北京郵電大學

本科毕业设计（论文）

题目：基于控制流分析的代码反抄袭系统

姓　　名　马庆元

学　　院　计算机学院

专　　业　网络工程

班　　级　2008211314

学　　号　08211543

班内序号　6

指导教师　房鸣

2012 年 6 月

北 京 邮 电 大 学
本科毕业设计（论文）任务书

学院	计算机学院	专业	网络工程	班级	14
学生姓名	马庆元	学号	08211543	班内序号	6
指导教师姓名	房鸣	所在单位	实验中心	职称	副教授
设计(论文)题目	基于控制流分析的代码反抄袭系统				
题目分类	1、工程实践类□ 研究设计类■ 理论分析类 □（1、2 类中各选一项在□内打 √） 2、软件■ 硬件□ 软硬结合□ 非软硬件□				
题目来源	题目是否来源于科研项目 是□ 否■				

主要任务及目标：

本项目旨在提供一套完整的解决方案，通过在控制流层面上分析学生提交的作业程序代码，发现可能的抄袭行为。该解决方案需要充分利用程序设计实践作业内容的特殊性，得到相对现有的通用反抄袭工具更高的准确/召回率。

主要内容：

1. 设计方便易用的接口工具和 API，可以连接现有的实践平台。
2. 设计高精确度，高召回率的基于控制流分析的代码比对算法，要求可以抵抗常见的反-反抄袭手段，将抄袭成本提高到抄袭收益之上。
3. 设计友好的操作平台，方便助教人员提取、整理分析数据，为检举抄袭提供充分依据。

主要参考文献：

1. Compilers: Principles, Techniques, and Tools (2nd Edition)，Sept ’06, Alfred V. Aho, Monica S. Lam, Ravi Sethi , Jeffrey D. Ullman, ISBN 0321486811
2. Introduction to Graph Theory (2nd Edition), Sept ’00, Douglas B. West, ISBN 0130144002
3. GCC, the GNU Compiler Collection, http://gcc.gnu.org/

进度安排：

第 1－3 周（2.21－3.11）：理解任务需求、对要开展的工作进行预调研；

第 4－6 周（3.12－4.1）：完成方案设计；

第 7－13 周（4.2－5.20）：完成调研工作，对调研的内容给出研究型结论；

第 14－16 周（5.21－6.12）：完成论文的编纂工作；

第 17－18 周（6.13－6.24）：准备论文答辩；

指导教师签字		日期	2012 年 2 月 日

北 京 邮 电 大 学

本科毕业设计（论文）诚信声明

本人声明所呈交的毕业设计（论文），题目《基于控制流分析的代码反抄袭系统》是本人在指导教师的指导下，独立进行研究工作所取得的成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

摘要

在现在的计算机科学实践教学中，程序设计类作业相对传统的手工作业所占的比例越来越大。由于程序代码的智力密集性和学生数目的膨胀，人工检查代码变得越来越难，并且随着学生的反-反抄袭手段逐渐增强，原有的反抄袭系统已经不能侦测出所有的不端行为。在这种背景下，我们急需一种新型的抄袭检测系统，可以避开学生反-反抄袭手段的影响，快速有效的找出抄袭的证据，并且要求设置简单，易于用户使用。

据此，本文工作设计并实现了一个满足上述要求的新型抄袭侦测系统。该系统的特点在于摈弃文本层面上检测抄袭行为，而改为基于分析代码的控制流图和执行时期的程序行为来判断互相抄袭的行为。比较起传统的抄袭侦测系统，本系统可以在算法逻辑层面上判断代码间是否具有抄袭迹象，这给学生的反-反抄袭活动带来很大的难度，提高了学生逃避审查的成本，从而更有效的减少抄袭现象的发生。

本文详细的介绍了本系统所涉及到的技术背景，细致的描述了该系统的原理，结构，核心算法及其分析，输入、中间数据和输出数据的格式，工具集中的工具的用途，运作原理和具体实现，和相关外围程序的二次开发过程，以及整个系统在特定测试环境下的性能和准确率表现。

关键词 反抄袭 代码意图分析

ABSTRACT

In modern computer science practicing teaching progress, comparing with traditional hand-writing homework, the rate of homework on programming increase fast these years. Checking the codes manually is becoming more and more difficult due to intellectual-intensive code and the increasing number of students. And with the stronger ability students have to avoid anti-cheating checking, our original system can no longer detect all of these misconduct. So, we are in urgent need of a new plagiarism detection system, which can fight against the anti-anti-cheating means, quickly and efficiently find the evidence of plagiarism, and easy to set up, easy to use.

According to these, this paper design and implement a new plagiarism detection system to meet the above requirements. This system does not detect plagiarism in the literal level, but based on the analysis of the control flow graph and run-time behavior of the code to judge the means of cheating. Compared to traditional plagiarism detection system, the system can determine the cheating intent in algorithm level, which increase the difficulty of anti - anti-cheating activities from the students ,and raise the cost student spend for avoiding detected by anti-cheating system. These can finally decrease the occurrence of plagiarism.

This paper provides detailed descriptions of the system, about technical background, principle of the system structure, core algorithm and its analysis, formats of the input ,intermediate, output data, usage, design principles and implement of the tools in the tool set, and deep development for external tools, as well as the performance and accuracy this system achieves in specialized environment.

Keyword: anti- plagiarism, intent analysis of source codes

目 录

第一章 背景介绍

1.1 项目背景

在计算机科学实践教学中，程序设计联系是重要的一环。随着实践教学比重的逐渐增加，程序设计类作业相对传统作业所占的比例越来越大。伴随互联网络技术和信息技术的发展，学生提交的电子版程序的抄袭变得前所未有的方便，并且由于程序代码的智力密集性，助教难以利用较少的劳动和时间发现抄袭行为，这在一定程度上对影响了实践环节的教学效果。因而，设计一个可以帮助助教分析学生的作业代码，快速搜寻抄袭证据的系统，对于揭发抄袭行为，吓阻抄袭意图，保证实践教学的效果而言，非常重要。本次研究旨在提供一套完整的解决方案，通过在控制流层面[1]上分析学生提交的作业程序代码，帮助助教发现可能的抄袭行为。

目前，学界已经研发了一系列反抄袭系统，这些系统可以有效解决相当一部分抄袭问题。但是上述通用系统在处理非常具体的反抄袭任务时，往往不能充分利用任务的特殊性质和额外资源，从而难以达到期望的反抄袭效果。本次研究将着力发掘利用程序设计实践中的特殊性，针对其设计算法，充分利用这些性质，缩减计算资源需求，提高准确性和召回率。

另外，针对目前已有的各种反抄袭系统的弱点，学生很容易设计出易于操作的反-反抄袭策略，削弱反抄袭系统的效用。本次研究将调查学生常见的代码反-反抄袭的策略，有针对性的加强反抄袭系统，同时对已有的反抄袭算法本身进行算法理论分析，找出其弱点，尽量在新系统中予以解决，使新系统可以作为已有系统的补充选项。

1.2 技术背景

该系统分为两部分：数据采集子系统和数据分析子系统。数据采集子系统负责处理用户给定的数据集，执行待判断的程序代码，整理控制流数据并将其封装；数据分析子系统分析数据采集系统的输出数据，运用多个算法（用户可选择，可在速度和精确度中均衡）比较处理过的控制流数据，并输出抄袭嫌疑报告。

整个解决方案绝大部分是利用一种新型的脚本语言 Python[8]实现的。python 是一种通用的高级编程语言，设计的重点包括优化代码可读性和易用性。Python 具有非常严格的语法要求，并且拥有一个非常庞大完整的标准库和扩展。python 支持多种编程范式，主要但不限于面向对象式，命令式和函数式。并且 Python 是一个支持自动内存管理的动态语言，有着非常复杂的自省（类似.Net 的反射）功能。在大多数情况下，Python 被用作撰写脚本，但依然有很多复杂的应用软件是用它写成的。利用一些转换器，Python 代码可以嵌入其他编程语言，或是转换为其他语言的源代码或中间表示形式。本解决方案使用了 Python 的 C 语言实现 CPython，这是一个开源自由软件。

在解决方案中两个子系统的链接部分采用的嵌入式数据库 SQLite。SQLite 是一个满足 ACID（原子性，一致性，隔离性和持久性）的关系数据库管理系统，不像很多利用了客户端/服务器模型的数据库，SQLite 在运行时没有自己的服务器或客户端的独

立实例，SQLite 一般作为一个共享库，或是直接以 C 源代码的形式，嵌入到利用 SQLite 作为数据存储模块的应用程序中，成为应用程序的一部分。应用程序在处理存储任务时，直接调用 SQLite API，这样可以大幅度节约 SQLite 与其他模块的交互时间，提高应用程序使用数据库功能时的性能，同时降低了应用程序作为解决方案部署的工作难度。SQLite 将整个数据库，包括表，索引，数据，存储过程等等直接存储在一个单一的文件或是内存的一个临时的区域内。SQLite 通过在写入数据库时锁定整个文件来保证并发正确性。在数据库管理器的实现上，SQLite 实现了除触发器，视图写入，和添加删除行这个 ALTER TABLE 以外的大部分 SQL92 标准。为了让更多的用户享受到 SQLite 的功能，很多编程语言都包含了 SQLite 的包装版，包括 C++，C#， java，PHP， Python 等。本解决方案使用了 SQLite 的 Python 包装版本 PySqlite。

本解决方案需要对源代码进行静态的和运行时的分析，静态分析所用的工具是 GCC(GNU 编译器集合，GNU Compiler Collection)。GCC[3]是在 GNU 计划中编写的编译器，是 GNU 计划中大多数软件的标准编译器，也是现代类 unix 的软件的事实标准编译器。GCC 可以运行于非常多的系统平台，包括 ARM，x86，PowerPC，System Z 等平台和 Linux， BSD 及各种 UNIX 实现版本，并且也能为很多系统平台生成目标代码，包括但不限于上述平台和系统。原始的 GCC 只能编译 C 语言，在 80 年代末，GCC 逐渐支持编译 C++， Ada， Fortran， Java 等语言，全称也由 GNU C Compiler 改为 GNU Compiler Collection。为了兼顾各种平台和各种语言，GCC 使用了模块化的结构，分为前端，中间段和后端。被编译的代码首先进入前端，GCC 在确定语言后调用相应的语言模块将原始代码文本解析处理为中间代码，此中间代码独立于语言，只表示控制流关系。中段负责读入前端生成的语言无关表示代码，进行各种分析，优化工作，最后本段输出利用临时变量的 3 地址代码。后端因机器不同而不同，不同的体系结构和操作系统的后端模块均是特制的。本段将中段生成的 3 地址代码转换为目标平台上的二进制代码。本解决方案利用 GCC 编译代码并且利用前端将代码基本块的信息嵌入目标代码，为后续的测量执行提供信息。

在分析数据之前，原始数据要经过 GCOV[7]的处理。GCOV（GNU 覆盖率测试器）是一个代码覆盖率测试程序。配合 GCC，利用这个软件，用户可以得知针对特定输入的代码覆盖率情况，包括每行的覆盖次数，分支覆盖次数，转移概率，函数调用次数，块执行时间等信息。该程序可以帮助用户利用这些信息“提示”编译器优化方向，以便构造更有效率的二进制代码。在使用时，用户需要利用 GCC 编译目标程序，并利用特别的命令嵌入覆盖率测试代码，然后用户按照测试集执行代码，获得一系列覆盖率输出，最后 GCOV 负责处理这些输出，给出人类可读的报告，本解决方案修改了 GCOV，使其输出特定的报告形式，方便后端软件处理。

1.3 毕业设计概况

我完成毕业设计的过程主要分为 5 个阶段：

第一阶段：在这个阶段，我根据任务书明确任务的需求，对整个系统结构进行了解，确认用户的需求，和系统的实现目标。之后，根据用户需求和系统设计要求，我

查阅了相关的资料，并收集一些测试数据，手动分析其特征，尝试大致的分析方案，并且学习开发该解决方案的若干软件技术。

第二阶段，在这个阶段我开始对系统进行总体的设计和调研，撰写关于系统框架的文档，尽可能的了解系统的大致框架。并且根据任务的要求，尝试若干种已有的方案，详细分析优劣，希望对自己的算法设计有所帮助。

