coverage report									
Current view: top level				Hit		Total		Coverage	
Test: v2.info				Lines:		3403		3701	
Date: 2024-04-08 17:50:23				Functions:		125		136	
				Branches:		1013		1190	
								91.9 %	
								91.9 %	
								85.1 %	
Directory				Line Coverage ↓		Functions ↓		Branches ↓	
src\code				<div><div></div></div> 91.9 %		3403 / 3701		91.9 % 125 / 136	
								85.1 % 1013 / 1190	
Generated by: LCOV version 1.14									

V3:

Current view: top level

Test: v3.info

Date: 2024-04-08 17:50:27

	Hit	Total	Coverage
Lines:	3403	3701	91.9 %
Functions:	125	136	91.9 %
Branches:	1013	1190	85.1 %

Directory	Line Coverage ▾	Functions ▾	Branches ▾
source	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V4:

LCOV - code coverage report									
Current view: top level				Hit		Total		Coverage	
Test: v4.info				Lines:		1118		3701	
Date: 2024-04-08 17:50:30				Functions:		45		136	
				Branches:		287		1190	
								30.2 %	
								33.1 %	
								24.1 %	
Directory				Line Coverage ↕		Functions ↕		Branches ↕	
SOURCE				<div><div></div></div> 30.2 % 1118 / 3701		33.1 % 45 / 136		24.1 % 287 / 1190	
Generated by: LCOV version 1.14									

V5:

LCOV - code coverage report

Current view: top level

Test: v5.info

Date: 2024-04-08 17:50:33

Lines: 3260 / 3701

Functions: 124 / 136

Branches: 934 / 1190

Hit

Total

Coverage

88.1 %

91.2 %

78.5 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	88.1 % 3260 / 3701	91.2 % 124 / 136	78.5 % 934 / 1190

Generated by: LCOV version 1.14

V6:

LCOV - code coverage report									
Current view: top level				Hit		Total		Coverage	
Test: v6.info				Lines:		1051		3701	
Date: 2024-04-08 17:50:36				Functions:		45		136	
				Branches:		257		1190	
								28.4 %	
								33.1 %	
								21.6 %	
Generated by: LCOV version 1.14									

V7:

LCOV - code coverage report

Current view: top level

Test: v7.info

Date: 2024-04-08 17:50:39

Lines:3399369791.9 %

Functions:12513691.9 %

Branches:1009118685.1 %

Directory	Line Coverage ↓	Functions ↓	Branches ↓
source	<div><div></div></div> 91.9 %3399 / 3697	91.9 %125 / 136	85.1 %1009 / 1186

Generated by: LCOV version 1.14

V8:

LCOV - code coverage report						
Current view: top level			Hit		Total	Coverage
Test: v8.info			Lines:	3399	3697	91.9 %
Date: 2024-04-08 17:50:42			Functions:	125	136	91.9 %
			Branches:	1009	1186	85.1 %
Directory			Line Coverage ↕		Functions ↕	Branches ↕
source			<div><div></div></div> 91.9 %	3399 / 3697	91.9 % 125 / 136	85.1 % 1009 / 1186
Generated by: LCOV version 1.14						

V9:

LCOV - code coverage report						
Current view: top level			Hit		Total	Coverage
Test: v9.info			Lines:	3403	3701	91.9 %
Date: 2024-04-08 17:50:45			Functions:	125	136	91.9 %
			Branches:	1013	1190	85.1 %
Directory			Line Coverage ↕		Functions ↕	Branches ↕
source			<div><div></div></div> 91.9 %	3403 / 3701	91.9 % 125 / 136	85.1 % 1013 / 1190
Generated by: LCOV version 1.14						

V10:

LCOV - code coverage report

Current view: top level

Test: v10.info

Date: 2024-04-08 17:50:48

	Hit	Total	Coverage
Lines:	3403	3701	91.9 %
Functions:	125	136	91.9 %
Branches:	1013	1190	85.1 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V11:

LCOV - code coverage report						
Current view: top level			Hit		Total	Coverage
Test: v11.info			Lines:	3403	3701	91.9 %
Date: 2024-04-08 17:50:51			Functions:	125	136	91.9 %
			Branches:	1013	1190	85.1 %
Directory			Line Coverage ↕		Functions ↕	Branches ↕
source			<div><div></div></div> 91.9 %	3403 / 3701	91.9 % 125 / 136	85.1 % 1013 / 1190
Generated by: LCOV version 1.14						

V12:

LCOV - code coverage report							
Current view: top level							
Test: v12.info							
Date: 2024-04-08 17:50:54							
			Lines:	Hit	Total	Coverage	
				3403	3701	91.9 %	
			Functions:	125	136	91.9 %	
			Branches:	1013	1190	85.1 %	
Directory		Line Coverage ↕		Functions ↕		Branches ↕	
source		<div><div></div></div> 91.9 %	3403 / 3701	91.9 %	125 / 136	85.1 %	1013 / 1190
Generated by: LCOV version 1.14							

V13:

LCOV - code coverage report						
Current view: top level			Hit		Total	Coverage
Test: v13.info			Lines:	3399	3701	91.8 %
Date: 2024-04-08 17:50:57			Functions:	124	136	91.2 %
			Branches:	1012	1190	85.0 %
Directory	Line Coverage ↕	Functions ↕	Branches ↕			
source	<div><div></div></div> 91.8 %3399 / 3701	91.2 %124 / 136	85.0 %1012 / 1190			
Generated by: LCOV version 1.14						

V14:

LCOV - code coverage report

Current view: top level

Test: v14.info

Date: 2024-04-08 17:51:00

	Hit	Total	Coverage
Lines:	3237	3701	87.5 %
Functions:	123	136	90.4 %
Branches:	958	1190	80.5 %

Directory	Line Coverage	Functions	Branches
source	<div><div></div></div> 87.5 %3237 / 3701	<div><div></div></div> 90.4 %123 / 136	<div><div></div></div> 80.5 %958 / 1190

Generated by: LCOV version 1.14

V15:

LCOV - code coverage report					
Current view: top level		Hit		Total	Coverage
Test: v15.info		Lines:	2806	3701	75.8 %
Date: 2024-04-08 17:51:03		Functions:	106	136	77.9 %
		Branches:	787	1190	66.1 %
Directory	Line Coverage ↕	Functions ↕	Branches ↕		
source	<div><div></div></div> 75.8 %2806 / 3701	77.9 %106 / 136	66.1 %787 / 1190		
Generated by: LCOV version 1.14					

V16:

LCOV - code coverage report					
Current view: top level		Hit		Total	Coverage
Test: v16.info		Lines:	3403	3701	91.9 %
Date: 2024-04-08 17:51:06		Functions:	125	136	91.9 %
		Branches:	1013	1190	85.1 %
Directory	Line Coverage	Functions	Branches		
source	<div><div></div></div> 91.9 % 3403 / 3701	91.9 % 125 / 136	85.1 % 1013 / 1190		
Generated by: LCOV version 1.14					

V17:

LCOV - code coverage report

Current view: top level

Test: v17.info

Date: 2024-04-08 17:51:10

	Hit	Total	Coverage
Lines:	3402	3700	91.9 %
Functions:	125	136	91.9 %
Branches:	1013	1190	85.1 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.9 %3402 / 3700	<div><div></div></div> 91.9 %125 / 136	<div><div></div></div> 85.1 %1013 / 1190

Generated by: LCOV version 1.14

V18:

LCOV - code coverage report

Current view: top level

Test: v18.info

Date: 2024-04-08 17:51:13

Lines:	3403	3701	91.9 %
Functions:	125	136	91.9 %
Branches:	1013	1190	85.1 %

Directory	Line Coverage	Functions	Branches
source	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V19:

LCOV - code coverage report

Current view: top level

Test: v19.info

Date: 2024-04-08 17:51:16

Lines:	3405	3703	92.0 %
Functions:	125	136	91.9 %
Branches:	1015	1192	85.2 %

Directory	Line Coverage	Functions	Branches
source	<div><div></div></div> 92.0 %3405 / 3703	<div><div></div></div> 91.9 %125 / 136	<div><div></div></div> 85.2 %1015 / 1192

Generated by: LCOV version 1.14

V20:

LCOV - code coverage report

Current view: top level

Test: v20.info

Date: 2024-04-08 17:51:19

Hit

Total

Coverage

Lines:

3403

3701

91.9 %

Functions:

125

136

91.9 %

Branches:

1013

1190

85.1 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V21:

LCOV - code coverage report

Current view: top level

Test: v21.info

Date: 2024-04-08 17:51:23

	Hit	Total	Coverage
Lines:	3403	3701	91.9 %
Functions:	125	136	91.9 %
Branches:	1013	1190	85.1 %

Directory	Line Coverage	Functions	Branches
source	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V22:

LCOV - code coverage report

Current view: top level

Test: v22.info

Date: 2024-04-08 17:51:26

	Hit	Total	Coverage
Lines:	3399	3697	91.9 %
Functions:	125	136	91.9 %
Branches:	1012	1188	85.2 %

Directory	Line Coverage	Functions	Branches
source	<div><div></div></div> 91.9 %3399 / 3697	<div><div></div></div> 91.9 %125 / 136	<div><div></div></div> 85.2 %1012 / 1188

Generated by: LCOV version 1.14

V23:

LCOV - code coverage report

Current view: top level

Test: v23.info

Date: 2024-04-08 17:51:29

Hit

Total

Coverage

Lines:

3403

3701

91.9 %

Functions:

125

136

91.9 %

Branches:

1013

1190

85.1 %

Directory	Line Coverage	Functions	Branches
source	91.9 % 3403 / 3701	91.9 % 125 / 136	85.1 % 1013 / 1190

Generated by: LCOV version 1.14

V24:

LCOV - code coverage report

Current view: top level

Test: v24.info

Date: 2024-04-08 17:51:32

Hit

Total

Coverage

Lines:

3391

3689

91.9 %

Functions:

125

136

91.9 %

Branches:

1002

1178

85.1 %

Directory	Line Coverage	Functions	Branches
source	<div><div></div>91.9 %3391 / 3689</div>	<div><div></div>91.9 %125 / 136</div>	<div><div></div>85.1 %1002 / 1178</div>

Generated by: LCOV version 1.14

V25:

LCOV - code coverage report

Current view: top level

Test: v25.info

Date: 2024-04-08 17:51:38

Hit

Total

Coverage

Lines: 3331370190.0 %

Functions: 12213689.7 %

Branches: 986119082.9 %

Directory	Line Coverage <div></div> <div>90.0 %3331 / 3701</div>	Functions <div></div> <div>89.7 %122 / 136</div>	Branches <div></div> <div>82.9 %986 / 1190</div>
source			

Generated by: LCOV version 1.14

V26:

LCOV - code coverage report

Current view: top level

Test: v26.info

Date: 2024-04-08 17:51:43

Lines:3373370191.1 %

Functions:12513691.9 %

Branches:996119083.7 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.1 %3373 / 3701	91.9 %125 / 136	83.7 %996 / 1190

Generated by: LCOV version 1.14

V27:

LCOV - code coverage report

Current view: top level

Test: v27.info

Date: 2024-04-08 17:51:46

	Hit	Total	Coverage
Lines:	3403	3701	91.9 %
Functions:	125	136	91.9 %
Branches:	1013	1190	85.1 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.9 %3403 / 3701	91.9 %125 / 136	85.1 %1013 / 1190

Generated by: LCOV version 1.14

V28:

LCOV - code coverage report					
Current view: top level		Hit		Total	Coverage
Test: v28.info		Lines:	2881	3701	77.8 %
Date: 2024-04-08 17:51:49		Functions:	112	136	82.4 %
		Branches:	839	1190	70.5 %
Directory	Line Coverage	Functions	Branches		
source	<div><div></div></div> 77.8 %2881 / 3701	82.4 %112 / 136	70.5 %839 / 1190		
Generated by: LCOV version 1.14					

V29:

LCOV - code coverage report

Current view: top level

Test: v29.info

Date: 2024-04-08 17:51:53

	Hit	Total	Coverage
Lines:	3396	3701	91.8 %
Functions:	125	136	91.9 %
Branches:	1008	1190	84.7 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.8 %3396 / 3701	91.9 %125 / 136	84.7 %1008 / 1190

Generated by: LCOV version 1.14

V30:

LCOV - code coverage report

Current view: top level

Test: v30.info

Date: 2024-04-08 17:52:08

Lines:1610370143.5 %

Functions:6513647.8 %

Branches:450119037.8 %

Directory

source

43.5 %

1610 / 3701

47.8 %

65 / 136

37.8 %

450 / 1190

Generated by: LCOV version 1.14

V31:

LCOV - code coverage report

Current view: top level

Test: v31.info

Date: 2024-04-08 17:52:11

Lines:3226370187.2 %

Functions:12313690.4 %

Branches:960119080.7 %

Directory	Line Coverage	Functions	Branches
source	87.2 % 3226 / 3701	90.4 % 123 / 136	80.7 % 960 / 1190

Generated by: LCOV version 1.14

V32:

LCOV - code coverage report

Current view: top level

Test: v32.info

Date: 2024-04-08 17:52:14

Lines:3403370191.9%

Functions:12513691.9%

Branches:1013119085.1%

Directory	Line Coverage ↕	Functions ↕	Branches ↕
source	<div><div></div></div> 91.9%3403 / 3701	<div><div></div></div> 91.9%125 / 136	<div><div></div></div> 85.1%1013 / 1190

Generated by: LCOV version 1.14

V33:

LCOV - code coverage report

Current view: top level

Test: v33.info

Date: 2024-04-08 17:52:18

Hit

3403

125

1013

Total

3701

136

1190

Coverage

91.9 %

91.9 %

85.1 %

Directory

source

Line Coverage

91.9 %

3403 / 3701

Functions

91.9 %

125 / 136

Branches

85.1 %

1013 / 1190

Generated by: LCOV version 1.14

V34:

LCOV - code coverage report					
Current view: top level		Hit		Total	Coverage
Test: v34.info		Lines:	3403	3701	91.9 %
Date: 2024-04-08 17:52:21		Functions:	125	136	91.9 %
		Branches:	1013	1190	85.1 %
Directory	Line Coverage	Functions	Branches		
source	<div><div></div></div> 91.9 % 3403 / 3701	<div><div></div></div> 91.9 % 125 / 136	<div><div></div></div> 85.1 % 1013 / 1190		
Generated by: LCOV version 1.14					

V35:




LCOV - code coverage report

Current view: top level

Test: v35.info

Date: 2024-04-08 17:52:25

Lines:	Hit3393	Total3695	Coverage91.8 %
Functions:	125	136	91.9 %
Branches:	1005	1186	84.7 %

Directory	Line Coverage 	Functions 	Branches 
source	91.8 % 3393 / 3695	91.9 % 125 / 136	84.7 % 1005 / 1186

Generated by: LCOV version 1.14

V36:

LCOV - code coverage report						
Current view: top level		Hit		Total	Coverage	
Test: v36.info		Lines:	3400	3701	91.9 %	
Date: 2024-04-08 17:52:29		Functions:	125	136	91.9 %	
		Branches:	1008	1190	84.7 %	
Directory		Line Coverage		Functions	Branches	
source		91.9 % 3400 / 3701		91.9 % 125 / 136	84.7 % 1008 / 1190	
Generated by: LCOV version 1.14						

V37:

LCOV - code coverage report

Current view: top level

Test: v37.info

Date: 2024-04-08 17:52:32

Lines:3403370191.9 %

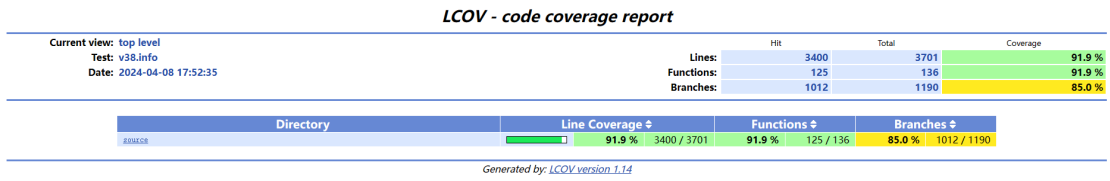
Functions:12513691.9 %

Branches:1013119085.1 %

Directory	Line Coverage	Functions	Branches
source	91.9 % 3403 / 3701	91.9 % 125 / 136	85.1 % 1013 / 1190

Generated by: LCOV version 1.14

V38:



(3) 整理得到覆盖率表格

表 7 覆盖率表格

版本号	语句覆盖率	函数覆盖率	分支覆盖率
v0	91.9%	91.9%	85.1%
v1	91.9%	91.9%	85.1%
v2	91.9%	91.9%	85.1%
v3	91.9%	91.9%	85.1%
v4	30.2%	33.1%	24.1%
v5	88.1%	91.2%	78.5%
v6	28.4%	33.1%	21.6%
v7	91.9%	91.9%	85.1%
v8	91.9%	91.9%	85.1%
v9	91.9%	91.9%	85.1%
v10	91.9%	91.9%	85.1%
v11	91.9%	91.9%	85.1%
v12	91.9%	91.9%	85.1%
v13	91.8%	91.2%	85.0%
v14	87.5%	90.4%	80.5%
v15	75.8%	77.9%	66.1%
v16	91.9%	91.9%	85.1%
v17	91.9%	91.9%	85.1%
v18	91.9%	91.9%	85.1%
v19	92.0%	91.9%	85.2%
v20	91.9%	91.9%	85.1%
v21	91.9%	91.9%	85.1%
v22	91.9%	91.9%	85.2%
v23	91.9%	91.9%	85.1%
v24	91.9%	91.9%	85.1%

v25	90.0%	89.7%	82.9%
v26	91.1%	91.9%	83.7%
v27	91.9%	91.9%	85.1%
v28	77.8%	82.4%	70.5%
v29	91.8%	91.9%	84.7%
v30	43.5%	47.8%	37.8%
v31	87.2%	90.4%	80.7%
v32	91.9%	91.9%	85.1%
v33	91.9%	91.9%	85.1%
v34	91.9%	91.9%	85.1%
v35	91.8%	91.9%	84.7%
v36	91.9%	91.9%	84.7%
v37	91.9%	91.9%	85.1%
v38	91.9%	91.9%	85.0%

