



中山大學
SUN YAT-SEN UNIVERSITY

本科生毕业论文（设计）

Undergraduate Graduation Thesis (Design)

基于黑盒测试和编辑距离算法的程序填空题自动评测系统的设计
和实现

院 系: 数据科学与计算机学院
School (Department)

专 业: 软件工程
Major

学生姓名: 王 嘉 威
Student Name

学 号: 13331251
Student No.

指导教师 (职称): 万海 副教授
Supervisor (Title)

时间: 2017 年 4 月 29 日

学术诚信声明

本人所呈交的毕业论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。本毕业论文的知识产权归属于培养单位。本人完全意识到本声明的法律结果由本人承担。

本人签名：王嘉威

日期：2017.4.19

Statement of Academic Integrity

I hereby acknowledge that the thesis submitted is a product of my own independent research under the supervision of my supervisor, and that all the data, statistics, pictures and materials are reliable and trustworthy, and that all the previous research and sources are appropriately marked in the thesis, and that the intellectual property of the thesis belongs to the school. I am fully aware of the legal effect of this statement.

Student Signature: JiaWeiWang Date: 19/04/2017

摘 要

随着在线课程管理系统 Matrix 在数据科学与计算机学院的上线和应用,以程序评测为主功能,各种新型题目的在线评测实现需求也被提出。其中程序填空题便是重要的一环,传统程序填空题的评分一般都由人为进行,人为评分的缺点在于效率的低下。程序填空题主观性不强,答案较为固定,人为进行评分并没有体现出优越性,这自动评测需求出现的原因。程序填空题自动评测的难点在于虽然答案的主观性不强,但也存在各种语义性等价答案,并且也需要对部分错误的填空给出部分的分数,而不是该空出现错误就完全不给分。所以总结起来,程序填空需要应对的较难处理的答案分为等效答案以及部分得分答案两种。对于前者,可使用填空代码,并对代码进行评测,看评测的结果解决,后者通过与标准答案进行相似度比对,计算所得分解决。经过基于 Matrix 评测系统,将理论的算法应用到系统的评测逻辑的工作后,程序填空的评测功能已实装到 Matrix 应用中,并用于某课程的期末考试之中。根据评测的结果,虽然有些评分仍存有不合理,需要人为评分修改的地方,但这是程序填空评测算法探索的关键一步。之后对于实用结果的不合理性,对算法进行了进一步的改进,并用之前学生提交的答案数据再一次进行了评测,与改进前相比,准确性提高了许多。本文第一次尝试将普通的程序评测与传统的填空题评测的算法相结合,具有第一步的尝试意义,对于之后的程序填空算法的研究探索,有一定的参考价值。

[关键词] 程序填空; 代码评测; 文本相似度

ABSTRACT

As the online judge system, Matrix, was published and used in the courses of the School of Data and Computer Science. The requirement of developing new function of judging new type programs based on the judge system of Matrix was put forward, in which judging program filling problems was an important part. In tradition, program filling problems are graded by human, which is not effective because program filling problem is not a subjective problem, the answers of the problem are not variable. So it can be judge by the computer. However, there are also some difficulties in the research. Although the answers of program filling problem are fixed, there are also some possible equivalent answers that may appeared. On the other hand, when the answer of the student is not equal to the standard answer, such as the answer lack of a semicolon compared to the standard one, the answer should get part of the grades. To solve these two situations, two solutions were put forward. One is judging the program with the filling answers of student, the other is computing the similarity between the student answer and the standard answer. With the work of applying the algorithm to the judge system of Matrix, the function was achieved in Matrix application and was applied to the exam. However, there were also some problems in the judging results of the exam, which make it need to be corrected by human. But it is an important step of this research. According to the insufficient of the results, some improvements were applied to the judging algorithm. With the compared of the judging results using the data of student submissions of the exam, the improved algorithm had better effects. The paper firstly combine the program judging algorithm with the similarity algorithm, different from the traditional algorithm of judging program filling problem, which mainly only use similarity algorithm. So this paper has some reference value for the research of judging algorithm of program filling problem.

[Keywords] program filling;code judging;text similarity

目录

| | |
|----------------------------|----|
| 摘 要 | i |
| ABSTRACT | ii |
| 第一章 引言 | 1 |
| 1.1 程序填空评测问题的背景和意义 | 1 |
| 1.2 研究背景和现状 | 1 |
| 1.3 本文的工作 | 2 |
| 1.4 本文的论文结构与章节安排 | 2 |
| 1.5 本章小结 | 3 |
| 第二章 预备知识 | 4 |
| 2.1 动态测试 | 4 |
| 2.2 静态分析 | 5 |
| 2.3 本章小结 | 9 |
| 第三章 动态测试与静态分析结合的评测算法的实现与部署 | 10 |
| 3.1 算法的选择 | 10 |
| 3.2 程序填空评测问题算法的实现 | 10 |
| 3.3 程序填空评测的应用试验 | 17 |
| 3.4 本章总结 | 18 |
| 第四章 填空评测算法的改进 | 20 |
| 4.1 问题与改进算法 | 20 |
| 4.2 本章总结 | 23 |
| 第五章 改进后的仿真实验结果与分析 | 24 |
| 5.1 实验环境和参数设置 | 24 |
| 5.2 结果图表与具体分析 | 24 |
| 第六章 动态测试与静态分析结合的评测算法的实现与部署 | 26 |
| 参考文献 | 27 |

| | |
|-----------------------|----|
| 致谢 | 29 |
| 附录 A 补充更多细节 | 30 |

插图目录

| | | |
|-----|------------------------------------------------------|----|
| 3-1 | Matrix 内部子系统交互 | 12 |
| 3-2 | 考试题目描述 | 17 |
| 5-1 | 改进前与改进后的动态测试评测分数对比 (横轴为学生, 纵轴为得分, 满分为 8 分) | 24 |
| 5-2 | 算法改进前与改进后的评测总得分 | 25 |
| A-1 | 题目与描述 | 30 |
| A-2 | 挖空代码 | 30 |
| A-3 | 标准填空答案和分值 | 30 |
| A-4 | 题目相关文件上传 | 31 |
| A-5 | 时间和内存限制 | 31 |
| A-6 | 动态测试评测各阶段分值设置 | 31 |
| A-7 | 标准输入输出测试样例和随机测试文件 | 32 |
| A-8 | 程序填空题题目描述 | 32 |
| A-9 | 程序填空答案提交 | 32 |

表格目录

| | | |
|-----|----------------------|----|
| 2.1 | 编辑距离的说明 | 6 |
| 3.1 | 考试试验部分成绩统计 | 17 |

第一章 引言

2016 年, Matrix 在线评测系统在中山大学软件学院诞生, 并迅速为学院各大程序课程作业、考试所应用。Matrix 支持编程题、选择题、简答题等题型的评测或批改。随着 Matrix 系统的扩展, 支持新的题型也成了 Matrix 评测系统的一个新需求。本文致力于程序填空题的评测算法设计以及在系统中的部署工作, 为 Matrix 评测系统增加新的程序填空题题型的评测功能。