第三阶段，我开始主要模块的具体实现。在设计的过程中我实现了自己的算法，并利用真实的测试集进行测试，尝试优化改进自己的算法。在设计的过程中特别注重软件的模块化，开发时考虑了降低模块的耦合度，提高了可维护性。

第四阶段，本阶段我开始编写开始软件代码，并完成代码的测试工作。

第五阶段，撰写论文。

第二章 设计目标

2.1 系统结构

该系统分为两个部分（如图 2-1 所示）：数据采集子系统（数据归一化模块）和数据分析子系统（代码雷同检查器），数据采集子系统接受从在线评测系统或其他需要反抄袭检测的教学系统中收集数据，数据分析子系统对代码进行分析，向需要反抄袭检查结果的教学系统提供抄袭嫌疑报告。由于该系统注重于检测算法的实现，在实际运用中，输入和输出都相对简单。数据采集子系统分为两个模块，数据导入模块和数据预处理模块。数据导入模块可以选择两个不同的来源：文件集合和数据库。文件集合是用户在存储设备上设置的一系列反抄袭检测所需要的文件，包括待测试程序的源代码，每一个题目所需程序的参考测试数据，还有程序基本情况的清单。数据库指的是在某一特定的关系数据库中的若干张表，这些表存储了待测试程序的源代码和每一个题目所需程序的参考测试数据。用户可以根据实际的使用场景自由的选择其中一个输入方式。无论是哪种输入方式，输入处理模块都会分析数据，将其转化为统一的输出格式供代码分析模块使用。

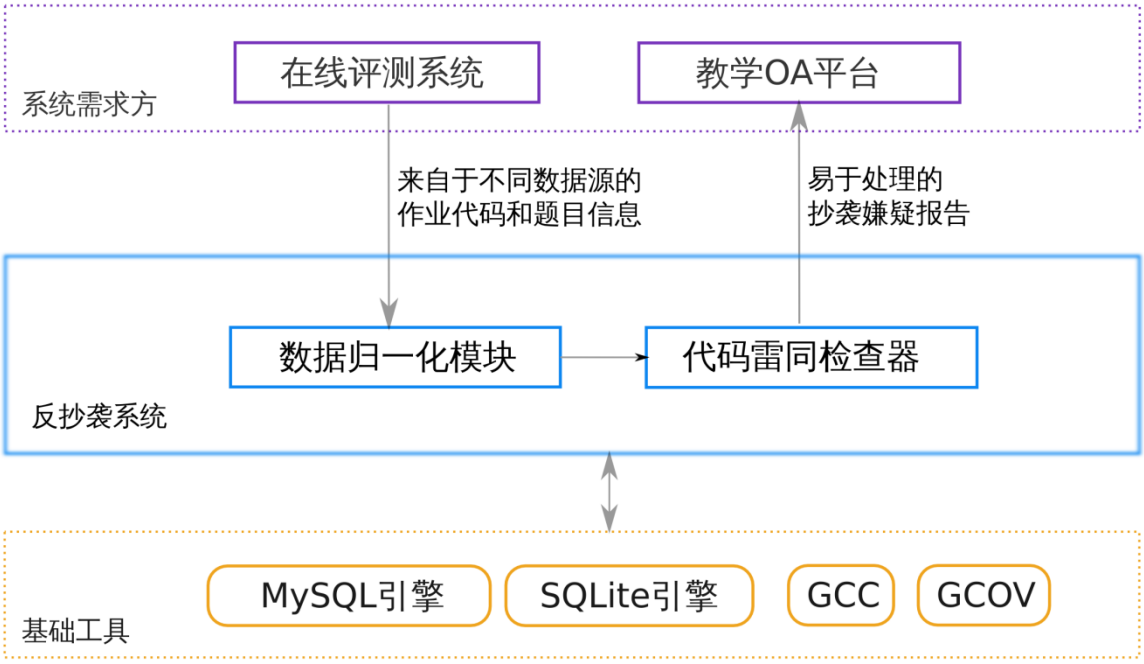


图 2-1 系统结构图

数据预处理模块包含了代码编译、执行、分析模块。该模块负责调用合适的编译器编译用户提交的待检测的源代码，按照分析器要求的方式，生成可执行的二进制文件，并由执行子模块设置执行环境，连接该源代码对应的问题的参考测试数据，用该数据作为输入，执行二进制文件，生成原始覆盖率分析报告，最后报告分析模块识读原始覆盖率分析报告，将其转换为本解决方案下一环节可以识别的输入文件。

数据分析子系统分为两个部分，控制流分析模块和报告模块。控制流分析模块用来依靠分析控制流信息找出可能的抄袭证据。该模块接受上一步的输入文件，经过分析计算后给出输入源代码代码之间的相似度。该模块分为两个子模块：数据结构处理

和匹配器。数据结构处理子模块将输入文件转换为抽象的数据结构（控制流图及其附加信息），匹配器负责匹配两两匹配已转换的数据结构，并且给出相似度，匹配器因有多种选择，可以允许用户在执行资源消耗和匹配准确度上权衡选择。报告模块负责将匹配数据排序整理，输出文本报告，报告最有可能互相抄袭的源代码对。

另外，本解决方案支持如下几个额外功能：

1. 支持一次检测处理多道题目，代码比对器仅比对在同一题目下的所有代码，并且在输出报告时可以根据题目分类按相似度大小排序。
2. 支持用户名标记，代码比对器会参考用户提交的所有代码，但仅比对不同用户间的代码对，输出报告也会忽略同一用户提交的代码之间的相似状况。
3. 作为基于控制流的代码比对算法，本解决方案可以在一定程度上支持侦测跨语言抄袭。

2.2 输入数据

在使用该系统之前，用户需要做如下准备：

1. 收集并整理题目数据。首先用户需要指定参与反抄袭检查的题目，用户需要给出题目 ID（为一整数，如果需要的话，用户要自己维护其他类型编号和该 ID 之间的一一对应关系）和每一道题目的参考测试数据集。对参考测试数据集的要求如下：
    a) 测试数据集必须通过标准输入（c 中的 stdin，unix 的 0 号管道）输入，并且代码在执行过程中不依赖除该测试数据外的任何其他输入的数据，另外，该测试数据必须能够使正确的程序代码得到正确的输出。
    b) 测试数据集要尽可能保证覆盖到大多数代码的全部执行分支，并且完整的考虑题目中的每一个常规实现角度，特殊情况和边界情况。
    c) 数据量要足够，对题目涉及的每一种情况要有大量的，但不等量的对应测试部分。
    d) 设计测试数据集时，要考虑让大多数代码要在较短的时间内运行完成，因为在抄袭检测的过程中每一份代码都要在打开分支覆盖测试的情况下运行一次，这会比运行常规编译条件下给出的可执行代码慢 3-10 倍。
2. 收集并整理代码数据。首先，用户需要从自己的考试系统中圈定并导出所有的待测试代码，并对每份代码标注所属的题目 ID 和用户 ID（为一整数，如果需要，用户需要自己维护其他类型的编号和 ID 之间的一一对应关系），和代码的程序设计语言类型。在抄袭检测的环节，代码比对系统只会比对相同题目，不同用户间的代码。对提交检测的代码要求如下：
    a) 可以编译通过，并且在规定的时间内正确的运行参考测试数据。
    b) 必须可以用默认编译指令编译。如果需要使用自定义的编译指令，例如加入其他的连接库，可以在解决方案中的脚本内修改，这样的操作比较复杂。

c) 不能有恶意代码，本解决方案假设代码在运行过程中，除了标准输入，标准输出，标准错误这三个方式（即 unix 的 0，1，2 号管道）以外不与系统产生任何交互。为安全起见，用户需要在运行该系统之前认真检查代码的操作意图。

在完成上述数据的准备时，就可以开始以制定格式导入数据了，本解决方案需要以下的输入数据：

1. 题目信息，包含每道试题的 ID 和测试数据。
2. 待检测的源代码信息，包含每份源代码的代码 ID，程序设计语言类型 ID，用户 ID，题目 ID，和源代码的文本

以上数据可以有 2 个不同的来源，根据用户的实际情况，如果选择从磁盘文件输入，需要提供文件清单，如果选择从线上数据库读入，需要提供数据库的连接凭据，数据库名和对应信息所在的表的名称。

2.3 输出数据

一个文本文件报告，包含 2 部分

1. 针对每一份源代码，输出最相似的若干份（可自定义具体数目）源代码的 ID 和相似度量化值。
2. 对于每道题目，提供最有可能是互相抄袭的若干个源代码对。

注意，在输出的报告中，不会包含相同用户不同代码间的相似性检测信息，也不会跨越题目检查代码。

该报告只以第三章所述算法的角度计算相似度量化值并排序，因而不可避免地与真实情况产生偏差。用户在获取该系统生成的报告后，需要按实际情况手动核实报告中涉及的抄袭嫌疑，并根据需要采用其他方式检测其余代码排除漏检，切不可直接按报告判定抄袭。

第三章 核心设计

本解决方案的核心部分是数据分析子系统，该子系统负责分析代码，计算相似度，因而事先的算法分析变得非常重要。

3.1 抄袭模型

在设计算法之前，对学生在程序设计时 间中的常见抄袭做详细的归纳分析，对实际的反抄算法的设计有决定性的帮助。

一般而言，我们认为学生在完成自己的程序设计实践作业时，对其他人的程序代码只做一点点修改或是完全不做修改，就直接当作自己的学习成果。因而反抄袭的技术工具的开发重点也应该落在“如何分辨两份代码是否是互相复制的结果”。现在反抄袭系统的进展方向主要是提升已有算法精确度和在更高的思维层次上侦测抄袭行为。对于前者，目前的系统一般采用文本比对的方法，近年来学术界也采用了许多新的，对程序源代码分析有针对性的文本匹配算法，可以有效快速的在文本层面上分析代码相似性，但本解决方案不打算在这个方向上钻研，而是选择了后者。

在经济学上，分析一个人从事某项事务的收益，一般做法是利用最后的总收入减去该事务运作过程中产生的所有成本，经济学假设每一个人都是自私的，也就是每个人的经济行为都是为了最大化其受益（不仅仅是钱，还有其他因素，比如考试成绩)。在本解决方案涉及的方向上，对于抄袭的学生，在理想的情况下（就是完全没有反作弊监察手段），其作弊行为应为直接复制其他人的源代码，这时其抄袭成本基本为零，收入是“完成了该项考评”，受益几乎全部是由抄袭行为带来的。但是，在有反作弊监察手段的干预下，为了避免抄袭被发觉，学生一般会采取各种手段逃避抄袭监察，这需要花费时间和智力成本，反抄袭监察越严格，为了逃避抄袭监察所采用的手段就越复杂，所消耗的成本就会越高，直到抄袭所消耗的成本超越了抄袭的收入，学生就会放弃抄袭，改用认真独立完成作业的策略。以下是常见的针对自动化反抄袭检查对抗方案：

1) 任意大规模增加或删除程序注释
2) 更改代码内的变量名、函数名、类名等，并用宏定义替换已有的标识名称。
3) 更改函数或子程序的位置
4) 将子过程展开，手动嵌入至调用子过程的函数中
5) 添加大量的无效语句、变量
6) 以等效语句替代原语句
7) 以等价表达式替代原表达式

目前，无论采用什么算法，基于文本的源代码抄袭检测系统只能有效的处理 1，3，4，5 这四个方面。部分针对某些程序设计语言实现的基于词法分析或语法分析的抄袭检测系统可以对抗 2，目前与校 ACM 集训队配套使用的反抄袭系统虽然是基于文本分析的，但采取了通过统一重命名所有的英文子串的策略排除了这种干扰。根据上述分析，可以料到，在基于文本的自动化抄袭监察系统盛行的时代，学生只要采用 6，7 两种方案就可以完全避开抄袭监测系统的检查。因此，目前设计任何反抄袭系统，所应关注的重点是适度提升对高级抄袭手段的侦测能力，而不是在提升较低级抄袭手段的