(4) 绘制折线图

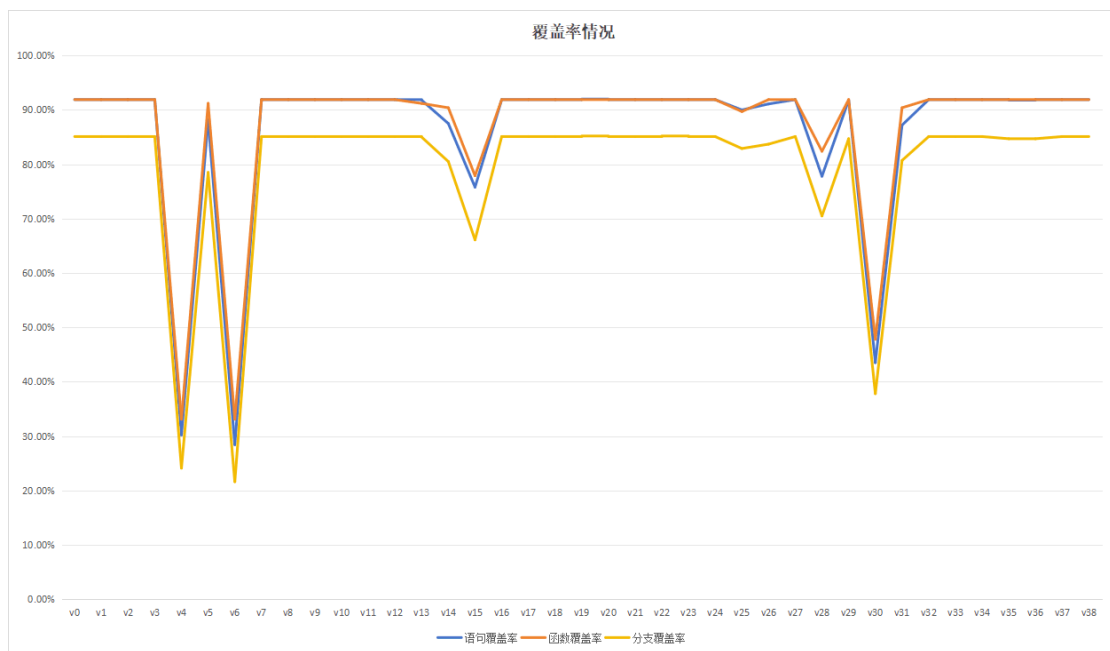


图 29 覆盖率折线图

(5) 源代码运行情况

①以正确版本 v0 为例；

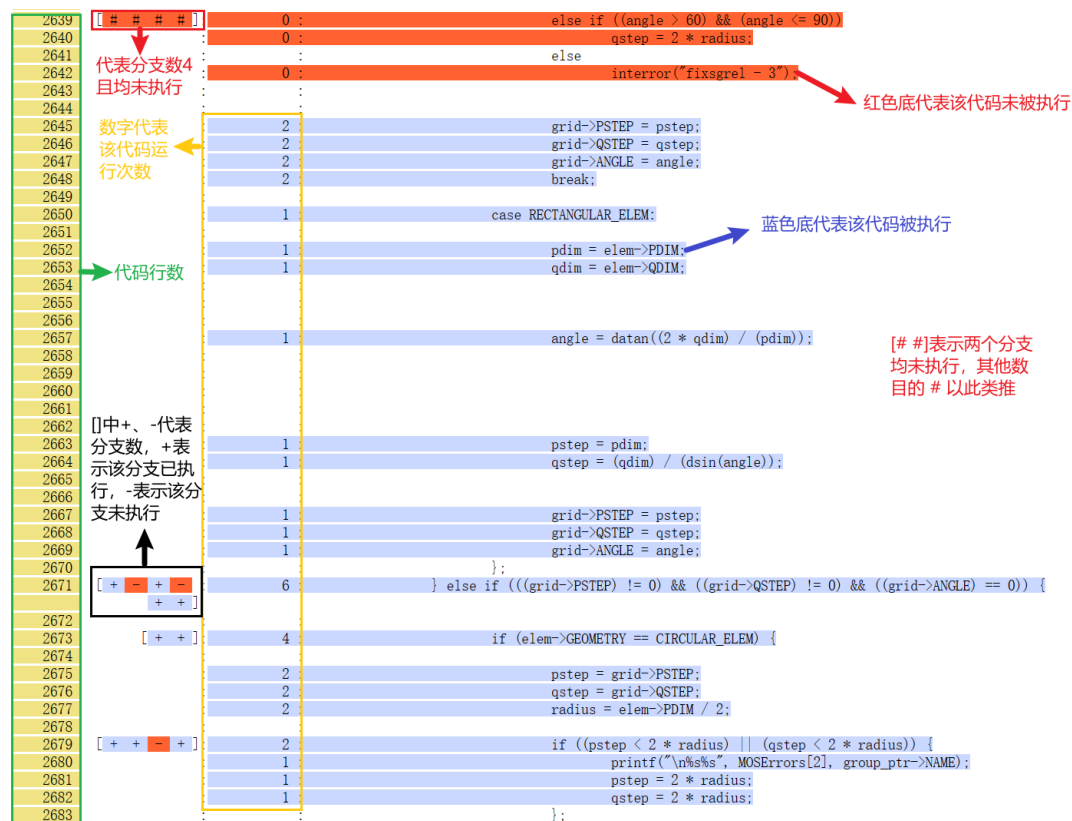


图 30 源代码运行情况示例 1

图中最左侧数字表示该代码语句的行数，右侧主体为代码语句内容，蓝底表示该语句有被执行，红底表示该语句未被执行，紧挨着代码语句前的数字表示该语句的执行次数，在执行次数之前有[]，其中的“+”“-”“#”表示分支数目，蓝底“+”表示该分支有被执行，红底“-”表示该分支未被执行，若有[# #]的情况，则表示此处的两个分支均未被执行，更多的“#”的情况以此类推。