本文分为四节, 首先阐述程序填空题评测问题背景和意义, 然后介绍国内外相关研究背景与现状, 接着介绍本文的大体工作, 最后介绍本文的结构。

1.1 程序填空评测问题的背景和意义

程序填空题在程序设计课程考试中是一种重要的考察题型, 学生通过阅读分析缺失部分的代码与题干内容, 理解题目的意图和程序的思路, 填写出程序中缺失的部分。由于程序的语法的程序填空题型的答案一般较为固定, 一般为一条语句或者表达式, 主观性较弱, 因此具有易批量自动批改性。如今大学课程的程序填空题多出现于笔试中, 由人工进行批改, 效率较低。学院 Matrix 在线评测系统的上线, 使得程序填空题的自动评测成为了可能和需求。在程序填空题自动评测实现的环境下, 学生通过电脑提交填空答案, 由评测服务器进行自动评测得出评分, 有效提高批改效率以及准确度。对于 Matrix 评测系统的功能扩展以及程序填空题的应用算法研究具有重大的意义。

1.2 研究背景和现状

程序填空题是一种语言性算法的应用场景。程序填空题的特征在于其题干主要分为程序代码, 答案为程序语言, 并且程序代码具有可执行性。因此与程序填空题相关算法的研究包括程序语言的研究以及一般文本语言的研究 (应用于普通填空题)。

1.2.1 填空题相关领域的研究

对于一般的文本填空题, 可看作为多道简答题的集合 (每个空为一道简答题)。简答题的一般评分方式为将答案与标准答案进行语义性比对或者字符相似性比对。因此可以着眼于基本的字符串相似性度量和从答案的语义性进行分析的相似性度量的相关

性研究。

Andres Marzal 和 Enrique Vidal(1993) 提出了正则化的编辑距离计算算法和应用^[1]。这种相似性度量算法是语义无关的。RadaMihalcea 等 (2006) 提出了给予语料库和知识库的文本相似度的度量, 将相似性的度量上升到了语义的级别^[2]。Xiaoling Wang 等 (2007) 提出了基于 Gram 框架的字符串相似度搜索算法^[3]。Marios Hadjieleftheriou 等 (2014) 提出了以编辑距离算法为基础的使用 B+ 树的字符串相似性搜索^[4]。Lorraine A. K. Ayad 等 (2016) 提出固定长度近似字符串匹配软件库的开发^[5]。

1.2.2 程序语言相关的研究

程序语言特征方面的研究也是对程序填空题的自动评测的一种可参考的领域。David T. Barnard 等 (1982) 提出了利用 LR 文法修复语法错误的方法^[6]。Naohiro Ishiid 等 (1996) 提出了移除源代码多样性的方法^[7]。Cliare Le Goues 等 (2012) 提出了自动修复程序出现 bug 的方法^[8]。Anyu Wang 等 (2015) 提出了局部修复码 (Locally Repairable Codes) 的基于程序的整形边界^[9]。Paul W. McBurney 等 (2016) 提出了对于源代码间文本相似度的经验性学习算法^[10]。Dave Towey 等 (2017) 提出了一种可变的支持无测试 oracle 的程序修复的测试方法。^[11]。针对程序语言进行分析可以从语句中获取关键的信息, 是对信息的另一种过滤, 对于提高程序填空评测的准确度有着一定的意义。

1.3 本文的工作

本文主要工作分为以下三个部分:

- 1 设计程序评测的算法及对应的交互流程。算法采用动态测试与静态分析相结合的算法。
- 2 将算法部署到 Matrix 应用系统中。本算法针对服务端和评测系统的代码逻辑, 将程序填空题型的评测算法在服务端、评测系统及之间的交互中实现。
- 3 对算法效果进行实验和改进。本文通过评测算法的初步应用, 找出算法之中存在的缺陷, 并设计改进算法的方案。通过相同的答案提交数据进行对比实验, 分析出改进算法的评测效果的准确率提高以及提高原因。

1.4 本文的论文结构与章节安排

本文共分为六章, 各章节内容安排如下:

第一章引言。

第二章介绍了本文相关的领域的一些选型算法知识，分为针对代码可执行性的动态测试以及针对语言及内容特性的静态分析。

第三章阐述了本文的主要工作，包括算法的选择和系统各部分的部署以及实验后算法出现的问题。

第四章阐述了针对算法问题的改进以及为改进做的工作。

第五章阐述了改进后的实验结果与分析。

第六章总结了这次工作的不足与后续工作的展望。

1.5 本章小结

本章介绍了本文研究的程序填空问题的意义以及相关研究背景、本文的工作以及论文结构，下一章会阐述程序填空评测算法设计所参考的各种选型算法以及基本概念。

第二章 预备知识

在这一章中，我们将介绍程序填空评测相关领域的一些研究的算法以及概念知识。根据一章中对程序填空题的特性分析，可以应用到程序填空评测的算法可归类动态测试和静态分析两种。下面对这两种相关算法进行阐述。

2.1 动态测试

动态测试是程序设计题评测所使用的最基本的评测方式。使用黑盒测试的方式，将学生提交的代码作为黑盒，使用测试输入样例运行程序，对比输出结果和标准程序对应输出是否一致。

2.1.1 基本动态测试

Matrix 系统现今使用基本的基于黑盒测试的动态测试算法。该算法广泛应用于多种在线程序评测网站平台。基本的评测流程如算法2.2

算法 2.1: 基本动态测试算法

输入: 学生提交代码

输出: 程序评测分数 *grade*

```
1 编译学生提交代码
2 如果 代码编译成功 则
3   对于所有 题目标准测例中的每个输入 进行
4     评测系统使用标准测例输入运行学生程序
5     如果 学生代码成功运行并得到输出 则
6       对比代码输出与标准测例输出
7       记录评测报告
8       如果 输出一致 则
9          $grade = grade + \text{该测例的分值}$ 
```

算法2.2为简化版的动态测试，由于在程序填空题中的题目的复杂性较低，因此未考虑内存消耗以及运行时间等问题，这几点因素均采用预设的默认的标准值，不计入采分点。该算法优点在于容易判断程序等价性，缺点是考试评测的程序中往往存在词法或语法错误，或常常由于语义错误（如多写了一个分号）导致死循环或无法通过编译，得不到执行结果，导致得分为零，违背了程序填空题的评分的容错性。

2.1.2 改进的动态测试

基本动态测试的问题在于对无法执行出结果的程序或者因部分内容错误导致输出结果产生偏差的程序，无法进行容错性评分。因此出现了通过将语法编译不通过的答案修整为可编译执行的语句或表达式的思路。该算法依据程序语言的语法特性，在保证不对答案正确部分产生破坏的前提下，尽最大可能修正答案的语法错误，转换为可执行的表达式或语句填入程序进行评测。

目前对程序进行语法修复的方法有基于 LR 文法的语法修复^[6]和基于搜索、基于代码穷举、基于约束求解的方法^[12]。更深入的研究则有基于云计算来改善代码的运行性 bug（死循环、内存泄漏等）^[8]。

结合程序修复的动态测试算法如下：

算法 2.2: 改进后的动态测试算法

输入: 学生提交代码

输出: 程序评测分数 $grade$

```
1 编译学生提交代码
2 如果 代码编译失败 则
3   | 使用程序修复算法修复代码，计算出修复补正系数 (0-1)，程序的语法错误
   | 性越高，补正系数越低。
4 否则
5   | 程序修复补正系数为 1。
6 对于所有 题目标准测例中的每个输入 进行
7   | 评测系统使用标准测例输入运行学生程序
8   | 如果 学生代码成功运行并得到输出 则
9   |   | 对比代码输出与标准测例输出
10  |   | 记录评测报告
11  |   | 如果 输出一致 则
12  |   |   |  $grade = grade + \text{该测例的分值}$ 
13  $grade = grade * \text{程序修复补正系数}$ 
```