侦测准确度上消耗设计资源。本解决方案就是要重点处理 6，7 两种对抗方案，用户可以独立使用这个解决方案（因为 1-5 本质上没有采取 6，7 的对抗方案，依然不会干扰本解决方案），也可以配合其他基于文本的反抄袭系统，加强系统的智能性。

3.2 检测方案

虽然解决 6，7 两点非常有吸引力，但是其难度也是很高的，要求反抄袭系统分析代码的深层次含义，并且可以在两个“深层次含义”中间进行匹配，量化出相似度。这个问题可以分为 2 部分：含义表达和表达匹配。前者的目标是提取特征，尽可能逼近原始含义的衡量代码，后者的目标是比较这些特征，尽可能准确的计算相似度。

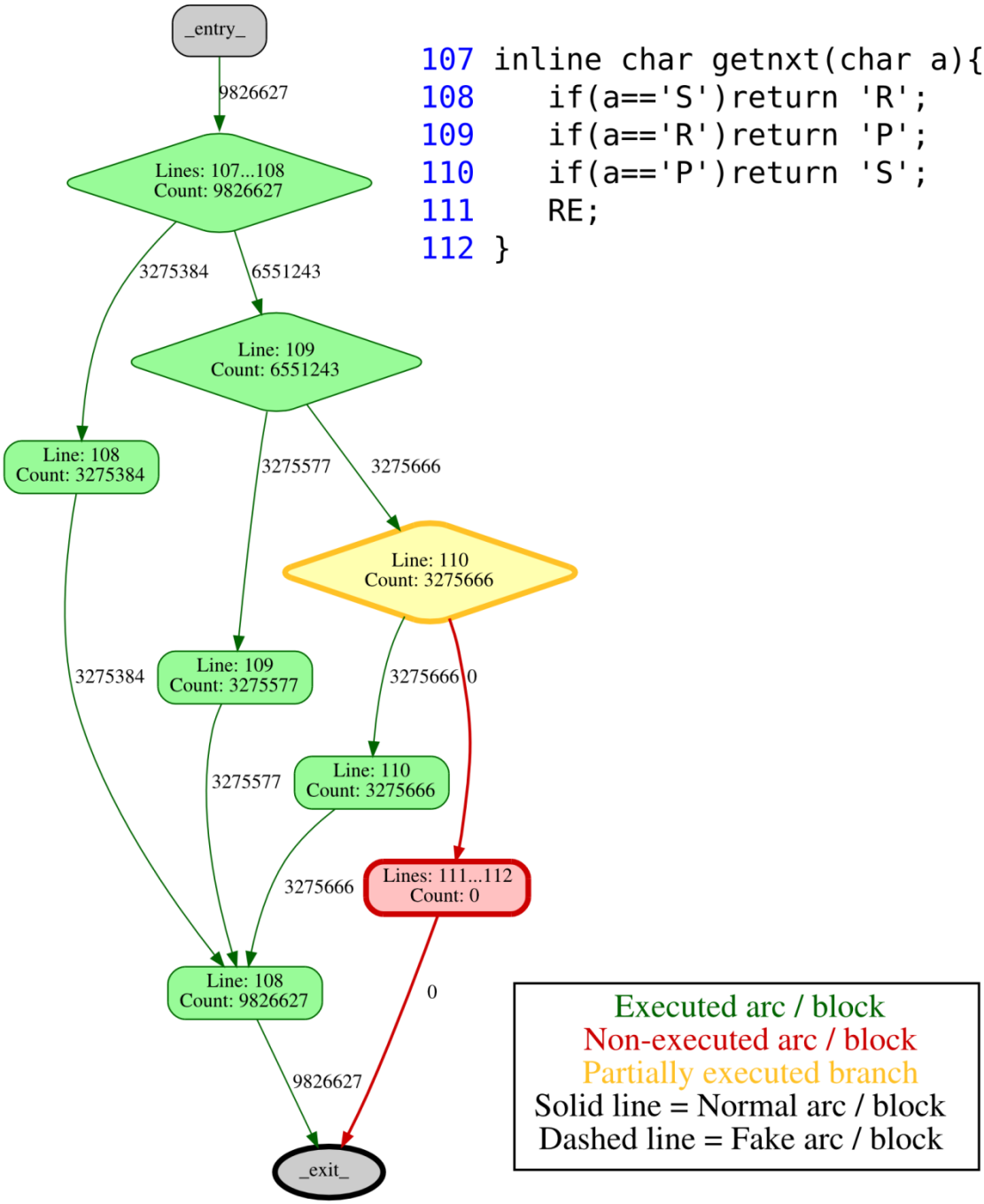


图 3-1 示例代码及其控制流图和覆盖率信息

在含义表达环节，考虑到学生可以使用完全改写每一句源代码的方式逃避反作弊检查，因此本解决方案并不打算直接处理源代码，取而代之的是，通过编译器事先编译源代码，并从中较高层次的信息，并对其加以分析。

在编译理论里，控制流图（如图 3-1）可以很好的表达代码的执行意图，控制流图是指程序代码在执行过程中的所有可能执行的代码路径的一种有向图表示，在编译器中，他是一个抽象的数据结构，用于代码优化，分析等功能。在控制流图中，每个节点代表一个基本块，一般的，在同一个块内的代码，要么一起执行，要么全不执行，边代表代码执行的可能路径，如果一个块有若干个入度，每一个入边的源为 a1，a2，...，an，代表该代码块可能在 a1，a2，...，an 块执行后跳转进入，如果一个快有若干个出度，代表该块内有条件转移语句，由该语句决定代码的下一个执行块。

理论上，针对两个待检查的程序源代码，由编译器编译并绘制控制流图，然后比对两个图的相似度，就可以在逻辑层面判断两端代码是否互相抄袭，这个方法完全不考虑源代码本身，避免了文本修改和代码替换造成的文本不一致对判断结果的干扰。但是，比对两个图的相似度是一个非常消耗计算资源的过程。即使是其简化问题，判断两个图是否同构[2]（指若有图 G1 和 G2， 它们的顶点集和边集之间都分别建立了一一对应的关系，并且 G1 的两顶点间的边对应 G2 对应顶点间的边，则称图 G1 和 G2 互为同构），根据目前计算复杂度理论的证明，该问题是 NP 完全的。因而我们要找一个近似的比对方案，节约计算量而保留重要信息。

通过对代码的观察，我们可以发现，在代码中决定控制流的一个非常重要的因素是条件分支，包括条件判断，条件循环控制在内的一些语句是实现他们的重要途径，这些语句构成了代码逻辑思想的核心部分。因此，我们只需要收集足够的信息来描述条件分支，就可以在很大程度上描述整个控制流图甚至整个程序的性质。

根据这个思路，我们可以构建相似度判断算法的大致框架：首先编译代码，得到控制流图，然后分析图中所有出度不为一的基本块（代表该基本块中包含了条件分支语句），量化这些块的入边和出边信息，最后与其他代码产生的类似信息比对，给出近似的相似度判断结果。

通过分析目前已有的判断图同构的算法，我们可以发现，这些算法会分析单一顶点的前驱和后继节点的信息，给出可能的匹配尝试，我们的基本块匹配也将参考这个做法。但是，在传统的图同构算法中，如果顶点的前驱，后继节点在算法判断的角度上过于相似，就会造成大量的回溯，这就是传统图同构算法需要耗费大量计算资源的根本原因。现在学术界研发的较新的图同构优化算法，其核心研究热点就是如何给匹配节点的过程添加足够有效的启发信息，使得匹配过程可以适当的考虑全局形势，避免失败产生回溯浪费计算资源，我们将沿着这个思路。在本解决方案中，由于我们要处理的有向图是特殊的，是由程序源代码生成的控制流图，我们可以利用这个性质来提供额外的启发信息，加速运算。

当我们给定一个数据集合，用该数据集合作为输入，执行一个计算机程序的过程中，我们可以通过调试器或是覆盖率检测器监测其代码执行顺序，这个执行顺序就是程序针对特定数据的控制流。根据上面的论证，我们只关心控制流图中的条件分支部

分的特征，为了收集足够的匹配用启发数据，我们可以考虑在程序运行时期（动态）收集控制流在条件分支部分的覆盖情况，包括命中次数，各分支命中几率等信息。

在收集信息之后，我们要量化这些分支信息，供匹配器使用。根据代码的性质，量化的具体要求是，要保留控制流在条件分支处的唯一性的指纹，又要考虑到两份互相抄袭的程序的微小不同（尤其针对上面所说的以等价表达式替代原表达式甚至是将代码段替换为本质一样的其他代码段），这个量化算法要有足够的鲁棒性，对此，我的量化方案是：

1.输入参考的测试数据，完整的运行整个程序。

2.在程序运行的过程中，挂载覆盖率检查软件。

3.抽出控制流图中的每一个条件分支，根据每个分支的执行次数，计算每个分支的概率。

4.将概率在[0.05,0.95]以外的分支去掉。

5.将每一个分支的命中概率分别对 0.1，0.3，0.7，0.9 求差，选出绝对值最小的一项

6.对每个分支按照 5 产生的值求和。

按照这个量化方案，每一个分支都会有一个大于 1 的权值。计算量化值这一步的复杂度分析如下：假设程序代码包含了 n 个分支节点，假设每一个节点都对其他节点产生了有效的条件转移（指转移的分支不会被上述第 4 步去掉），那么计算一个节点的量化值的复杂度是 O(n)，完整的计算需要 O(n^2)，当然，没有哪个实际存在的程序有着如此密集的条件转移，假设一个程序的分支节点的分支数符合负指数分布，平均分支数为 k，可知该量化算法的平均复杂度是 O(kn)[10]，一般而言，k 可以取较小的常数。

为了让用户根据实际需求，可以在准确度和运行速度之间找到合适的平衡点，本解决方案实现了 3 种不同的量化整合和匹配算法。

3.2.1 NAIVE 算法

该算法的运行速度最快，但是准确度欠佳。在上一步所述的条件分支节点量化之后，每一份待检测的程序源代码会被转换为一个正数集合，这个集合包含了每一个量化植大于 0.001 的条件分支节点的量化值。这个集合将代表原始的程序源代码进入下面的匹配算法。

我们的匹配算法采用了动态规划的思想，动态规划[5]是一种在数学和计算机科学中使用的，用于求解包含重叠子问题的最优化问题的方法。其基本思想是，将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。动态规划的思想是多种算法的基础，被广泛应用于计算机科学和工程领域。比较著名的应用实例有：求解最短路径问题，背包问题，项目管理，网络流优化等。动态规划在查找有很多重叠子问题的情况的最优解时有效。它将问题重新组合成子问题。为了避免多次解决这些子问题，它们的结果都逐渐被计算并被保存，从简单的问题直到整个问题都被解决。因此，动态规划保存递归时的结果，因而不会在解决同样的问题时花费时间。动态规划只能应用于有最优子结构的问题。最优子结构的意思是局部最优解能决定全

局最优解(对有些问题这个要求并不能完全满足，故有时需要引入一定的近似)。简单地说，问题能够分解成子问题来解决。在本解决算法中，我们采用了类似最长公共子序列的动态规划算法。

首先我们定义节点间的相似度函数，当我们进行匹配时，两个节点的相似度 diffv_navie(a,b)=10.0/(0.1+math.fabs(a-b))，然后，我们将代表两个源代码 A，B 的量化值集合{an}和{bn}由大到小排序，得到两个数列 la，lb，我们使用以下的递推式计算最终的相似度：

```
F=diffv_naive
la=sort({an}),lb=sort({bn})
m[i][j]=max(m[i-1][j-1]+F(la[i-1],lb[j-1]),m[i-1][j] if j<b else 0.0,m[i][j-1] if i<a else 0.0)
s=match_naive({an},{bn},F(x,y))=m[a][b]
```