②以覆盖率较低的错误版本 v4 为例；

```

601      0 : if (error == 17) {
602      0 :     *addrem_ptr = NULL;
603      0 :     parserro(*curr_ptr, 58, " ");
604      0 :
605      0 :     return 17;
606      0 : }
607
608
609      0 : if (error == 0) {
610      0 :     *addrem_ptr = (struct AddRem *) malloc(sizeof(struct AddRem));
611      0 :
612      0 :     if (*addrem_ptr == NULL) {
613      0 :         parserro(*curr_ptr, 55, " ");
614      0 :
615      0 :         interror("adddef()");
616      0 :     }
617
618      0 :     (*addrem_ptr)->ADDREM_TYPE = ADD_TYPE;
619      0 :     (*addrem_ptr)->BLOCK_TYPE = NODE_BLOCK;
620      0 :     (*addrem_ptr)->NODE_PTR = *node_ptr;
621      0 :     (*addrem_ptr)->NEXT = NULL;
622      0 :
623      0 :     *pp2 = *curr_ptr;
624      0 :     return 0;
625      0 : }
626
627
628
629
630
631
632
633      0 : error = blockdef(*curr_ptr, curr_ptr, node_ptr);
634
635      0 : if (error == 17) {
636      0 :     *addrem_ptr = NULL;
637      0 :     parserro(*curr_ptr, 59, " ");
638      0 :
639      0 :     return 17;
640      0 : }
641
642
643
644      0 : if (error == 0) {
645      0 :     *addrem_ptr = (struct AddRem *) malloc(sizeof(struct AddRem));
646      0 :
647      0 :     if (*addrem_ptr == NULL) {
648      0 :         parserro(*curr_ptr, 55, " ");
649      0 :
650      0 :         interror("adddef()");
651      0 :     }
652
653      0 :     (*addrem_ptr)->ADDREM_TYPE = ADD_TYPE;
654      0 :     (*addrem_ptr)->BLOCK_TYPE = BLOCK_BLOCK;
655      0 :     (*addrem_ptr)->NODE_PTR = *node_ptr;
656      0 :     (*addrem_ptr)->NEXT = NULL;
657      0 :
658      0 :     *pp2 = *curr_ptr;
659      0 :     return 0;
660      0 : }
661

```

图 31 源代码运行情况示例 2

可以观察到该版本中许多代码均未被执行，符合覆盖率低的结果。

4.1.2 分析

通过观察表格和折线图，可以分析总结出以下结论：

(1) 高覆盖率版本：

版本 v0、v1、v2、v3、v7 至 v13、v16、v17、v18、v19、v20、v21、v22、v23、v24、v25、v27 至 v38 显示出了非常高的覆盖率，其中代码行覆盖率接近或达到了 92%，函数覆盖率维持在 91.9%或略低，分支覆盖率在 84.7%到 85.2%之间。这表明这些版本的代码已经通过测试用例得到了很好的验证，大部分的代码路径都被执行过。

(2) 中等覆盖率版本：

版本 v14 和 v15 的覆盖率相对较低，尤其是版本 v15 的代码行覆盖率下降到了 75.8%，函数覆盖率是 77.9%，分支覆盖率是 66.1%。这可能意味着这些版本

的代码还有相当一部分没有被测试用例覆盖到，需要增加更多的测试用例来提高覆盖率。

(3) 低覆盖率版本：

版本 v4、v6 和 v30 的覆盖率非常低。特别是 v30 的代码行覆盖率只有 43.5%，函数覆盖率是 47.8%，分支覆盖率是 37.8%。这些低覆盖率可能指示着代码中存在未被发现的 Bug 和潜在问题，需要立即关注并采取措施提高测试覆盖率。

(4) 覆盖率波动：

版本 v5 的代码行覆盖率较高（88.1%），但函数覆盖率略低（91.2%），分支覆盖率也较高（78.5%）。这可能表明虽然大部分代码被执行过，但仍有部分函数没有被完全测试。

(5) 覆盖率变化：

版本 v19 的代码行覆盖率略有提升，达到了 92.0%，分支覆盖率也提高了 0.1 个百分点，这表明测试用例可能对代码的某些部分产生了更好的覆盖效果。

(6) 覆盖率下降版本：

版本 v28 的覆盖率相比其他版本有显著下降，这可能是因为测试用例没有覆盖到更多的代码路径，或者代码本身的变动导致了覆盖率的下降。

总结来说，大部分版本的覆盖率都很高，但仍有几个版本的覆盖率较低，需要进一步的测试和分析。特别是版本 v30 的覆盖率极低，应当是优化和改进的重点。此外，对于覆盖率异常的版本，也需要进一步分析原因，是否是测试用例集的问题，或者是代码变动导致的覆盖率变化。通过这些分析，可以帮助我们更好地理解代码质量和测试效果，从而提高软件的质量和可靠性。

4.2 测试用例通过结果及分析

4.2.1 测试用例通过结果

因为本次程序运行结果中存在大量信息，不便于人工观看，所以我们对每个测试用例输入正确版本和错误版本后得到的输出内容进行处理和比较。当错误版本的输出结果与正确版本存在不同时，则视作该测试用例在该错误版本中没有通过，并将最终结果记录在 `source/compareInfo` 文件中，如下图，其中版本号后的数字 a/b，b 表示总的测试用例数目，a 表示该版本中成功通过测试、与正确版本输出相同的测试用例数目。