该算法解决了基本测试算法对答案的可编译性限制，可以应对程度较轻编译语法错误的答案的动态测试。然而该算法应用难点在于修复补正系数的计算，算法语义补正规则比较复杂，并且该算法无法解决因非编译性语法错误导致的结果偏差。

2.2 静态分析

静态分析，是指不运行软件或代码，对程序代码进行分析^[13]。在程序填空的评测上，应用方式为对学生提交的答案（语句或表达式）和标准答案进行分析，根据分析得出的处理结果进行对比，根据对比相似度得出分数。

2.2.1 编辑距离算法

编辑距离算法是一种经典的相似度比对算法。这种比对算法可运用于各种数据结构，比如基于树的相似度比对^[14]、集合的相似度比对^[15]。字符串到目标字符串的编辑距离就是计算从原字符串转换到目标字符串所需要的最少插入、删除和替换的数目，此算法是由俄国科学家 Levenshtein 提出的，故又叫 Levenshtein 算法。

下面给出编辑距离具体的数学定义^[1]：

定义 2.1 (编辑距离) 对于一个符号对 $(a, b) \neq (\lambda, \lambda)$ ，其中 a, b 都是长度为 0 或 1 的字符串，相对的，对 (a, b) 的编辑操作写作 $a \rightarrow b$ ，这样的编辑操作称为单位编辑操作 (elementary edit operation)。

单位编辑操作有三种，分别是插入，替换和删除。

定义 2.1 (编辑转换) 从 X 到 Y 的编辑转换是将符号序列 X 转换到 Y 所需的一串单位编辑操作序列 S 。

单位编辑转换可以被一个任意权函数 γ 加权，该函数对每个单位操作 $a \rightarrow b$ 赋予一个非负的实数值 $\gamma(a \rightarrow b)$ 。该函数可以通过 $\gamma(S) = \sum_{i=1}^m \gamma(S_i)$ 使扩展为编辑转换 $S = S_1 S_2 S_3 \dots S_m$ 。

符号序列 X 与 Y 之间的编辑距离为：

$$\gamma(X, Y) = \min \{ \gamma(S) | S \text{ is the edit transform from } X \text{ to } Y \} \quad (2.1)$$

表 2-1 给出了几个例子：

表 2.1 编辑距离的说明

| 字符串 1 | 字符串 2 | 采用操作 | 编辑距离 |
|--------|--------|------------------------------------|------|
| 123456 | 12456 | 插入，在字符串 2 的第 2 个字符后插入 3 | 1 |
| 12345 | 12346 | 替换，将字符串 2 的第 5 个字符替换为 6 | 1 |
| 123456 | 133466 | 替换，将字符串 2 的第二个字符替换为 2，第 5 个字符替换为 5 | 2 |
| 123 | 1235 | 删除，将字符串 2 的第 4 个字符删除 | 1 |

编辑距离算法的运行步骤如算法 2.3：

编辑距离算法是一种传统的相似度比对算法，多用于搜索以及主观题的评测以及抄袭检查等应用场景，对于程序填空题这种答案变化性比较小的题目评测效果良好，特

算法 2.3: 编辑距离算法^[16]

输入: 2 个待对比的字符串 $Str1$ 与 $Str2$
输出: 相似度 α

- 1 **如果** $Length1 = 0$ **则**
- 2 编辑距离 $EditDistance = Length2$
- 3 **如果** $Length2 = 0$ **则**
- 4 编辑距离 $EditDistance = Length1$
- 5 **如果** $Length1 \neq 0 \ \&\& \ Length2 \neq 0$ **则**
- 6 构造一个 $(Length1+1), (Length2+1)$ 大小的矩阵 $DistanceMatrix$, 矩阵的下标从 0 开始。
- 7 **对于所有** 矩阵 $DistanceMatrix$ 中的元素 **进行**
- 8 第 1 行与第 1 列从零开始, 以步长为 1 递增的编号。
- 9 **对于** $each \ i \in [1, rows \ of \ DistanceMatrix]$ **进行**
- 10 **对于** $each \ j \in [1, columns \ of \ DistanceMatrix]$ **进行**
- 11 **如果** $Str1[i] = Str2[j]$ **则**
- 12 $Distance = 0$
- 13 **如果** $Str1[i] \neq Str2[j]$ **则**
- 14 $Distance = 1$
- 15 $temp1 = DistanceMatrix[i-1, j] + 1$
- $temp2 = DistanceMatrix[i, j-1] + 1$
- $temp3 = DistanceMatrix[i-1, j-1] + Distance$
- $DistanceMatrix[i, j] = \min(temp1, temp2, temp3)$
- 16 编辑距离 $EditDistance = DistanceMatrix[Length1, Length2]$
- 17 相似度

$$\alpha = 1 - \frac{EditDistance}{Max(Length1, Length2)} \quad (2.2)$$

别是部分字符缺失或错误的回答。该算法的缺点是其具有语法无关性, 对于执行等效的答案无法识别出等价性。在一些复杂的相似度比对上, 对编辑距离的算法性能有更高的要求, 因此也出现了各种编辑距离的算法优化, 比如基于 q-gram 倒排索引的优化^[17]。

与文本类型的相似度匹配相关的算法还有基于属性论的相似度计算^[18]、基于语义分析的相似度计算^[2]、基于机器学习的相似度识别^[19]、基于语料库和字符串相似度的语义文本相似度计算^{[20][21]}等算法。这些算法大多用于搜索引擎的关键词搜索以及人工智能的语句分析之中。

2.2.2 基于编译原理的匹配算法

对程序语言的语法特性, 可采用编译原理的算法进行分析。编译原理除了用于编译程序外, 还用于一些程序的自动分析和优化, 程序竞赛的程序相似性检查^[22]。通过词

法分析和语法分析生成一定的信息结构数据，如程序的作用域树^[23]、程序依赖图^[24] 等。在程序填空题中，答案类型一般为语句和表达式，因此可以通过编译原理的词法和语法分析生成语法分析树进行关键的采分点比对^[25]。

基本的算法伪代码如算法2.4

算法 2.4: 基于编译原理的匹配算法

输入: 学生填空答案 *answer* 与标准答案 *standardanswer2*

输出: 相似度 α

- 1 对 *answer* 和 *standardanswer* 进行标准化处理
 - 2 对 *answer* 和 *standardanswer* 进行词法分析和语法分析，生成语法分析树
 - 3 对两棵语法分析树使用树匹配算法进行相似度计算，得出相似度 α 。
-

2.2.2.1 语句的类型

语句的类型可以进行划分，以 C 语言为例，可以划分为：

- 1 表达式语句和空语句
- 2 变量、函数等声明语句
- 3 函数调用语句
- 4 控制语句与其控制包含的 (1)、(2)、(3)、(4) 语句

2.2.2.2 基于语法分析树的语句处理

对于填入的答案和标准答案，在进行匹配之前，必须将标准答案以及待评答案转化为语法分析树的表现形式，并进行标准化减少语句多样性^[7]。

2.2.2.3 对语句的标准化

标准化是在指定的规则基础上，对提交答案和标准答案进行语义等价转换，目的是消除程序表达方式的多样性，统一标注进行对比。如变量定义语句、循环语句都有多种表达方式，标准化可以更好的对语句进行对比，减少语义的多样性。