其中，a 为源代码 A 的长度，b 为源代码 B 的长度，F(x,y)是 lambda 表达式，代表两节点间相似度计算函数，在这里是 diffv_naive，最后，相似度存储在 s 中，这个值将作为两份代码互相抄袭的嫌疑值提交到最后的报告生成环节。

下面是这个算法的复杂度分析如下：假设两份代码基本等长，有 n 个分支节点，在计算相似度的过程中，上面的算法需要填充 m[i][j](0<=i<=a,0<j<=b)各一次，在每次填充时的读取次数是常量，计算量也是常量，所以整个算法的时间复杂度是 O(n^2)。

3.2.2 ADVANCED 算法

在测试 NAIVE 算法的过程中，我发现了一个影响精确度的重要问题：很多节点的相似度差距非常的近，这可以利用二八定理解释：大多数的条件转移有着非常单一的偏置的转移方向，在典型的计算机程序里我们可以看到，很多的 1/0 条件分支的某一个转移方向的概率远远大于另一个方向，虽然在量化的过程中，我们抛弃了所有的产生极端转移的条件分支节点，但是剩下的节点足够集中到让干扰匹配算法不能良好的区分量化值接近的不同的条件转移节点带来的量化值差异和由于逃避作弊审查产生的代码修改对相同节点的量化值影响，于是，在计算节点差别的过程中，我在直接比对量化值以外又加入了对其前驱节点量化值的匹配过程。于是，相对 NAIVE 算法，ADVANCED 算法的两节点间相似度函数为：

```
diffv_advanced(Ai,Aj)=diffv_naive(Ai.val,Aj.val)+3*match_naive(Ai.prev,Aj.prev)
```

其中，Ai.val 代表节点 i 的量化值，Ai.prev 代表节点 Ai 的前驱节点的量化值集合。随后我们进行量化值集合匹配：

```
s=match_advanced(A,B)=match_naive(A,B,diffv_advanced)
```

这里又一次使用了类似最长公共子序列的的动态规划算法。S 这个值将作为两份代码互相抄袭的嫌疑值提交到最后的报告生成环节。

在实现环节，由于我们生成的控制流图的过程中没有生成前驱节点信息，因而我们要对节点进行拓扑排序[9]，再按照拓扑序标记每个节点的前驱节点。注意，拓扑序仅用于 DAG(有向无环图），在拓扑排序前需要深度遍历控制流图，删除所有的回指的转移路径。

下面是这个算法的复杂度分析：假设两个程序的条件分支数几乎相同，量级为 n，执行拓扑排序和事先去除回边的算法的时间复杂度为 O(n)，假设每个包含条件分支的块可以有意义地转移到其他所有块，那么，执行 diffv_advanced 的时间复杂度是 O(n^2)(证明同上一节的 match_naive({an},{bn},diffv(x,y)))，而最终的匹配需要调用 O(n^2)次 diffv_advanced，所以总体的时间复杂度是 O(n^4)。但是有效的程序是不存在如此密集的条件转移的，假设程序的条件分支节点的分支个数符合负指数分布，平均有 k 个分支，那最终的时间复杂度为 O((kn)^2)，一般 k 很小，可以取常数，结果为 O(n^2)。

3.2.3 INSANE 算法

虽然我们通过 ADVANCED 算法已经得到了较满意的结果，但是分析匹配过程中，我们对量化值集合使用了排序+类最长公共子序列的匹配算法，实际上这个算法是基于贪心的，并不能求出最优的匹配（只能找到次优解），为了达到理论上的最佳匹配结果，我们构建这样的的图论模型：假设有二分图 G(A,B,F(x,y))，A，B 代表个程序的条件分支节点，为二分图的两个顶点群，然后，我们在每一个 Ai 和 Bj 间连边，边权为 F(x,y)=diffv(Ai.val,Bj.val)，为 Ai 节点和 Bj 节点的量化值差异，最后我们尝试找出 A，B 的一个最大权匹配{E}，使得 E 中的边的权值和最大，并且 E 中不存在两条边共享同一个 Ai 或 Bj。解决这个问题可以使用 K-M 算法[6]，整个 INSANE 算法为

```
match_advance({An},{Bn},F(x,y))=KM(G(A,B,F(x,y))
```

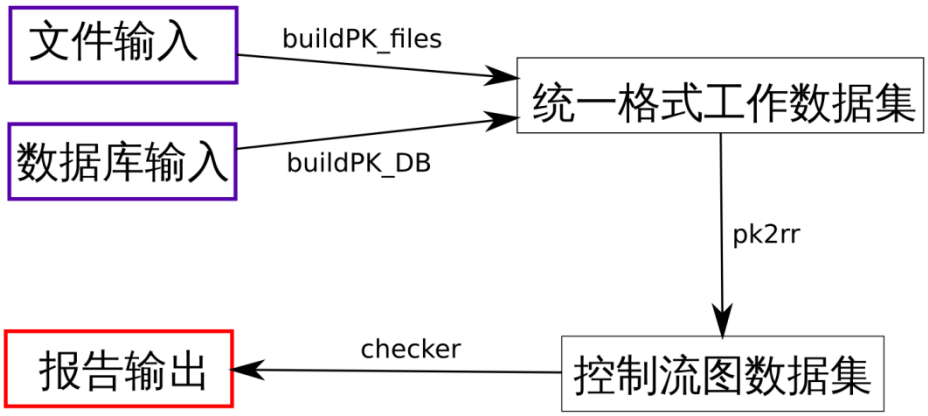
这个算法的时间复杂度就是 KM 的时间复杂度，一般认为是 O(n^4)，如果采用了 slack 优化，可以降为 O(n^3)。

在所有的程序源代码两两比对计算出相似度后，相似度矩阵 s[i][j]将被送入报告生成模块，生成报告。由于源代码的相似度本身没有含义，不同的源程序的复杂度和不同的抄袭程度均会影响这个指标，因而没有一个特别的门限可以判断两份代码之间是否有互相抄袭的痕迹，但可以确定的是，在算法实现的角度，假设代码 A 与 B 的相似度为 Sab，A 与 C 的相似度为 Sac，若 Sab>Sac，可以说在一定程度（并不是绝对的意义）上 B 与 A 相对 C 与 A 更有抄袭的可能。因此对于每个待检测的作业题目，输出若干项相似度最高的代码项，用户可以根据自己的劳动量有的放矢的人工复查一部分代码。

第四章 详细设计

4.1 系统运行流程

图 4-1 是整个解决方案运行时的流程图，本解决方案根据用户需要，可以选择 2 种不同的输入方式：文件输入和数据库输入，前者是一系列文件和文件清单，后者是已有的线上数据库中的数据表。这两种数据经过格式化工具 buildPK_files 和 buildPK_DB，可以生成一个统一格式的工作数据集，然后工作数据集经过 pk2rr 处理，生成控制流图数据集，最后，checker 负责分析控制流图，计算程序源代码间的相似度，给出报告输出。



4-1 系统运行流程图

下面我们将详细的描述整个解决方法的数据，模块结构和实现。

4.2 数据

本节将描述本解决方案的输入，输出和中间数据的格式，用途。具体的数据类型为”文件输入“，”数据库输入“，”统一格式工作数据集“，”控制流图数据集“，和”报告输出。

4.2.1 文件输入

文件输入包含若干个文件和 2 个文件清单。

清单 probs 是一个文本文件，文件有 n 行，代表 n 个题目，每行有 2 个数据，依次为每一道程序设计问题的题目 id 和数据文件名，前者为一 32 位有符号整数，后者为一不包含制表符的字符串，两者以制表符分割。

清单 codes 为一个文本文件，文件有 m 行，代表 m 份待检测的程序源代码，每行有 5 个数据，依次为每一份源代码的代码 ID（用于后续标识代码），语言 ID（0 代表 C，1 代表 C++，2 代表 Java，理论上所有 GCC 支持的语言本系统都可以支持），问题 ID（标识改份源代码，用户 ID 和程序源代码的文件名，代码 ID，语言 ID，问题

ID 和用户 ID 均为 32 位有符号整数，文件名为不包含制表符的字符串，这 5 个数据以制表符分割。

除了文件清单，用户还需要提供文件清单中涉及的所有文件，包括参考测试数据和程序源代码。

用户在执行转换工具 buildPK_files 时要提供两个文件清单并保证所有文件可用。

4.2.2 数据库输入

数据库输入要求 2 个数据表，probs 和 codes。

probs 表存储了题目数据，为 2 列，列 pid 为 32 位有符号整数(INT)，为题目 ID，列 data 为大容量二进制数据(BLOB)，为题目的对应参考测试数据，本表有 n 行，代表 n 个问题。

codes 表存储了待检测的程序源代码数据，为 5 列，分别为 cid，代表源代码的标识 ID，lid，代表语言类型，pid，代表题目 ID，uid，代表用户 ID，和 code，为程序源代码文本。前 4 列为 32 位有符号整数(INT)，最后一列为文本行(TEXT)。

用户在执行转换工具 buildPK_DB 要提供数据库管理器（这里是 MySQL）的远程连接地址，端口号，用户名，密码，表所在的数据库的名称，和两个表的名称。buildPK_DB 支持用两个视图（view）代替两个表提供数据，这方便了用户执行从内部数据库到本解决方案所需数据的转换过程。

4.2.3 统一格式工作数据集

无论是文件输入还是数据库输入，最后都会被转换为统一格式工作数据集，以方便后续处理。统一格式工作数据集是一个 sqlite3 数据库文件，包含了两种输入方式引入的所有数据，并且只有一种格式。每一个统一格式工作数据集包含了一个数据库，内含 2 张表，codes 和 datas，前者由以下指令创建：

```
create table codes(cid int primary key,lid int,pid int,uid int,code text)
```

其中，cid 为源代码的标识 ID，lid 为语言类型，pid 为题目 ID，uid 为用户 ID，code 为源代码文本。可以发现，这个表和数据库输入所用的表的模式基本一致，因而为了性能考虑，尽量使用数据库输入的输入方式。

Datas 由以下指令创建：

```
create table datas(pid int primary key,data BLOB)
```

其中，pid 为题目 ID，data 为二进制存储的参考测试数据。

统一格式数据集包含了整个反抄袭检测工具需要的一切信息，将信息封装在一个单一文件有利于简化部署环境，同时使用标准化的文件格式也有助于软件维护甚至是二次开发。

4.2.4 控制流图数据集

当统一格式数据集生成后，用户需要调用 pk2rr，这个工具提取出统一格式数据集中包含的信息，并自动化安排代码的编译，加跟踪符号，按参考测试数据运行对应的二进制可执行文件并记录控制流图和覆盖率信息，并将其存储在控制流图数据集中。

控制流图数据集是一个文本文件，使用了类此 python 的缩进嵌套的树形结构。文件的每一行由 k 个空格开头，它与最近的所有同为 k 个空格开头的行同属于最后出现的一个 k-1 个空格开头的行的子信息，所有的行依此组成一个树形结构，每行为一个节点。在空格之后是一个@开头的标识符，代表该节点描述的数据类型，然后是若干组数据，以空格分离，代表具体的数据值，这些值没有单独的名字。每个@块要在结束的位置标注“@.”(@和小数点）开头的同名标识符，代表块的结束。

在保存控制流图数据集的用途中，整个文件在顶层分为若干个@rr 块，每一个块代表一份有效的，可以编译并正确执行的源代码产生的控制流图及其覆盖率测试信息。在@rr 块下面有一个@uid 节点，所附数据为编写该源代码的用户的 ID，还有一个@pid 节点，所附数据为该代码所属的题目的 ID，最后为若干个@func 块，代表控制流图中的每个函数子图，所附数据为函数名，内含的子节点中，节点@src 代表该函数所属的源代码的文件名，若干个@block 代表基本块（控制流图的顶点），所附数据为基本块的块 ID，下属有 2 种节点，@lines 所附数据代表该块在源代码中的行位置，@arc 代表从该块发出的边（基本块间转移）的信息，所附数据为边的 ID，内含数据为@dest，所附数据为边的指向目标块的 ID，@hit 所附数据为该转移方式的命中次数，@fack 所附数据为该边是否为循环的回边。