(1) 文字形式汇总

1 v1: 152/152
2 v2: 152/152
3 v3: 134/152
4 v4: 28/152
5 v5: 110/152
6 v6: 20/152
7 v7: 149/152
8 v8: 150/152
9 v9: 131/152
10 v10: 146/152
11 v11: 143/152
12 v12: 152/152
13 v13: 148/152
14 v14: 130/152
15 v15: 99/152
16 v16: 143/152
17 v17: 151/152
18 v18: 152/152
19 v19: 144/152
20 v20: 151/152
21 v21: 151/152
22 v22: 151/152
23 v23: 148/152
24 v24: 141/152
25 v25: 130/152
26 v26: 139/152
27 v27: 152/152
28 v28: 106/152
29 v29: 145/152
30 v30: 53/152
31 v31: 131/152
32 v32: 152/152
33 v33: 152/152
34 v34: 152/152
35 v35: 148/152
36 v36: 149/152
37 v37: 150/152
38 v38: 151/152

图 32 测试用例通过结果

(2) 整理得到表格

表 8 通过率表格

版本号	通过用例/总用例	通过率
v1	152/152	100%
v2	152/152	100%
v3	134/152	88.16%
v4	28/152	18.42%
v5	110/152	72.36%
v6	20/152	13.16%
v7	149/152	97.98%
v8	150/152	98.68%
v9	131/152	86.18%

v10	146/152	95.98%
v11	143/152	94.08%
v12	152/152	100%
v13	148/152	97.36%
v14	130/152	85.46%
v15	99/152	65.13%
v16	143/152	94.08%
v17	151/152	99.34%
v18	152/152	100%
v19	144/152	94.74%
v20	151/152	99.34%
v21	151/152	99.34%
v22	151/152	99.34%
v23	148/152	97.36%
v24	141/152	92.76%
v25	130/152	85.46%
v26	139/152	91.45%
v27	152/152	100%
v28	106/152	69.74%
v29	145/152	95.39%
v30	53/152	34.87%
v31	131/152	86.18%
v32	152/152	100%
v33	152/152	100%
v34	152/152	100%
v35	148/152	97.36%
v36	149/152	97.98%
v37	150/152	98.68%
v38	151/152	99.34%

(3) 绘制折线图

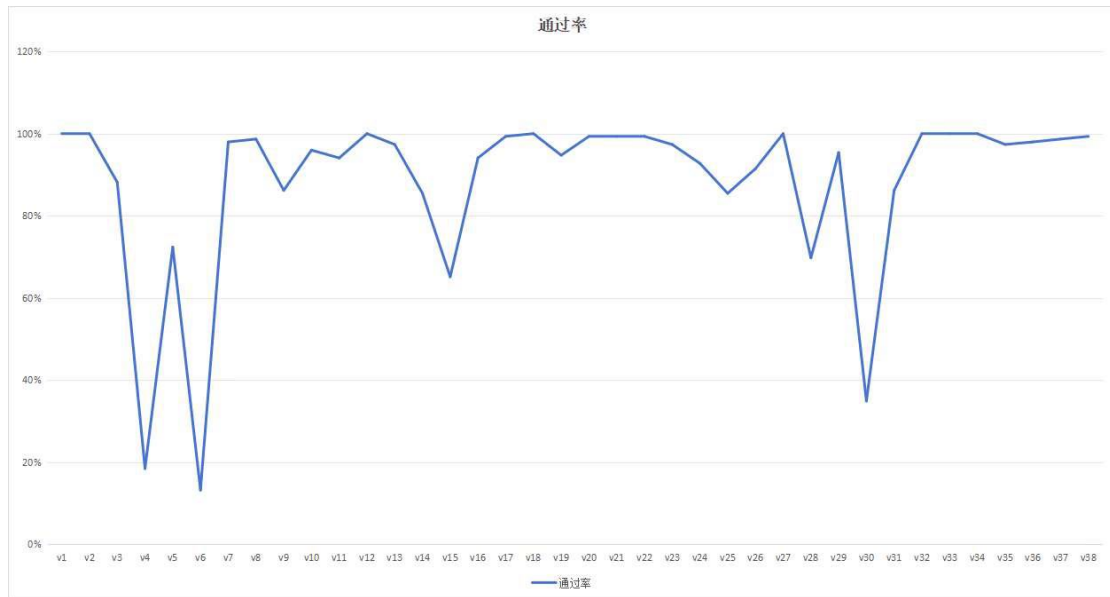


图 33 通过率折线图

4.2.2 分析

通过观察表格和折线图，我们可以分析总结出以下结论：

（1）高通过率版本：

版本 v1、v2、v7、v12、v18、v27、v32、v33 和 v34 的通过率均为 100%，这意味着这些版本的所有测试用例都成功通过，显示出这些版本的软件质量非常高，测试基本覆盖了所有的功能点。

（2）良好通过率版本：

版本 v3、v8、v9、v10、v11、v13、v16、v17、v19、v20、v21、v22、v23、v24、v26、v29、v31、v35、v36、v37、v38 的通过率在 88.16%到 99.34%之间，这些版本的测试用例执行情况良好，但仍有提升空间。特别是 v17、v20、v21、v22 和 v38 的通过率接近 100%，表明这些版本的软件质量较为良好。