2.2.2.4 对语法分析树的匹配评分

对标准答案和学生提交答案解析成语法分析树后，对两颗子树进行匹配。匹配的算法有许多，可以用上面所提到过的基于树的编辑距离算法，也有基于采分点的匹配算法。

2.2.2.5 对基于语法分析树的静态分析算法的评价

基于语法分析树的静态分析算法充分利用了程序题的语法特点，利用编译原理的语言分析将语句进行了语义化解析，使得比对更加准确科学。缺点是算法实现成本过高，生成语法分析树的规则、标准化的规则制定都需要大量的工作去检验其充分性，并且无法处理过于复杂的等价表达，如复杂的多项式等（如表达式 $(1+3)*(2+2+a)$ 和表达式 $(0+4)*(1+3+0.5a+0.5a)$ 的等价），对于更加复杂的表达式进行等价化的判断计算量过大）。该算法适合处理程序设计编程题的静态分析评测，但对于程序填空题来说，则显得过于复杂，性价比不高。

2.3 本章小结

本章介绍了可应用到程序填空自动评测的各种选型算法，包括动态测试和静态分析两方面的评测。在下一章会讲解本文的算法选型、评测算法以及算法在系统中的部署工作流程。

第三章 动态测试与静态分析结合的评测算法的实现与部署

在这一章节，会着重介绍本文的工作研究内容。中心内容为程序填空题评测的算法的设计。

本章将先给出算法的选型，然后提出算法的结合设计以及算法流程，之后详细说明算法在 Matrix 系统中实现的部署工作。

3.1 算法的选择

根据第二章介绍的算法选型，本文采用动态测试结合静态分析的评测思路，对第二章提出的算法进行选择和结合，产生独有的程序评测算法。

首先，动态测试的算法选型上，采取基本动态测试算法。基本动态测试算法对于答案简短的程序填空题效果良好，可直接使用 Matrix 原生评测系统功能实现。对于程序语法编译错误导致无法进行动态测试，或者部分错误导致结果误差的情况，可以交给静态分析去处理。

在静态分析的算法选型上，采用编辑距离算法。编辑距离算法相比于分析语义的结合编译原理的相似度比对算法，实现成本低，对应用的环境要求不大，不需要出题者进行复杂的出题设置，不需要构建大量的语法规则。考虑到实验时时间的紧迫性，简单的编辑距离算法更符合现实需求。另一方面，对于语言等价答案问题的处理，可通过设置多种标准答案以及动态测试部分评测来解决。

本文的研究重点在于如何将动态测试与静态分析相结合，使得算法评测的分数更加接近于人工的评测结果。

3.2 程序填空评测问题算法的实现

程序填空的评测功能实现的工作可以总结为以下五个方面：

- 1 评测流程的设计
- 2 评测算法的初步设计
- 3 系统交互方式的设计
- 4 以及题目数据的设计
- 5 各系统的部署

3.2.1 评测流程的设计

程序填空题的评测流程广义上包括从出题到做题评分获取分数的整个完整的过程，大体的流程分为：

- 1 出题
- 2 做题
- 3 评测
- 4 分数查询

3.2.1.1 出题

在 Matrix 系统中，题目以题库的形式存储在数据库以及文件系统，以便题目的多次复用。出题后，老师和教学助理可以选择将题库的题目发布为课程的作业或者是某场考试的题目。由于程序填空题包括动态和静态评测，因此出题需要提供的材料包括题干本身、挖空的程序代码（填上标准答案可以完美执行出标准结果）、标准填空答案以及相关的动态测试设置和相关的库代码文件。出题后，评测系统对标准答案进行一次动态测试，以验证标准答案的正确性。

3.2.1.2 做题

题目作业开始或者对应考试开始后，学生登录 Matrix 应用系统，进行题目的阅读并作答。此时若发布的作业或题目设置为评测定时（一般在考试场景），则提交后不会马上给出评分。

3.2.1.3 评测

评测时机开始时，评测系统会在数据库中查找出学生提交的答案，并经过程序填空自动评测算法的运行评测，得出评测分数，写入数据库。

3.2.1.4 分数查询

用户（老师、教学助理和学生）可通过系统查询权限内的学生的提交评测成绩。

3.2.1.5 评测算法的初步设计

评测算法采用动态测试的程序运行评测和静态的编辑距离算法相结合。

动态测试的技术实现在 Matrix 评测系统上已比较成熟，目前支持 c 和 c++ 语言的评测。通过评测系统原有的黑盒函数运行得出评测分数。

静态分析采用传统的编辑距离算法。下面为动态测试与静态分析结合的评测算法伪代码：

| |
|-----------------------------------------------------------------------------------|
| 算法 3.1: 程序填空评测算法 |
| 输入: 学生提交的填空答案数组 <i>Answers</i> |
| 输出: <i>FinalGrade</i> |
| 1 对于所有 填空答案数组 <i>Answers</i> 中的答案 进行 |
| 2 填入挖空的代码 <i>Code</i> |
| 3 对 <i>code</i> 进行动态测试，得出动态测试部分评分 <i>DynamicGrade</i> |
| 4 通过编辑距离算法相似度计算函数 <i>EditDistanceSimilarity</i> 静态分析部分得分 <i>StaticGrade</i> = |
| $\sum_{i=0}^n EditDistanceSimilarity(Answer_i, StandardAnswer_i) * score_i$ |
| 5 最终得分 <i>FinalGrade</i> = max(<i>DynamicGrade</i> , <i>StaticGrade</i>) |

3.2.2 系统交互方式的设计

程序填空题评测依赖于 Matrix 应用系统，该系统包括 web 应用端、评测系统端以及文件系统端、数据库及其它小型服务系统。web 应用、评测系统和文件系统和数据库是核心，它们之间通过 HTTP 协议或数据库连接协议进行交互。主要系统的交互联系如图3-1。

交互方式是根据算法与系统的应用结合，确定了交互方式，结合算法便可设计出各系统交互的具体数据内容。

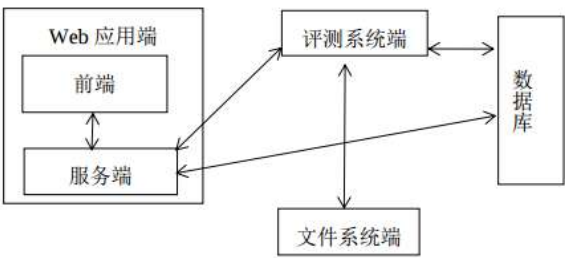


图 3-1 Matrix 内部子系统交互

- 结合图3-1的交互和评测算法的流程，成功的交互流程可以分为以下几个步骤:
- 1 老师或 TA 通过前端发送出题信息到服务端
 - 2 服务端将题目信息写入数据库，同时向评测系统发起评测请求，检验题目的正确性。

- 3 评测系统检验通过，向数据库写入题目数据。
- 4 老师或 TA 通过前端将题目发布到课程作业或者考试题目的请求发给服务端。服务端写入数据库，题目发布成功。
- 5 学生通过前端阅读题目，提交答案数据到服务端。
- 6 服务端将答案写入数据库，将填充了答案的完整代码发送到文件系统，并发送请求“通知”评测系统进行评测。
- 7 评测系统根据“通知”在数据库中找到对应提交答案代码的文件路径，再从文件系统中获取对应的答案代码进行评测，并将评测结果写入数据库。
- 8 用户（学生、老师、教学助理）通过前端获取题目的评测结果。

3.2.2.1 静态分析初期的交互设计产生的问题

初期开发设计中，因与评测系统开发者的交互对接问题，静态分析部分的逻辑实现写在了 web 服务端中。导致了动态测试和静态分析处理分离在了服务端和评测系统，因此产生了异步问题。根据算法3.1，题目最终的得分取动态测试和静态分析两者的分数中较大的分值，因此最终得分需要等待动态测试和静态分析部分评测的完成。由于这两部分的评测在服务端和评测系统分离运行，难以确定两者完成的时间并计算出最终得分。因此初期的策略为两部分的评测各自把分数写入数据库，当前端向服务端查询分数时，服务端从数据库中查看两部分的分数，进行计算得出最后得分并返回。