由此可见，控制流图数据集保存了每一个程序源代码的控制流图的邻接表表示，并将覆盖率测试信息（即每个转移的遍历次数）作为边权记录在邻接表中。控制流图数据集会作为唯一的数据输入，进入执行反抄袭检查的工具中。

4.2.5 报告输出

在 checker 执行完成后，会生成一个检测报告，报告为文本文件，分为两部分，第一部分由若干份代码信息组成，每份信息与每一个待检查的源代码一一对应，并通过代码 ID 标识，所属的信息包括与该代码最相似的若干份代码的代码 Id 和相似度量化值。报告的第二部分由若干个代码对组成，代表每道问题所涵盖的所有源代码中，最有可能互相抄袭的若干个代码对及其相似度量化值。

4.3 工具集

本节将描述参与系统运行的工具集合。由图 4-1，在这个工具集中有 4 个工具，分别为 buildPK_files，buildPK_DB，pk2rr，checker。

4.3.1 buildPK_files

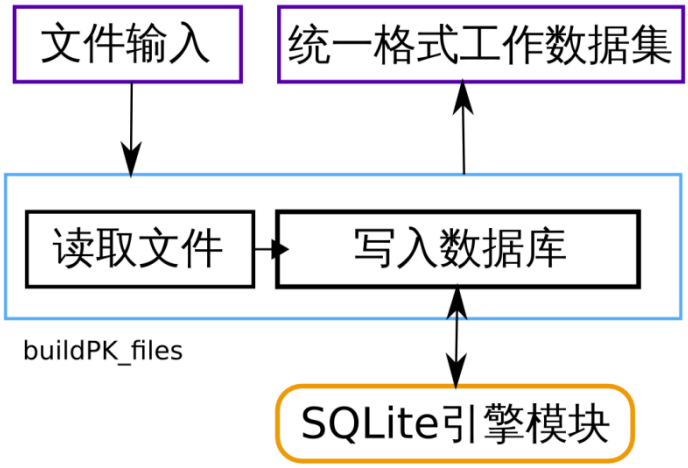


图 4-2 buildPK_files 结构

该工具负责将文件输入转换为统一格式工作数据集。文件输入包含 2 个清单和清单所列出的所有文件，统一格式工作数据集为一个 sqlite 数据库文件。该工具的内部工作流程如下：

1.检查清单合法性，判断文件是否齐备。

2.尝试打开输出所用的 sqlite 数据库，如果该数据库文件已经存在，则清空数据库。

3.在数据库中建立统一格式工作数据集中包含的两个表格。

4.读取题目清单，并读入参考测试数据，将其以指定格式写入数据库。

5.读取代码清单及源代码文件，将其以指定格式写入数据库。

6.提交事务并关闭数据库。

4.3.2 buildPK_DB

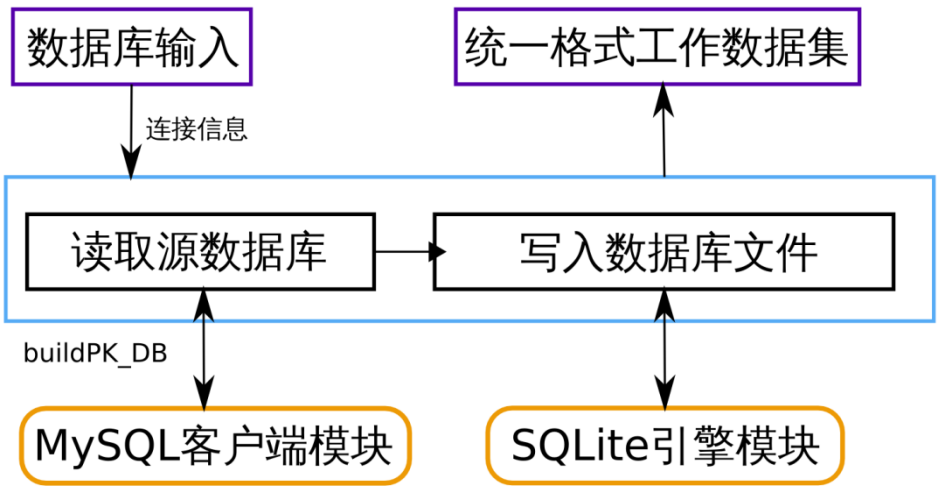


图 4-3 buildPK_DB 结构

该工具负责将数据库输入转换为统一格式工作数据集，数据库输入包含数据库的地址，连接凭据，和数据库中两个表的位置，输出统一格式工作数据集。该工具的内部工作流程如下：

1.尝试连接目标数据库，检查表的合法性。

2.尝试打开输出所用的 sqlite 数据库，如果该数据库文件已经存在，则清空数据库。

3.在数据库中建立统一格式工作数据集中包含的两个表格。

4.读取数据库中的两个表，并将对应信息写入统一格式工作数据集中。

5.切断远程数据库连接，提交事务并关闭本地数据库。

4.3.3 Pk2rr

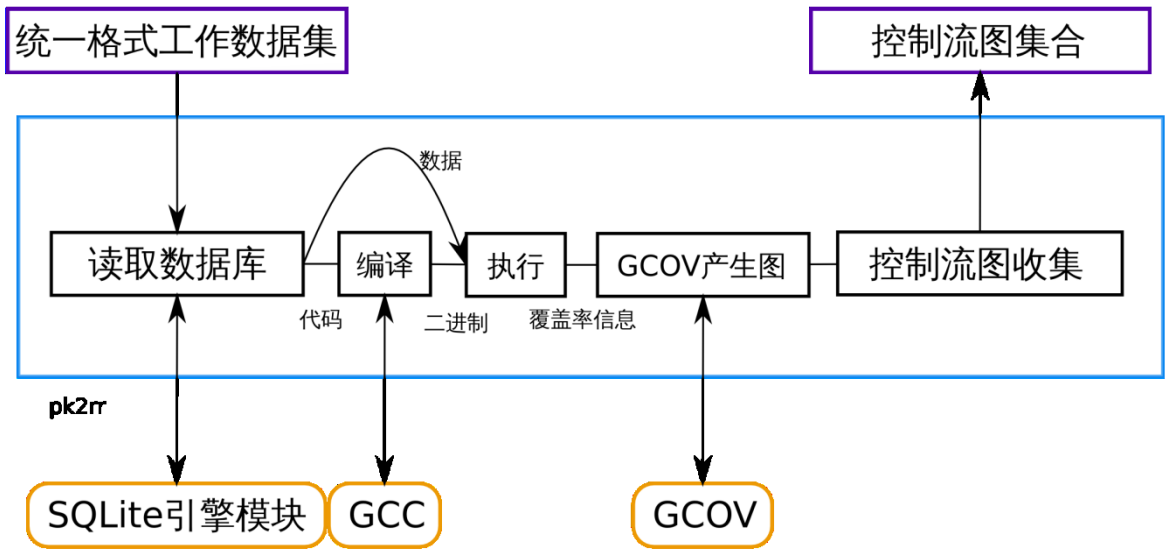


图 4-4 pk2rr 结构

该工具负责将统一格式工作数据集转换为控制流图集。在具体的实现中，该工具调用了修改过的 GCOV。

GCOV 原为 GNU 覆盖率测试器，用户在使用 GCOV 之前，需要以特殊方式编译待检测的可执行文件，即打开选项让 GCC 向二进制文件输出中添加基本块和控制流图信息，并在基本块间转移的位置添加计数器代码。随后，用户运行可执行文件，这是会在当前目录下生成两个文件，分别存储了控制流图信息及图中各块间转移弧的遍历计数，以及控制流图中的每一个基本块到程序源代码的映射关系。GCOV 通过读取这两个文件，输出一份报告，其中的主要内容是源代码本身，但在每一个有效的可执行行都标记了他的执行次数，如果打开了高级选项，还可以输出基本块信息和条件分支的转移概率。但是这些信息对于我们的应用依然显得缺乏，于是我自己修改了 GCOV 的报告输出部分，去掉了以文本方式生成源代码的报告部分，以输出方便程序处理的树形结构文件，并且在输出文件中完整的导出了整个控制流图和每一个转移弧的命中

次数和转移户的属性，而不是想 GCOV 原先那样，为了生成易读的文本报告而对控制流图进行高度处理和大幅度修改。

Pk2rr 的工作流程为：

1.检查所需的临时文件路径是否正在使用，然后打开统一格式工作数据集，检查其数据结构。

2.抽出一份源代码，并将其对应的题目数据抽出，将这两者保存在磁盘临时文件中。

3.调用 GCC 以 GCOV 指定方式编译程序源代码，随后运行源代码，最后利用本解决方案特制的 GCOV 提取出控制流图和覆盖率报告。

4.将报告写入输出的控制流图集合，清除临时文件。

5.若还有源代码未处理，转到 2，直到处理完所有的源代码。

6.关闭统一格式工作数据集，清除所有临时文件。

4.3.4 checker

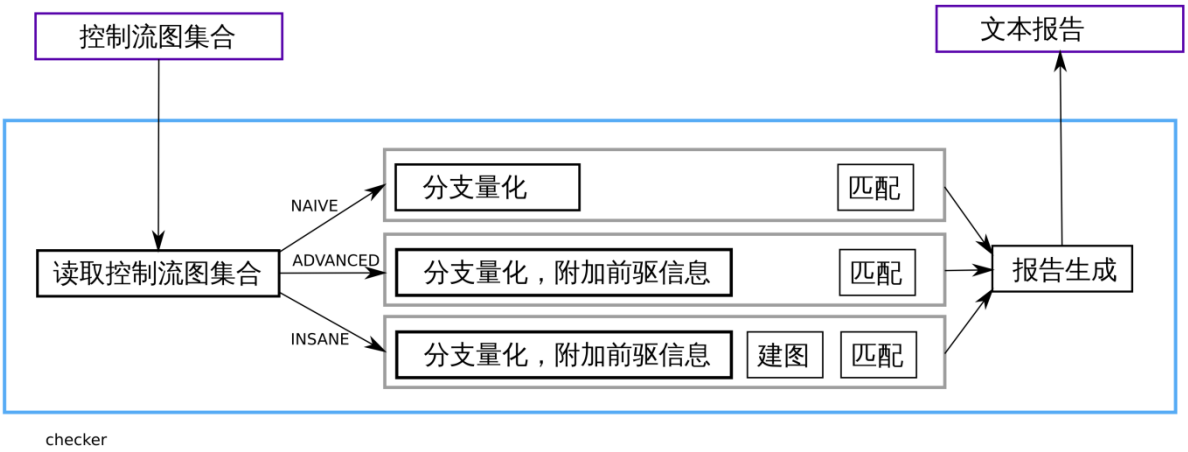


图 4-5 checker 结构

本工具是本解决方案的核心工具，负责两两比对同一题目，不同用户提交的代码的相似度，并给出检测报告。

checker 包含了 2 个外部模块，分别为：

1.graph_ops，该模块负责提供在 checker 需要在控制流图上运行的一些基本操作，包含了检查控制流图，合并一进一出块（这些块对与基于条件分支检测的功能无用），删除无用边，删除无覆盖边，标记循环回边，出度/入度计算和计算分支概率的代码。

2.basictype_rr，该模块负责将控制流图文件反持久化为抽象的，包含覆盖率信息的，以邻接表表示的控制流图。模块包含控制流图中各元素的抽象数据结构和反持久化代码。实际使用时，只需调用 basictype_rr 中的 rrreader 即可一行读入一份程序源代码的所有上述信息。

另外，为了给用户提供在精确度和计算资源消耗的权衡选择，checker 可以选择调用 3 个不同的算法：NAIVE，ADVANCED，INSANE（详细说明请参见 3.1.2，检测方

案）。这三个算法以三个独立但接口完全一致的模块的方式实现，用户可以修改checker代码中的import选择具体使用哪一个算法。

Checker的详细工作流程为：

1.读入控制流数据集，按题目ID，用户ID分类。

2.预处理每一份控制流数据，具体的过程为：检查控制流图，找出控制流图中每个子图的入口，然后清除所有无效的边，主要是没有执行过的路径，然后计算整个控制流图的每个基本块的度数，随后将一进一出的基本块合并到最近的包含有效条件分支的块，然后计算每个分支的覆盖率，分支概率。