（3）中等通过率版本：

版本 v5、v14、v25 和 v28 的通过率在 69.74%到 85.46%之间，这些版本的测试用例执行情况一般，需要进一步分析未通过的测试用例，找出原因并进行修复。

（4）低通过率版本：

版本 v4、v6、v15 和 v30 的通过率较低，尤其是 v4、v6 和 v30 的通过率远低于平均水平，这可能指示着这些版本的软件存在较多问题或者测试用例设计不够全面。对于这些版本，需要重点关注，增加测试用例的覆盖率，修复发现的问题，并可能需要重新设计或增加测试用例。

（5）后续改进：

对于通过率较低版本，需要检查测试用例的设计是否合理，是否覆盖了所有关键功能和边界条件。如果测试用例设计不足，即使软件本身质量较高，通过率也可能会受到影响。同时，对于所有版本，都应该持续跟踪和改进测试过程。对于通过率高的版本，可以总结成功的经验；对于通过率低的版本，需要找出原因并采取措施进行改进。

总结来说，大多数版本的测试通过情况良好，但仍有一些版本需要特别关注和改进。通过持续的测试和分析，可以提高软件的整体质量，并确保最终产品能够满足用户的需求和期望。

结论

通过本次在 Ubuntu 系统中实现的基于 space 程序的软件测试实验，我们使用了大量测试用例对正确版本和 38 个错误版本的程序进行测试，并进行了覆盖率数据和通过率数据的收集和分析，通过总结，我们可以发现以下几点：

（1）高覆盖率与高通过率的版本：

版本 v1、v2、v7、v12、v18、v27 和 v32 在覆盖率和通过率方面都表现出色，显示出这些版本的代码质量很高，测试用例设计得当，能够充分覆盖并验证关键功能。

（2）覆盖率与通过率不一致的版本：

版本 v3、v8、v9、v10、v11、v13、v14、v16、v17、v19、v20、v21、v22、v23、v24、v25、v26、v29、v31、v35、v36 和 v37 的覆盖率和通过率之间存在一定的差异。尽管这些版本的覆盖率较高，但通过率并不是 100%，表明虽然测试用例覆盖了大部分代码，但仍有部分测试用例未能通过。这可能是由于测试用例的设计不够完善，或者存在一些边缘情况没有被完全考虑到。

（3）低覆盖率与低通过率的版本：

版本 v4、v5、v6、v15、v25、v28 和 v30 的覆盖率和通过率都较低，这可能表明这些版本的代码存在较多问题，或者测试用例未能充分覆盖到所有关键路径。这些版本需要重点关注，可能需要增加更多的测试用例，改进测试策略，以及对代码进行深入的审查和重构。

总体而言，通过本次实验，我们进一步熟悉了 Ubuntu 系统，学习到了 gcc、gcov、lcov 等工具的用法，明白了测试用例设计的重要性。高覆盖率是确保软件质量的关键因素，通过跟踪语句覆盖率、函数覆盖率和分支覆盖率等，我们可以更好地理解测试用例覆盖了代码的哪些部分，以及还有哪些部分需要进一步的测试关注。需要注意的是，高覆盖率是一个好的开始，但只有当测试用例能够有效地执行并且通过时，才能确保软件质量。通过率数据显示，即使某些版本的覆盖率较高，仍然存在未通过的测试用例，这提示我们需要重新审视和改进测试用例的设计，确保它们能够有效地检测出软件中的缺陷。这为我们未来的工作提供了指导，我们在实际的软件开发和测试过程中需要不断回顾和改进测试用例，确保它们能够捕捉到潜在的缺陷。最终，通过提高软件质量，我们可以提供更好的用户体验，进而提升用户满意度和信任度。

参考文献

- [1] 李正言. 计算机软件测试方法的研究[J]. 自动化应用, 2024, 65(02):199-201. DOI:10.19769/j.zdhy.2024.02.062.
- [2] 邱桥春, 陈平, 尚京威, 等. 嵌入式软件并行测试方法及系统框架[J]. 电子元件与信息技术, 2023, 7(11):54-57. DOI:10.19772/j.cnki.2096-4455.2023.11.014.
- [3] 刘娜. 计算机软件的测试方法与应用[J]. 集成电路应用, 2023, 40(12):60-61. DOI:10.19339/j.issn.1674-2583.2023.12.023.
- [4] 梁立新. 软件测试课程的项目驱动式教学方法与实践[J]. 高教学刊, 2023, 9(23):92-95. DOI:10.19980/j.CN23-1593/G4.2023.23.022.
- [5] 石佳琦, 陈鹏. Linux 平台下代码覆盖率报告自动化输出设计[J]. 计算机系统应用, 2019, 28(02):68-74. DOI:10.15888/j.cnki.csa.006776.
- [6] 褚悦. 代码覆盖率驱动的用例管理系统的设计与实现[D]. 西安电子科技大学, 2018.
- [7] 费训, 罗蕾. 利用 GNU 工具实现汇编程序覆盖测试[J]. 计算机应用, 2004(12):95-98.
- [8] 周雷. 嵌入式代码覆盖率统计方法[J]. 计算机应用与软件, 2014, 31(05):326-327.