动态测试与静态分析的算法执行分离产生了两个问题，一是进行统计分数时无法直接读取数据库，必须进行二次计算。二是将评测部分的逻辑代码写入负责业务逻辑的服务端中不合规范。因此后期通过与评测系统开发人员协调，将静态分析部分的评测实现迁移到了评测系统，解决了异步处理问题，评测系统可以控制两者完成的时间并计算出最终分数写入数据库。

3.2.3 题目数据的设计

程序填空题基于 Matrix 系统原有的编程题数据结构（用于动态测试），增加以及程序填空特有的数据字段，当评测系统进行评测，会从数据库查询获取对应题目的数据，从中获取评测需要的数据字段，程序填空需要的数据字段包括动态测试部分和静态分析部分。

3.2.3.1 动态测试部分的数据设计

动态测试部分需要的数据字段为原有编程题数据结构中的字段，包括：

- 1 grading: 编程检查每个阶段的分数，包括编译阶段、静态检查、标准测试、随机测试、内存检查、Gtest 单元测试等。由于程序填空题不同于编程题，用户不需要输入完整的代码，执行代码的变化十分有限，不必要设置多方面层次的评测，因此默认只设置标准测试和随机测试的分数。
- 2 language: 编程语言，现阶段支持 c 与 c++。
- 3 standard_language: 标准程序的编程语言。
- 4 compilers: 对应不同语言的编译命令，该字段内容一般为固定，不需要用户设置。
- 5 limits: 程序运行限制的内存大小，该字段一般由出题者进行设置。由于填空题的变化有限，一般取设定好的默认值。
- 6 random: 随机检查程序的编译命令和随机检查次数，编译命令一般固定，随机检查次数由出题者进行设置。
- 7 standard: 程序题对应的依赖代码文件的名称，包括 support（依赖的库文件，显示在题目中）、hidden_support（依赖的库文件，不显示）、random_source（随机数生成器代码文件，用于随机测试）、standard_input（标准输入文件）、standard_output（标准输出文件）。
- 8 submission: 学生提交的代码文件名。
- 9 output_program: 编译提交程序生成可执行文件的名字。
- 10 entry_point: 标准程序入口。
- 11 standard_score: 题目评测部分的满分。
- 12 exec_flag: 测试程序执行命令时附加的参数。

以上是基本的程序评测需要的数据字段，但只有部分 grading、language、standard_language、random、standard、submission、standard_score 需要进行设置。其余一般去设定的默认值。

3.2.3.2 静态分析部分的数据设计

静态分析部分，也是程序填空题基于编程题增加的数据字段，包括以下字段：

- 1 standard_answers: 填空的标准答案数组
- 2 answer_scores: 填空的分值设置数组，说明每一个空占的分值。
- 3 code: 挖空关键填空的代码，其中需要填空的部分以字符串 \$blank 表示

3.2.3.3 程序提交记录的数据设计

学生提交答案时，服务端会将答案以提交记录的形式保存在数据库中，评测系统评测时会根据评测请求给出的提交记录 id，获取对应提交记录，再根据提交记录关联的题目获取题目信息进行评测，提交记录中学生的答案保存在 detail 字段中，在 detail 字段中，包含该字段：

student_answers: 学生提交的填空答案，为字符串数组。

3.2.4 各系统的部署

3.2.4.1 web 前端的部署

前端是 web 应用与用户交互的窗口，保持一个简洁明了舒适的前端界面，是一个重要的工作。前端的工作就是将题目数据可视化显示给用户，并收集用户交互的信息交给服务端处理。

程序填空题的用户交互中，前端部分核心为两个交互页面：出题页面和题目页面。

出题页面是老师出题的交互界面，分为题目、描述、挖空代码、填空标准答案、编程语言、题目相关文件、时间内存要求、动态测试评测各阶段评分权重、标准输入输出样例和随机测试文件这几个板块。板块页面图见附录图 A-1 到 A-7。

题目页面，包含问题描述、答案提交和成绩反馈板块，界面见附录图 A-8 至 A-10。

前端包括 HTML、CSS、Javascript 文件的编写，前端除去页面样式 (HTML、CSS) 外，需要在脚本代码 (Javascript) 完成获取用户填入的表单信息，当用户点击提交按钮，将用户输入的表单信息通过 http 协议发送到服务端中处理的业务。

3.2.4.2 web 服务端的部署

服务端需要完成以下几个任务：

- 1 接收前端发送的出题请求，将题目信息写入数据库，创建新题目。
- 2 接收前端发送的题目答案提交请求，将提交答案写入数据库，将填空答案填入挖空代码形成完整的程序发送到文件系统储存，并向评测系统发送评测请求。
- 3 接收前端发送的成绩查询请求，从数据库中查询出成绩并返回到前端。以上每个任务以对应的 URL 接口完成。

服务端使用 Node.js 环境编写，使用 express 框架搭建服务器，主要工作是以接口模块的形式完成上面的接口对应的逻辑处理。

Matrix 评测系统采用前后端分离的形式，对于页面、图片等静态资源的请求，使用 nginx 服务器进行转发处理，服务端处理数据方面的 api。对于程序填空题，增加了查看题目信息、提交答案、查询题目分数的 api。这些 api 对应的 URL 为：

- 1 查看题目信息（学生） /api/courses/:course_id/assignments/ GET
- 2 提交答案（学生） /api/courses/:course_id/assignments/submission POST
- 3 查询最后一次提交记录（学生） /api/courses/:course_id/assignments/submission/last GET
- 4 查看题库题目信息（老师、TA） /api/libraries/:library_id/problem/:problem_id GET
- 5 更新题库题目信息（老师、TA） /api/libraries/:library_id/problem/:problem_id POST

具体处理逻辑的伪代码见算法3.2：

| 算法 3.2: 服务端对程序填空题提交请求的处理 | |
|--------------------------|------------------------------------|
| 输入: 浏览器端的答案提交请求报文 | |
| 输出: 返回的处理响应报文 | |
| 1 | 检查报文提交格式 如果 报文格式正确 则 |
| 2 | 提取报文中学生填空答案。 |
| 3 | 获取题目信息。 |
| 4 | 将学生答案填入题目挖空代码，得到完整代码 |
| 5 | 生成新的提交记录，写入数据库。 |
| 6 | 将生成的代码发送至文件系统保存，与提交记录的 id 关联。 |
| 7 | 向评测系统发送评测请求。 |
| 8 | 向浏览器端发送成功提交的响应报文 |
| 9 | 否则 |
| 10 | 向浏览器端发送错误信息的响应报文 |

3.2.4.3 评测系统的部署

评测系统是部署程序填空评测系统的核心，主要的算法逻辑都在评测系统中执行，步骤如下：

- 1 首先评测系统接收到服务端发送的评测请求，根据发送的报文中的提交 id，查询数据库找到对应的提交，获取提交的答案。
- 2 执行静态分析，将提交的答案与标准答案使用前面所述的静态分析算法计算出静

态分析部分的评测得分。

- 3 根据提交的 id 从文件系统获取对应的动态测试的代码文件，进行动态测试，得出动态测试部分评测得分。
- 4 当两部分都评测完毕，取出两部分分数的较大分作为最终得分写入数据库。