3.调用用户选择的算法，在所有题目ID相同，用户ID不同的代码对之间计算匹配度。

4.整理匹配度数据，按照每题的相似度和每份代码的相似度提取出抄袭嫌疑最大的若干项。目前，每分源代码分拣出前10个相似度最高的源代码，每道题目分拣出前100个相似度最大的代码对。

5.输出文本报告。

第五章 测试

5.1 测试环境

计算资源：

服务器一台，型号：Dell R910，CPU 为 2 个 Intel Xeon 7520，内存为含 ECC 校验的 DDR3-1066，共安装 32G，硬盘为 6 个 7200rpm 的 500G SAS 硬盘，以 PERC H700 阵列卡组成一个 RAID5 阵列。

软件资源：

Ubuntu Server 11.04 64bit，使用发行版光盘镜像，按照默认选项安装，安装 CPython 3.1.2，不安装任何更新。

5.2 数据源

测试所用的数据来源为北京邮电大学第一代 OJ（o.boj.me)的 1791 题至 2012 年 3 月 1 日前的所有提交结果，从中选取所有可以完整运行参考测试数据并按照时间空间限制给出正确输出结果的代码，共 145 份。

5.3 测试结果

示例报告输出（ADVANCED 算法，部分省略）：

```
Working on problem 1791.
      Calc self similarity.
      Calc mutual similarity.
0.0% 0.0% 0.0% 0.1% 0.1% … 59.8% 60.3% 60.8% 61.4% 61.9% 62.4% 63.0% 63.5%
64.1% 64.6% … 99.4% 100.0%
      Top 10 similar with Code 359223 (MTV:9900.0000)
            229300: 8314.8244(0.16012)
            277476: 8067.9050(0.18506)
            228689: 8013.3166(0.19057)
            228565: 7982.0479(0.19373)
            260528: 7884.2102(0.20362)
            228435: 7881.7825(0.20386)
            227875: 7871.4306(0.20491)
            228638: 7803.3162(0.21179)
            353123: 7768.0880(0.21534)
            228143: 7751.3428(0.21704)
      Top 10 similar with Code 362423 (MTV:11000.0000)
            …

      In all code,Top 100 similar pairs are:
            227351 with 231780, diff:0.00000
            227651 with 230714, diff:0.00000
            …
            227672 with 240848, diff:0.02908
```

```
233313 with 311890, diff:0.06161
228977 with 233592, diff:0.08579
231166 with 233592, diff:0.08579
228266 with 277476, diff:0.08791
231125 with 277476, diff:0.08791
233313 with 286286, diff:0.09262
…
```

其中，典型的抄袭代码，227672 与 240848，部分内容如下：

```
   //240848
#define P printf
#define E continue
int F(char c) {
    if (c >= 'a' && c <= 'z' || (c >= 'A' && c <= 'Z'))
        return 1;
    return 0;
}
int G(char c) {
    if (F(c) || (c >= '0' && c <= '1') || c == '-' || c == '_')
        return 1;
    return 0;
}
......

scanf("%s", d);
l = strlen(d);
if (l < 4) {
    P("NO\n");
    E;
}
if (F(d[l - 2]) && F(d[l - 1])) {
    if (H(d[l - 3]))d[l - 3] = '\0';
    else if (H(d[l - 4]) && F(d[l - 3]))d[l - 4] = '\0';
    else {
        P("NO\n");
        E;
    }
} else {
    P("NO\n");
    E;
}
```

```
   //227672
int isLetter(char c) {
    if (c >= 'a' && c <= 'z')return 1;
    if (c >= 'A' && c <= 'Z')return 1;
    return 0;
}

int isSymbol(char c) {
    if (isLetter(c))return 1;
    if (c >= '0' && c <= '1')return 1;
    if (c == '-')return 1;
    if (c == '_')return 1;
    return 0;
}
......

scanf("%d", &N);
while (N--) {
    scanf("%s", dat);
    l = strlen(dat);
    if (l < 4) {
        printf("NO\n");
        continue;
    }
    if (isLetter(dat[l - 2]) && isLetter(dat[l - 1])) {
        if (isDot(dat[l - 3]))dat[l - 3] = '\0';
        else if (isDot(dat[l - 4]) && isLetter(dat[l - 3]))dat[l -
4] = '\0';
        else {
            printf("NO\n");
            continue;
        }
    } else {
        printf("NO\n");
        continue;
    }
```

时间消耗：

当采用 NAIVE 算法时，完整的运行时间是 6.8 秒。

当采用 ADVANCED 算法时，完整的运行时间是 32.5 秒。

当采用 INSANE 算法时，完整的运行时间是 431.7 秒。

效果：

根据观察，因为控制流较简单，上述 3 种算法在测试代码集合中选取的前 20 个相似度最高的代码对上没有差异，仅是排序略微不同。从准确率角度来看，前 20 个结果

包含了 5 个完全一致抄袭，2 例通过修改变量名逃避抄袭检查，2 例通过重新排列函数位置逃避抄袭检查，1 例通过手动内联函数逃避抄袭检查，其他的各例均为代码逻辑非常相似的疑似抄袭样例，虽然两两间的代码风格几乎完全不同。

在一个基于文本的抄袭检测系统的辅助下，经过人工检查其他代码，没有发现明显的作弊行为。

5.4 总结

该解决方案基本可以对应所有的简单的反-反抄袭手段，包括文本层级的替换，重排，函数手动内联，在这部分的效果与传统的基于文本的抄袭检测系统基本一致，而在更高层次的反-反抄袭手段对抗中，本解决方案也有效的找出了嫌疑最大的若干项，总之，这个解决方案可以作为较严格的程序设计实践环节的抄袭检测系统。

参考资料

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman .Compilers: Principles, Techniques, and Tools.2nd Edition.2006
[2] Douglas B. West .Introduction to Graph Theory. 2nd Edition. 2000
[3] Free Software Foundation. GCC. the GNU Compiler Collection.http://gcc.gnu.org/
[4] D. Richard Hipp.SQLite. http://www.sqlite.org/
[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.Introduction to Algorithm.1990
[6] Harold W. Kuhn. "The Hungarian Method for the assignment problem". Naval Research Logistics Quarterly, 2:83–97. 1955.
[7] Free Software Foundation. GCOV, http://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[8] Python Software Foundation, Python, http://www.python.org/
[9] Kahn, A. B. "Topological sorting of large networks". Communications of the ACM 5 (11): 558–562, DOI:10.1145/368996.369025
[10] Olav Kallenberg; Foundations of Modern Probability, 2nd ed. Springer Series in Statistics. 2002.

致 谢

首先要感谢我的毕业设计指导老师房鸣老师，在毕业设计的过程中，房老师都始终给予我细心的指导和不懈的支持。房老师在整个系统的需求挖掘和结构上给予我大量指导，使我可以快速掌握系统设计的要领，顺利的完成设计和开发工作，在论文撰写期间，房老师给予了我很大的帮助和指导、孜孜不倦地审阅和改进意见，使我受益匪浅，顺利地完成论文。

感谢计算机学院实验中心提供给我这样一个非常好的动手实践的机会，包括在非常高端的服务器上提供足够的使用机时，感谢中心所有关心我学习、工作并给予指导的老师，同时非常感谢中心给我分配了良好的工作环境。

感谢北京邮电大学 ACM 在线评测系统的开发者，洪定坤、周闻青和赵鑫和其他参与 bug 维护或功能开发的同学，这些同学为我的开发过程提供了一个非常好的实验平台和数据来源，感谢 o.boj.me 的用户们，这些同学为系统的测试提供了大量且高质量的资源。

最后感谢北京邮电大学 ACM 集训队的全体同学，在系统的设计和测试阶段，他们都给出了十分宝贵的意见和建议，使得系统在各个方面不断完善。

外文译文

Connecting databases to Python with SQLObject

Summary: An object-relational mapping tool helps improve your productivity by providing classes and objects to manipulate database tables. The best object-relational mapping tool for Python is SQLObject -- an open-source project that does just about everything you might need to program a database. This article introduces SQLObject and its capabilities. After reading this article, you'll be able to connect Python to databases without writing any SQL code.

You'll often see an object-relational mapping when the object-oriented programming paradigm meets the relational paradigm of most databases. An object-relational mapping is a bridge between the two worlds. It lets you define classes that correspond to the tables of a database. You can then use methods on those classes and their instances to interact with the database, instead of writing SQL. Using an object-relational mapping doesn't mean that you don't need to know how relational databases work, but it does save you from having to write SQL -- a common source of programming errors.

You can find more than a dozen open source Python packages for working with a SQL database, not counting the special-purpose modules that connect Python to specific databases. SQLObject is the best such module. It's a full object-relational mapping package that's simple to use. SQLObject can do just about everything you might need to program a database.

This article shows you how SQLObject interacts with a database, how to use SQLObject to write database access and data-validation code, and how to use it with legacy or pre-existing databases. I assume that you have some knowledge of Python and relational databases.

Installing and setting up SQLObject

SQLObject has a setup.py file and installs in the same way as any other Python package. If you're using Python V2.2, you also need to install the mxDateTime Python package (SQLObject uses Python V2.3's built-in datetime module, if it's available).

To actually use SQLObject, you need to set up a database package and the Python interface to that type of database. SQLObject connects to several types of databases, including the three big open source products: MySQL, PostgreSQL, and the serverless SQLite.

Finally, you need to create a database for your application. For SQLite, this means creating a file in which to store the database. For other databases, it means connecting to the database server, issuing a CREATE DATABASE command, and granting some database user access to the new database, so that SQLObject can use that user account to connect.

Listing 1 shows how to create a new database in MySQL.

Listing 1. Code to create a new database in MySQL

```
mysql> use mysql;
Database changed
```

```
mysql> create database sqlobject_demo;
Query OK, 1 row affected (0.00 sec)
mysql> grant all privileges on sqlobject_demo to
'dbuser'@'localhost'
identified by 'dbpassword';
Query OK, 0 rows affected (0.00 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

Connecting to the database

The first Python code you need to write is the database connection code. This is the only place you need to write different code based on which database you're using.

For instance, if you want your application to use a SQLite database, you need to pass the path to the database file into the SQLite connection builder located in the sqlobject.sqlite package. If the database file doesn't exist, SQLObject will tell SQLite to create it with code such as:

```
import sqlobject
from sqlobject.sqlite import builder
conn = builder()('sqlobject_demo.db')
```

If you're using MySQL, or another database with a server, you pass in database connection information to the connection builder. Listing 2 provides an example for the MySQL database created in the previous section.

Listing 2. Code to pass in MySQL database connection information

```
import sqlobject
from sqlobject.mysql import builder
conn = builder()(user='dbuser', passwd='dbpassword',
                 host='localhost', db='sqlobject_demo')
```

Whatever database you're connected to, the connection code should go in a file called something like Connection.py, stored in some commonly accessible location. That way, you can import all the classes you define and use the conn object you've built. The conn variable will take care of all database-related details.

Note, however, that some features of SQLObject are not available in SQLite or MySQL. You can't totally separate your choice of database from the code you write after you've connected.

Defining the schema

SQLObject makes it easy to act on a database table. As a first example, consider a database schema consisting of a single table for a phone book application as shown in Table 1.

Table 1. Description of the phone_number table

Field	Type	Notes
id	Int	Primary key
number	String	"(###) ###-####" string format; should be unique
owner	String	Whose number is this?
last_call	Date	When did the user last call this number?

The SQL for this table would look something like this, depending on your flavor of SQL:

```
CREATE TABLE phone_number (
  id INT PRIMARY KEY AUTO_INCREMENT,
  number VARCHAR(14) UNIQUE,
  owner VARCHAR(255),
  last_call DATETIME,
  notes TEXT
)
```

With SQLObject, you don't need to write this SQL code. You define the database table by defining a Python class. This code goes into a file called PhoneNumber.py, as shown in Listing 3.

Listing 3. PhoneNumber.py