3.3 程序填空评测的应用试验

3.3.1 试验结果分析和存在问题程序填空题的评测算法部署到 Matrix 应用系统后，在一场程序设计课程的期末考试中进行了应用试验，图3-2为该考试某填空题的题目描述。表??为部分的评测结果。

题目包含四个填空，其标准答案依次为 int*, int、int *,const int*, int, int、int*, int*、int*, int*。题目每空 2 分，总分 8 分。



图 3-2 考试题目描述

表 3.1 考试试验部分成绩统计

| 填空 1 答案 | 填空 2 答案 | 填空 3 答案 | 填空 4 答案 | 最 终 得 分 | 动 态 测 试 得 分 | 静 态 分 析 得 分 |
|-------------------|---------------------|------------------|--------------------|------------|-------------------|-------------------|
| Max | Max | Max | max | 0 | 0 | 0 |
| int *,const int*, | int *swap_num, | int*, int* | int *arr_start, | 3 | 0 | 3 |
| int, int | int *swap_num1 | | int *arr_end | | | |
| int [], int | int *, int [], int, | int *, int * | int*, int* | 7 | 0 | 7 |
| | int | | | | | |
| int*, int | int * max, const | int * a, int * b | int * const start, | 8 | 8 | 4 |
| | int * const arr, | | int * const end | | | |
| | const int start, | | | | | |
| | const int end | | | | | |

从评测结果统计看，可以看出问题如下：

- 1 动态评测对于大多数提交都是 0 分，没有起到评测作用。

从考生答案的程序评测报告分析,发现问题在于第二个填空,对应标准答案为 `int*,const int*, int, int`。提交给评测系统的挖空代码(也是标准答案的测试代码),因防泄露答案(隐藏代码中含有答案),实际上含有隐藏未公开给学生的部分,实际的挖空代码中未显示给学生的部分如3.1所示。

Listing 3.1 执行代码中未显示的部分

```
void input(int* p, int n) {
    printf("input is OK\n");
}
void findmax(int * a, const int*b, int n, int m) {
    printf("findmax is OK\n");
}
void swap(int* x, int*y) {
    printf("swap is OK\n");
}
void sort(int *, int *) {
    printf("sort is OK\n");
}
```

可以看出,隐藏的代码片段3.1是上面填空函数声明对应的定义实现,若两者的参数类型不严格相等,则会产生编译错误,而大多数考生的第二个填空的第二个参数漏了 `const` 声明,导致整个代码无法通过编译。因为一个填空的出错,导致所有的填空无法参与动态测试,这是初期程序填空评测算法的一个重大缺陷。

另一方面,从题目上看,对于数组作为参数, `int[]` 这样的参数类型是可行的,但是由给出的测试挖空代码来看,必须严格为指针类型才能视为正确答案。这是出题过程的一个疏漏。

3.4 本章总结

本章说明了本文的主要工作。首先是程序填空算法的选型和设计,该部分的中心思想是动态测试和静态分析算法的结合,如何将两部分的评测结果综合成一个最终得分。

然后是在 Matrix 评测系统中的实现部署,该部分内容的中心思想是算法与系统的结合,根据算法流程设计数据结构和交互的报文数据内容。

同时也指出了算法在试验中存在的缺陷问题,问题包括题目的设计和算法的设计

两个方面。在下一章，将会说明评测算法的改进方案。

第四章 填空评测算法的改进

上一章的结尾，指出了初期的程序填空评测算法的缺陷，在这一章，本文会详细分析算法出现的问题以及解决方案，然后阐述改进算法的过程，最后说明对于改进算法所做的系统代码的实现修改工作。

4.1 问题与改进算法

4.1.1 旧评测算法中的问题

通过总结之前算法在 Matrix 考试系统的部署和实践，总结出以下两点问题：

- 1 动态测试评测的整体限制，必须所有填空通过语法编译，动态评测才能运行。
- 2 对于一些出题，测试的程序有时无法应对一些等效答案。下面对这两点问题提出解决的改进方式。

4.1.2 评测算法改进方案

上一节分析了旧程序填空评测算法的问题主要在于动态评测的整体性限制上，因此本文在独立填空评测上做突破点，对题目中的每一个填空，都进行一次动态测试，具体的算法设计如算法 algo:ibc。

4.1.2.1 动态测试的改进

算法 4.1: 改进后的动态评测代码生成算法

输入: 第 n 个填空的答案 $answer_n$

输出: 该填空对应的评测代码 $code_n$

- 1 将挖空代码 $blankcode$ 赋值: $code_n = blankcode$
 - 2 对于所有 挖空的代码 $blankcode$ 中的填空位置 进行
 - 3 **如果** 该空为第 n 个空 **则**
 - 4 填入学生答案 $answer_n$
 - 5 **否则**
 - 6 填入标准答案 $standardanswer_n$
-

该算法对程序填空题提交的每一个填空，都生成了对应的一个提交代码，隔离了语法错误的答案对其它答案造成的影响，使得动态测试部分的评测更为科学。

4.1.2.2 静态分析的改进

一般的等效答案在填入挖空程序中执行，都会得到相同的结果。然而在图3-2 描述的题目与其测试结果测试的意图有所差异，因此出现了等效答案却无法得分的情况。面对这种情形，较科学的解决方法为提供多个等效答案进行静态分析，取最大相似度作为静态分析的评分依据，从而增大静态分析的准确率。

4.1.3 对算法改进所进行的工作

4.1.3.1 因多次程序评测造成的问题

对算法进行改进后，原本评测一次的动态测试过程变成了多次代码评测，这对于评测系统产生了一定的问题。一是评测系统原本的评测逻辑是一次提交对应一次代码评测，与程序填空题一次提交对应多次代码评测不符。二是评测系统只能识别当前评测的提交代码，无法将其与其它提交联系在一起，因此评测系统无法将几次评测得出填空得分同时获取并相加得出最终得分。由于评测系统的评测逻辑存在交互协作问题，暂时无法进行修改，因此改进工作的重点是在不对评测系统进行修改的情况下，完成对改进算法的适应性部署。

4.1.3.2 服务端对算法改进所做的适应性修改

评测算法改进后，服务端对前端的提交的动态测试部分的处理流程如伪代码??:

算法 4.2: 服务端对学生提交答案的处理

输入: 学生提交的填空答案数组 *answers*

输出: 该学生提交记录在数据库中的 *id*

- 1 **对于所有** 答案数组 *answers* 中的答案 **进行**
 - 2 按照算法4.1的描述，生成对应的评测程序，发送到文件系统保存，并在数据库中生成提交记录。
 - 3 请求评测系统对该提交记录进行动态测试评测。
 - 4 生成一个总提交记录，内容为之前每一个答案生成的提交记录的 *id*。
-

由于评测系统无法识别来自同一个程序填空题的多个填空对应的提交，因此无法计算动态测试总分。因此计算总分的工作放在服务端进行，算法同伪代码4.3

该逻辑修改解决了评测系统对程序填空评测的改进的不适应的问题，然而缺陷在于需要经过服务端的二次查询和计算才能得出动态测试部分的评测得分，在统计多个学生的考试成绩时会有较大的代价消耗。

算法 4.3: 改进后服务端计算动态测试部分得分

输入: 要查询程序填空题成绩的学生学号 id

输出: 该学生的动态测试得分 $grade$

- 1 根据学号 id 在数据库中查出学生最后一次的提交记录 $finalsub$ 。
 - 2 根据 $finalsub$ 中记录的填空提交记录 id 从数据库中获取每个填空对应的提交记录, 得到提交记录数组 $submissions$ 。
 - 3 动态测试得分 $dgrade = 0$ 。
 - 4 **对于所有** $submissions$ 中的每一个提交记录中的得分 **进行**
 - 5 $\quad grade = grade + dynamicgrade/100 * \text{该填空分值}$
-

4.1.4 另一个问题

虽然分次动态测试解决了动态测试的整体性问题, 但是另一个问题在于动态测试划分的粒度。比如某程序填空题, 它的第一个填空答案为 $inta = 5;$, 第二个填空答案为 $cout << 2 * a$, 并且之后的非填空程序部分都没有这两个变量的引用。像这种情况, 其实只要保证两个填空的变量都是 a 即能得分。

然而现有的动态测试并不能保证这种情况的正确评分。原因是动态测试是每次取学生提交的一个填空答案, 其他填空都填上标准答案, 以上面为例, 并不能保证学生填写的第一个填空和标准答案的第二个填空是同一个变量, 这样反而会导致学生的两个填空都无法正确得分, 尽管学生的答案是正确的。

要克服这个问题, 一种方案是对每种可能出现的情况都进行评测, 问题就变为: 将 n 个填空分成 m 组 ($n \leq m$), 每组至少一个填空, 将所有分组的可能情况计算出来, 对于每种情况, 都进行一次动态分析处理, 处理方式同算法4.1, 只是答案的填空和替换是以组为单位的。计算出每一组的动态分析得分后, 取最高分为动态分析得分。

然而该方案的另一个缺点在于分组情况数过多, 根据排列组合原理, 对于 n 个填空的分组情况数是随着 n 的增多呈爆炸式增长的, 并且对于每种情况进行一次改进动态分析, 就要执行分组数 m 次的测试代码。这对于评测系统的压力是巨大的。并且对于一道程序填空题, 实际上产生关联的答案组并不是很多, 很多时候的分组其实没有任何意义的。

因此得出了让出题者设定关联填空答案的第二种方案, 在出题时, 由出题者根据题目答案自行进行分组设定, 动态分析时以出题者的分组进行评测。但是该方案增加了出题者的工作量, 增大了出题难度, 需要出题者仔细进行思考, 用户体验较差。由于时间以及成本原因, 并未在后面的实验中采用该解决方案。

4.2 本章总结

本章阐述了对程序填空算法的改进，以及对应在服务端的适应性修改，最后对改进后的算法进行了分析。下一章会对改进的算法与初期算法进行实验性对比分析。

第五章 改进后的仿真实验结果与分析

本章主要阐述对程序填空评测改进算法与初期算法的仿真实验的部署、运行结果展示和结果分析。

5.1 实验环境和参数设置

5.1.1 实验环境

Linux 系统笔记本，安装 Node.js 服务器运行环境 (运行本地服务端)。测试数据库服务 (服务器)、测试评测系统服务 (服务器)。

5.1.2 参数设置

- 1 以图3-2的题目作为实验题目，从数据库提取考试中学生的提交作为实验样本。
- 2 根据上章服务器对学生提交答案的处理算法，对实验样本的提交数据进行处理，批量提交评测请求到评测系统。评测完毕后，批量查询分数进行统计。
- 3 对比参照组为两组，分别为使用改进前评测算法（动态测试评测只提交一次程序评测）和改进后评测算法（每个填空答案对应一次评测）。
- 4 对比结果为两组算法得出的动态测试评测得分。

5.2 结果图表与具体分析

图??与图5-2 为使用考试学生提交数据为实验样本的动态评测结果的结果折线图和为同样的实验中动态测试评测算法改进前与改进后的总得分折线图。



图 5-1 改进前与改进后的动态测试评测分数对比 (横轴为学生, 纵轴为得分, 满分为 8 分)

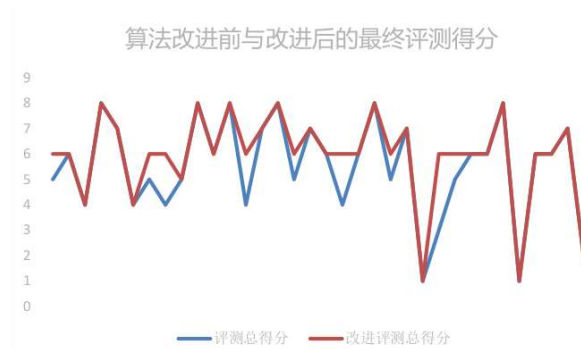


图 5-2 算法改进前与改进后的评测总得分

5.2.1 结果分析

从动态测试评测的结果图表来看，动态测试的评测的评分变化幅度很大，许多原本评分为 0 分的在改进算法后都有了较高的分数。这是除去第二空中 `const` 声明缺失造成的整体的编译不通过的影响后的改进结果。将每个空的学生答案独立与其它标准答案结合在一起进行评测，即使出现编译错误，也只会影响到出现语法错误的该空的分值，不会影响到其它填空的评分。

而从最终得分的结果图表来看，动态测试部分评测分的提高带动提高了不少学生的最终得分，使得最后的评分结果更为合理，动态测试也起到了应有的评分作用。

第六章 动态测试与静态分析结合的评测算法的实现与部署

6.0.1 工作总结

本文结合程序评测与文本相似度算法用于程序填空题型的评测，是一种新的尝试。尤其在考试试验后对程序评测部分做的进一步改进大大提高了评测的效果。与他在相关领域的工作相比，本文由于时间、成本、应用需求等原因，没有采用语义分析、语法修复等算法进行实验，这是一点遗憾和缺陷之处。但本文在算法与工程的结合方面作出了一定的努力，这是一点可取之处。

6.0.2 系统的不足和改进方向

在程序填空算法应用到 Matrix 应用系统的工作中，职责的分离度仍不够高。比如改进后的动态测试评测算法中，对一道题需要向评测系统发起多次的提交评测，这是比较不合理的。因为这属于评测逻辑的一部分，按照职责粒度的划分，应该有评测系统收到评测请求后运行多次程序计算出动态测试评测部分的总得分，而不应该由服务端去进行多次评测请求和发送。然而无法这样做的原因是评测系统本身内部的逻辑结构难以改动、改动成本高以及和评测系统开发者的对接沟通等问题。

另一方面，静态分析部分使用编辑距离算法计算相似度仍有缺陷，一是编辑距离算法是语义无关的算法，无法抓住答案的关键得分点在哪里，一些无关紧要的相似部分也会被编辑距离算法所检测而算入得分（如括号，分号，运算符等非重点或无意义的字符）。要想在静态分析部分的评测效率更近一步，如何设计语义化的评测以及配合语义化的评测算法的交互流程是关键。