```
import sqlobject
from Connection import conn

class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    number = sqlobject.StringCol(length=14, unique=True)
    owner = sqlobject.StringCol(length=255)
    lastCall = sqlobject.DateTimeCol(default=None)
PhoneNumber.createTable(ifNotExists=True)
```

Here's where you use the conn variable defined earlier. Each of your table classes needs to store a reference to a database connection in its _connection member. This information is used behind the scenes for all the database access to this class' table. From this point on, you don't have to worry about SQL or any particular database because your code can be expressed in terms of the abstract relational schema.

A class that defines a table also has a set of members that define the table's fields. SQLObject provides StringCol, BoolCol, and so on -- one class for each of the database field types.

The first time the createTable() method runs, SQLObject will create a table called phone_number. Afterward, it will just use that table because you call that method with ifNotExists set to True.

Finally, note that there's no need to create a field object in PhoneNumber for the id field. Because SQLObject always needs this field object, it always creates one.

Handling that old CRUD

The famous acronym *CRUD* represents the four things you might do to a database row: Create, Read, Update, and Delete. After you've defined a class that corresponds to a database table, SQLObject represents operations on the rows of that table as operations on the class and its instances.

Create

To create a database row, you create an instance of the corresponding class with the code:

```
>>> from PhoneNumber import PhoneNumber
>>> myPhone = PhoneNumber(number='(415) 555-1212',
owner='Leonard Richardson')
```

Now the phone_number table has a row with my name in it. The PhoneNumber constructor takes values for the table columns as keyword arguments. It creates a row of phone_number using the data you provide. If, for some reason, that data can't go into the database, the constructor throws an exception. Here's what happens when the phone number isn't valid:

Listing 4. Invalid phone number

```
>>> badPhone = PhoneNumber()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
...
TypeError: PhoneNumber() did not get expected keyword argument
number
```

And here's what you'd see if the phone number was already in the database:

Listing 5. Phone number already in database

```
>>> duplicatePhone = PhoneNumber(number="(415) 555-1212")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
...
TypeError: PhoneNumber() did not get expected keyword argument owner
```

Read (and query)

An instance of an SQLObject class has all its fields available as members. This is in contrast to some other database-mapping tools that make a database row act like a dictionary. Thus, a PhoneNumber object has a member for each of the fields in its database row:

Listing 6. Fields available as members

```
>>> myPhone.id
1
>>> myPhone.owner
'Leonard Richardson'
>>> myPhone.number
'(415) 555-1212'
>>> myPhone.lastCall == None
True
```

But how do you retrieve PhoneNumber objects that are already in the database? You need to get them by running a query against the database. This brings up SQLObject's query construction kit, one of the package's most interesting features. It lets you represent SQL queries as chains of Python objects. It's similar to what you can do with Criteria objects in Torque, if you're familiar with that object-relational package for the Java™ programming language.

Each SQLObject class you define has a select() method. This method takes an object that defines a query and returns a list of items that match that query. For instance, this method call returns a list containing the first phone number in the database:

Listing 7. The select() method

```
>>> PhoneNumber.select(PhoneNumber.q.id==1)
<sqlobject.main.SelectResults object at 0xb7b76cac>
>>> PhoneNumber.select(PhoneNumber.q.id==1)[0]
<PhoneNumber 1 number='(415) 555-1212' lastCall=None
owner='Leonard Richardson'>
```

PhoneNumber.q.id indicates that you want to run a query on the id field of the phone_number table. SQLObject overloads the comparison operators (==, !=, <, >=, and so on) to make queries out of what would otherwise be Boolean expressions. The expression PhoneNumber.q.id==1 becomes a query that matches every row whose id field has a value of 1. Here are some more examples:

Listing 8. More examples

```
>>> PhoneNumber.select(PhoneNumber.q.id < 100)[0]
<PhoneNumber 1 number='(415) 555-1212' lastCall=None
owner='Leonard Richardson'>
```

```
>>> PhoneNumber.select(PhoneNumber.q.owner=='Leonard
Richardson').count()
1
>>>
PhoneNumber.select(PhoneNumber.q.number.startswith('(415)')).count()
1
```

You can use SQLObject's AND and OR functions to combine query clauses:

Listing 9. AND and OR functions to combine query clauses

```
>>> from sqlobject import AND, OR
>>> PhoneNumber.select(AND(PhoneNumber.q.number.startswith('(415)'),
>>>                        PhoneNumber.q.lastCall==None)).count()
1
```

The following query gets everyone you haven't called in a year and everyone you've never called at all:

Listing 10. Everyone you haven't called in a year and everyone you've never called

```
>>> import datetime
>>> oneYearAgo = datetime.datetime.now() -
datetime.timedelta(days=365)
>>> PhoneNumber.select(OR(PhoneNumber.q.lastCall==None,
...                       PhoneNumber.q.lastCall <
oneYearAgo)).count()
1
```

Update

If you change a member of a PhoneNumber object, the change is automatically mirrored to the database:

Listing 11. Change automatically mirrored to database

```
>>> print myPhone.owner
Leonard Richardson
>>> print myPhone.lastCall
None
>>> myPhone.owner = "Someone else"
>>> myPhone.lastCall = datetime.datetime.now()
>>> #Fetch the object fresh from the database.
>>> newPhone = PhoneNumber.select(PhoneNumber.q.id==1)[0]
>>> print newPhone.owner
```

```
Someone else
>>> print newPhone.lastCall
2005-05-22 21:20:24.630120
```

There's just one caveat: SQLObject won't let you change the primary key of an object. It's usually best to let SQLObject manage the id field of your table, even if you happen to have another field you'd make the primary key if you weren't using SQLObject.

Delete

You can delete a particular row object by passing its ID into its class' delete() method:

Listing 12. Delete a row object

```
>>> query = PhoneNumber.q.id==1
>>> print "Before:", PhoneNumber.select(query).count()
Before: 1
>>> PhoneNumber.delete(myPhone.id)
>>> print "After:", PhoneNumber.select(query).count()
After: 0
```

Validating and converting data

So far, you've been passing phone numbers in 14-character U.S. format. The database schema is designed to accept numbers in that format, which implies the underlying application -- whatever it might be -- expects them in that format. As it is, though, the code doesn't do anything to make sure that clumsy users or programmer bugs don't cause incorrectly formatted data to be put into the number field.

SQLObject solves this problem by letting you define hook methods for validating and massaging incoming data. You can define one method for each field in a table. A field's hook method is named _set_[field name](), and it gets called whenever a value is about to be set for that field, whether as part of a create operation or an update operation. The hook method should (optionally) massage an incoming value into an acceptable format ,and then set it. Otherwise, it should throw an exception. To actually set a value, the method needs to call the SQLObject method _SO_set_(field name).

Listing 4 shows a _set_number() method for PhoneNumber. If a phone number is totally without formatting, as in 4155551212, the method formats the number as (415) 555-1212. Otherwise, if the number is not in the correct format, the method throws a ValueError. A correctly formatted phone number -- or one that got massaged into a correct format -- is passed right on to SQLObject's _SO_set_number() method.

Listing 13. The _set_number () method for PhoneNumber

```
   import re
   def _set_number(self, value):
```

```
      if not re.match('\([0-9]{3}\) [0-9]{3}-[0-9]{4}', value):
         #It's not in the format we expect.
         if re.match('[0-9]{10}', value):
            #It's totally unformatted; add the formatting.
            value = "(%s) %s-%s" % (value[:3], value[3:6],
value[6:])
         else:
            raise ValueError, 'Not a phone number: %s' % value
      self._SO_set_number(value)
```

Defining relationships among tables

So far, everything you've seen operates on a single table: phone_number. But real database applications usually have multiple interrelated tables. SQLObject lets you define relationships among tables as foreign keys. To demonstrate, let's apply a little database normalization to the previous example, splitting the owner field of PhoneNumber into a separate person table. The code shown in Listing 14 goes into a file called PhoneNumberII.py.

Listing 14. Code for PhoneNumberll.py

```
import sqlobject
from Connection import conn


class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    number = sqlobject.StringCol(length=14, unique=True)
    owner = sqlobject.ForeignKey('Person')
    lastCall = sqlobject.DateTimeCol(default=None)


class Person(sqlobject.SQLObject):
    _idName='fooID'
    _connection = conn
    name = sqlobject.StringCol(length=255)
    #The SQLObject-defined name for the "owner" field of PhoneNumber
    #is "owner_id" since it's a reference to another table's primary
    #key.
    numbers = sqlobject.MultipleJoin('PhoneNumber',
joinColumn='owner_id')


Person.createTable(ifNotExists=True)
PhoneNumber.createTable(ifNotExists=True)
```

This PhoneNumber class has the same members as the old one, but its owner member is a reference to the primary key of the person table, instead of a reference to a string column in the phone_number table. This makes it possible to represent a single individual with two phone numbers:

Listing 15. Represent a single individual with two phone numbers

```
>>> from PhoneNumberII import PhoneNumber, Person
>>> me = Person(name='Leonard Richardson')
>>> work = PhoneNumber(number="(650) 555-1212", owner=me)
>>> cell = PhoneNumber(number="(415) 555-1212", owner=me)
```

The numbers member of Person, an SQLObject MultipleJoin, makes it easy to do a query based on a join of person to phone_number:

Listing 16. Query based on join of person to phone_number

```
>>> for phone in me.phoneNumbers:
...     print phone.number
...
(650) 555-1212
(415) 555-1212
```

Similarly, SQLObject lets you do many-to-many joins, using the MultipleJoin class.

Using SQLObject with pre-existing tables

One common use of SQLObject is to give a Python interface to a database created by another application. SQLObject has several features that help do this.

Database introspection

If you're working with tables that already exist in a database, you don't need to define the columns in Python. SQLObject can extract all the information it needs through introspection on the database. For example, the code in Listing 17 could go into PhoneNumberIII.py.

Listing 17. Code for PhoneNumberlll.py

```
import sqlobject
from Connection import conn
class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    _fromDatabase = True
```

This class will take on the properties of the existing phone_number database table. You can interact with it just as if you had manually defined the class with all its columns, as in the previous examples. With SQLObject, you need to write the table definition only once -- whether you do it in SQL or in Python is up to you.

However, this feature brings your choice of database back into the picture. For instance, the feature doesn't work at all with SQLite. It works at a basic level with MySQL, but it won't pick up foreign key relationships. If you're using MySQL and want to define foreign keys for a table, you'll need to define those fields in code after loading the schema from the database.

Naming conventions

The code in the previous section assumes that the pre-existing table conforms to SQLObject's naming conventions (for instance, a table's primary key field is called id, and words in column names are separated by underscores). The naming conventions for a table are defined in a Style class.

SQLObject provides some Style classes corresponding to common database naming conventions. For instance, if your column names look likeThis, instead of like_this, you can use the MixedCaseStyle:

Listing 19. Using the MixedCaseStyle