参考文献

- [1] MARZAL A, VIDAL E. Computation of Normalized Edit Distance and Applications[J]. Pattern Analysis and Machine Intelligence IEEE Transactions on, 1993, 15(9): 926–932.
- [2] MIHALCEA R, CORLEY C, STRAPPARAVA C. Corpus-based and Knowledge-based Measures of Text Semantic Similarity[C] // National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, Usa. 2006: 775–780.
- [3] HU H, ZHENG K, WANG X, et al. GFilter: A General Gram Filter for String Similarity Search[J/OL]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(4): 1005–1018. <http://dx.doi.org/10.1109/TKDE.2014.2349914>.
- [4] LU W, DU X, HADJIELEFTHERIOU M, et al. Efficiently Supporting Edit Distance Based String Similarity Search Using B +-Trees[J]. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(12): 2983–2996.
- [5] AYAD L A K, PISSIS S P, RETHA A. libFLASM: a software library for fixed-length approximate string matching[J/OL]. BMC Bioinformatics, 2016, 17: 454:1–454:12. <http://dx.doi.org/10.1186/s12859-016-1320-2>.
- [6] BARNARD D T, HOLT R C. Hierarchic syntax error repair for LR grammars[J]. International Journal of Parallel Programming, 1982, 11(4): 231–258.
- [7] HATTORI N, ISHII N. A method to remove variations in source codes[J]. Information & Software Technology, 1996, 38(1): 25–36.
- [8] GOUES C L, DEWEYVOGT M, FORREST S, et al. A systematic study of automated program repair: Fixing 55 out of 105 bugs for each[J], 2012, 8543(1): 3–13.
- [9] WANG A, ZHANG Z. An Integer Programming-Based Bound for Locally Repairable Codes[J/OL]. IEEE Trans. Information Theory, 2015, 61(10): 5280–5294. <http://dx.doi.org/10.1109/TIT.2015.2472515>.
- [10] MCBURNEY P W, MCMILLAN C. An empirical study of the textual similarity between source code and source code summaries[J/OL]. Empirical Software Engineering, 2016, 21(1): 17–42. <http://dx.doi.org/10.1007/s10664-014-9344-6>.
- [11] JIANG M, CHEN T Y, KUO F, et al. A metamorphic testing approach for supporting program repair without the need for a test oracle[J/OL]. Journal of Systems and Software, 2017, 126: 127–140. <http://dx.doi.org/10.1016/j.jss.2016.04.002>.
- [12] 玄跻峰, 任志磊, 王子元, et al. 自动程序修复方法研究进展 [J]. 软件学报, 2016, 27(4): 771–784.
- [13] 张健. 精确的程序静态分析 [J]. 计算机学报, 2008, 31(9): 1549–1553.
- [14] BILLE P. A survey on tree edit distance and related problems. Theor Comput Sci 337(1-3):217-239[J]. Theoretical Computer Science, 2005, 337(1-3): 217–239.
- [15] 林学民, 王伟. 集合和字符串的相似度查询 [J]. 计算机学报, 2011, 34(10): 1853–1862.
- [16] 杜利峰, 牛永洁. 字符串相似度在自动评分系统中的应用 [J]. 电子设计工程, 2011, 19(7): 42–44.
- [17] 王金宝, 高宏, 李建中, et al. 外存中高效的字符串相似性查询处 [J]. 计算机研究与发展, 2015, 52(3): 738–748.

- [18] 潘谦红, 王炬, 史忠植. 基于属性论的文本相似度计算 [J]. 计算机学报, 1999, 22(6): 651–655.
- [19] BILENKO M, MOONEY R J. Adaptive duplicate detection using learnable string similarity measures[C] // ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2003: 39–48.
- [20] ISLAM A, INKPEN D. Semantic text similarity using corpus-based word similarity and string similarity[J]. ACM Transactions on Knowledge Discovery from Data (TKDD), 2008, 2(2): 10.
- [21] MIHALCEA R, CORLEY C, STRAPPARAVA C, et al. Corpus-based and knowledge-based measures of text semantic similarity[C] // AAAI: Vol 6. 2006: 775–780.
- [22] WHALE G. Identification of Program Similarity in Large Populations[J], 1990, 33(2): 140–146.
- [23] COLIN A, BERNAT G. Scope-Tree: A Program Representation for Symbolic Worst-Case Execution Time Analysis[C] // Euromicro Conference on Real-Time Systems. 2002: 50.
- [24] KRINKE J. Identifying Similar Code with Program Dependence Graphs[C] // Reverse Engineering, 2001. Proceedings. Eighth Working Conference on. 2001: 301–309.
- [25] WANG J T-L, ZHANG K, JEONG K, et al. A system for approximate tree matching[J]. IEEE Transactions on Knowledge and Data Engineering, 1994, 6(4): 559–571.

致谢

感谢论文导师万海老师，在程序填空评测工作的设计工作上给我不少的思路和灵感，给出了我参考论文的查找方向，也提供了考试应用试验与应用的环境。没有老师的指导，就没有这篇论文的完成。

感谢参与相关研究的李天培同学提供了编辑距离算法的模块代码。

感谢魏传柳同学在程序评测算法部署到评测系统中的工作中给予我的工作交流和配合部署。

最后，向所有对我的工作给予支持的师长、同学和家人致以由衷的感谢和祝福！

王嘉威

2017 年 4 月 29 日

附录 A 补充更多细节

Title

Description









H₁ H₂ H₃ B I        

图 A-1 题目与描述

Code

请输入需要填空的代码，填空处用“\$blank”表示:

1

图 A-2 挖空代码

Stand Blank Answers

请依次填入空内的标准答案和分数：

Blank 0

请填入标准答案

请填入该空分数

图 A-3 标准填空答案和分值

Language

c

Code Files

(Invisible) Answer Files(必填) [+Choose a file](#)

Visible Support Files [+Choose a file](#)

Invisible Support Files [+Choose a file](#)

Entry Point

standard_main.exe

图 A-4 题目相关文件上传

Time-limit

1000

ms(500-5000)

Memory-limit

32

MB(1-256)

图 A-5 时间和内存限制

Total 100

Compile Check

0

%

Static Check

%

Standard Check

%

+Random Check

%

times

GTest Check

%

Memory Check

%

图 A-6 动态测试评测各阶段分值设置

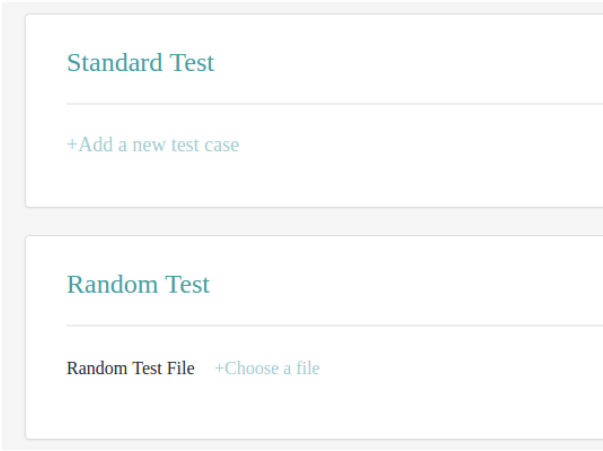


图 A-7 标准输入输出测试样例和随机测试文件

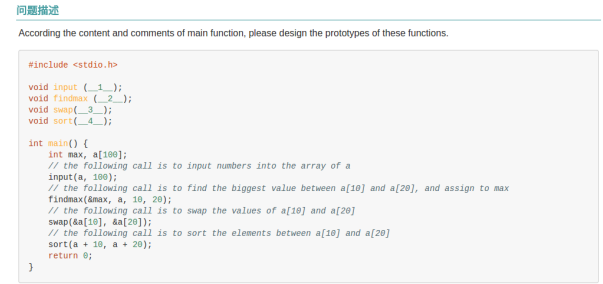


图 A-8 程序填空题题目描述

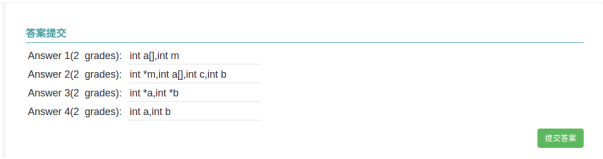


图 A-9 程序填空答案提交