```
import sqlobject
from sqlobject.styles import MixedCaseStyle
from Connection import conn
class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    _fromDatabase = True
    _style = MixedCaseStyle
```

If none of the prepackaged Style classes fit your needs, you can subclass the base Style class and define your own naming conventions. In the worst case, where a table's field names have been assigned with no rhyme or reason at all, you can name each field individually.

A note on SQLObject limitations

SQLObject wants you to think in object-oriented terms instead of relational terms. This is good for your comprehension and your programming productivity, but it's not so good for performance. After all, the database is still relational. How do you mark every phone number in the database as having been called? With SQL, you would use a single UPDATE command. With SQLObject, you need to iterate over the entire result set and modify the last_call member of each object, which is much less efficient.

SQLObject sacrifices processor time for developer time. This is usually a good deal, but even in simple applications, you may need to drop down a level to the Python database interface and write raw SQL for some critical-path operations.

Conclusion

This article has covered SQLObject in breadth, but not much depth. It's a versatile tool with many small, convenient features. (I haven't even mentioned result slices, just to pick one.) Its limitations are easy to understand, and you can work around them by writing the SQL you need. SQLObject is to relational database programming what Python is to application programming: a convenient way to get the work done in a lot less time.

用 SQLObject 连接 Mysql 和 Python

当面向对象编程范例满足大多数数据库的关系范例时，通常会看到对象关系映射。对象关系映射是这两个世界的桥梁。它允许您定义与数据库表对应的类。然 后您可以使用这些类及其实例上的方法来与数据库交互，而不用编写 SQL。使用对象关系映射并不意味着不需要知道关系数据库如何工作，而是不必要编写 SQL，从而避免编程错误。

您可以找到一打以上的操作 SQL 数据库的开放源码 Python 包，这还没包括用于连接 Python 与特定数据库的特殊用途模块。SQLObject 是其中最好的模块。它是简单易用的完全对象关系映射包。SQLObject 几乎可以完成编程数据库所需的所有操作。

本文展示了 SQLObject 如何与数据库交互，如何使用 SQLObject 编写数据库访问和数据验证代码，以及如何将它用于遗留或现有数据库。这里假设您已经具备 Python 和关系数据库的知识。

安装和设置 SQLObject

SQLObject 具有一个 setup.py 文件，安装方式与其他任何 Python 包一样。如果您使用的是 Python V2.2，则还需要安装 mxDateTime Python 包（SQLObject 使用 Python V2.3 的内置 datetime 模块，如果该模块可用的话）。

要实际使用 SQLObject，需要设置数据库包以及这种数据库的 Python 接口。SQLObject 连接多种数据库，其中包括三个大的开放源码产品：MySQL、PostgreSQL 和无服务器 SQLite。

最后，需要为应用程序创建数据库。对于 SQLite，这意味着创建一个存储该数据库的文件。对于其他数据库，这意味着连接数据库服务器，执行 CREATE DATABASE 命令，并授权数据库用户对新数据库的一些访问，以便 SQLObject 可以使用该用户帐户来连接。

清单 1 展示了如何用 MySQL 创建新数据库。

清单 1. 用 MySQL 创建新数据库的代码

```
mysql> use mysql;
Database changed
mysql> create database sqlobject_demo;
Query OK, 1 row affected (0.00 sec)
mysql> grant all privileges on sqlobject_demo to
'dbuser'@'localhost'
identified by 'dbpassword';
Query OK, 0 rows affected (0.00 sec)
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

连接数据库

需要编写的第一个 Python 代码是数据库连接代码。基于所使用的数据库，这是惟一需要编写不同代码的地方。

例如，如果想让应用程序使用 SQLite 数据库，则需要将数据库文件的路径写入位于 sqlobject.sqlite 包的 SQLite 连接构建器中。如果数据库文件不存在，QLObject 将告诉 SQLite 创建一个，代码如下：

```
import sqlobject
from sqlobject.sqlite import builder
conn = builder()('sqlobject_demo.db')
```

如果使用的是 MySQL 或带有服务器的其他数据库，则将数据库连接信息传递到连接构建器中。清单 2 提供了在上一节创建的 MySQL 数据库的示例。

清单 2. 传递 MySQL 数据库连接信息的代码

```
import sqlobject
from sqlobject.mysql import builder
conn = builder()(user='dbuser', passwd='dbpassword',
                 host='localhost', db='sqlobject_demo')
```

不管连接哪种数据库，连接代码都应该放置在一个名称类似 Connection.py 的文件中，且该文件存储在一些通常可访问的位置中。这样，可以导入您定义的所有类，并使用已经构建的 conn 对象。conn 变量将包含所有与数据库相关的详细信息。

但是要注意，SQLObject 的一些特性不可用于 SQLite 或 MySQL。不能将数据库选择与连接之后编写的代码完全分离。

定义模式

SQLObject 使得操作数据库表变得容易。看第一个简单的例子，考虑一个电话簿应用程序的由单个表组成的数据库模式，如表 1 所示。

表 1. phone_number 表的描述

字段	类型	说明
id	Int	主键
number	String	“(###) ###-####”字符串格式；应该惟一
owner	String	这是谁的号码？

last_call	Date	用户最后一次呼叫该号码是什么时候?

取决于您的 SQL 风格，该表的 SQL 类似如下：

```
CREATE TABLE phone_number (
  id INT PRIMARY KEY AUTO_INCREMENT,
  number VARCHAR(14) UNIQUE,
  owner VARCHAR(255),
  last_call DATETIME,
  notes TEXT
)
```

使用 SQLObject，不需要编写该 SQL 代码。通过定义 Python 类来定义该数据表。该代码将进入名为 PhoneNumber.py 的文件中，如清单 3 所示。

清单 3. PhoneNumber.py

```
import sqlobject
from Connection import conn
class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    number = sqlobject.StringCol(length=14, unique=True)
    owner = sqlobject.StringCol(length=255)
    lastCall = sqlobject.DateTimeCol(default=None)
PhoneNumber.createTable(ifNotExists=True)
```

在此使用了先前定义的 conn 变量。每个表类需要将对数据库连接的引用存储在它的 _connection 成员中。该信息隐式用于对该类的表的所有数据库访问中。因此，不必担心 SQL 或任何特殊数据库，因为代码可以按照抽象关系模式来表示。

定义表的类还有一组定义表字段的成员。SQLObject 提供了 StringCol、BoolCol 等等 —— 一个类对应一种数据库字段类型。

createTable() 方法第一次运行时，SQLObject 将创建一个名为 phone_number 的表。然后，它将只使用该表，因为您将 ifNotExists 设置为 True 来调用该方法。

最后注意，无需在 PhoneNumber 中为 id 字段创建字段对象。因为 SQLObject 总是需要该字段对象，所以它总会创建一个。

处理旧 CRUD

著名的缩写词 *CRUD* 代表对数据库行进行的四种操作：Create、Read、Update 和 Delete。定义了与数据库表对应的类之后，SQLObject 将对表行的操作表示为对类及其实例的操作。

创建

要创建数据库行，需创建对应类的实例，代码如下：

```
>>> from PhoneNumber import PhoneNumber
>>> myPhone = PhoneNumber(number='(415) 555-1212',
owner='Leonard Richardson')
```

现在 phone_number 表有一个存储我的姓名的行。PhoneNumber 构造函数将表列的值作为关键字参数。它使用您提供的数据创建一行 phone_number。如果由于某种原因，数据不能进入数据库，则构造函数抛出一个异常。当电话号码无效时，会发生如下情况：

清单 4. 无效的电话号码

```
>>> badPhone = PhoneNumber()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
...
TypeError: PhoneNumber() did not get expected keyword argument
number
```

如果电话号码已经在数据库中，将会看到：

清单 5. 电话号码已经在数据库中

```
>>> duplicatePhone = PhoneNumber(number="(415) 555-1212")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
...
TypeError: PhoneNumber() did not get expected keyword argument owner
```

读取（和查询）

SQLObject 类的实例的所有字段可用作成员。这与其他一些将数据库行当作字典的数据库映射工具相反。因此，对于数据库行中的每个字段，PhoneNumber 对象都具有一个成员。

清单 6. 可用作成员的字段

```
>>> myPhone.id
1
>>> myPhone.owner
'Leonard Richardson'
>>> myPhone.number
```

```
'(415) 555-1212'
>>> myPhone.lastCall == None
True
```

但如何检索数据库中已经存在的 PhoneNumber 对象呢？需要对数据库执行查询来获取。这就是 SQLObject 的查询构造工具包（该软件包最有趣的特性之一）发挥作用的地方。它允许将 SQL 查询表示为 Python 对象链。如果您熟悉 Java™ 编程语言的对象关系包的话，它与可以对 Torque 中 Criteria 对象执行的操作一样。

您定义的每个 SQLObject 类都有一个 select() 方法。该方法接受一个定义查询的对象，返回与该查询匹配的项列表。例如，下面这个方法调用返回包含数据库中第一个电话号码的列表：

清单 7. select() 方法

```
>>> PhoneNumber.select(PhoneNumber.q.id==1)
<sqlobject.main.SelectResults object at 0xb7b76cac>
>>> PhoneNumber.select(PhoneNumber.q.id==1)[0]
<PhoneNumber 1 number='(415) 555-1212' lastCall=None
owner='Leonard Richardson'>
```

PhoneNumber.q.id 指明想要对 phone_number 表的 id 字段运行查询。SQLObject 过载比较操作符（==、!=、<、>= 等等）来执行除布尔表达式之外的查询。表达式 PhoneNumber.q.id==1 是与 id 字段值为 1 的每行相匹配的查询。更多示例如下：

清单 8. 更多示例

```
>>> PhoneNumber.select(PhoneNumber.q.id < 100)[0]
<PhoneNumber 1 number='(415) 555-1212' lastCall=None
owner='Leonard Richardson'>
>>> PhoneNumber.select(PhoneNumber.q.owner=='Leonard
Richardson').count()
1
>>>
PhoneNumber.select(PhoneNumber.q.number.startswith('(415)')).count()
1
```

可以使用 SQLObject 的 AND 和 OR 函数来组合查询子句：

清单 9. 用于组合查询子句的 AND 和 OR 函数

```
>>> from sqlobject import AND, OR
>>> PhoneNumber.select(AND(PhoneNumber.q.number.startswith('(415)'),
>>>                        PhoneNumber.q.lastCall==None)).count()
```

```
1
```

下列查询获取一年中呼叫过的所有人以及从未呼叫过的所有人：

清单 10. 一年中呼叫过的所有人以及从未呼叫过的所有人

```
>>> import datetime
>>> oneYearAgo = datetime.datetime.now() -
datetime.timedelta(days=365)
>>> PhoneNumber.select(OR(PhoneNumber.q.lastCall==None,
...                       PhoneNumber.q.lastCall <
oneYearAgo)).count()
1
```

更新

如果更改 PhoneNumber 对象的一个成员，则该更改被自动镜像映射至数据库：

清单 11. 更改被自动镜像映射至数据库

```
>>> print myPhone.owner
Leonard Richardson
>>> print myPhone.lastCall
None
>>> myPhone.owner = "Someone else"
>>> myPhone.lastCall = datetime.datetime.now()
>>> #Fetch the object fresh from the database.
>>> newPhone = PhoneNumber.select(PhoneNumber.q.id==1)[0]
>>> print newPhone.owner
Someone else
>>> print newPhone.lastCall
2005-05-22 21:20:24.630120
```

这只是一个警告：SQLObject 不允许更改对象的主键。通常最好是让 SQLObject 来管理表的 id 字段，即使是您不使用 SQLObject 时碰巧用另一个字段作为主键。

删除

删除特定行对象的方法是将其 ID 传递到其类的 delete() 方法中：

清单 12. 删除行对象

```
>>> query = PhoneNumber.q.id==1
>>> print "Before:", PhoneNumber.select(query).count()
```

```
Before: 1
>>> PhoneNumber.delete(myPhone.id)
>>> print "After:", PhoneNumber.select(query).count()
After: 0
```

验证和转换数据

到目前为止，一直用 14 字符美国格式传递电话号码。数据库模式被设计成接受这种格式的数字，这暗示着底层应用程序 —— 不管怎样 —— 都希望数字是这种格式。尽管如此，代码不能确保笨拙的用户或程序员漏洞不会导致未正确格式化的数据放入 number 字段中。

SQLObject 通过允许定义验证和转换入站数据的钩子方法来解决这个问题。可以为表中的每个字段定义一个方法。字段的钩子方法命名为 _set_[field name]()，不管是作为 create 操作还是 update 操作的一部分，每当要为该字段设置一个值时，都会调用该方法。钩子方法应（可选）将入站值转换为可接受格式，然后设置该值。 否则，它应抛出异常。要实际设置一个值，该方法需要调用 SQLObject 方法 _SO_set_(field name)。

清单 4 展示了 PhoneNumber 的 _set_number() 方法。如果电话号码完全没有格式化，比如 4155551212，则该方法将该数字格式化为 (415) 555-1212。否则，如果数字格式不正确，该方法会抛出 ValueError。正确格式化的电话号码 —— 或者是转换为正确格式的电话号码 —— 被正确传递给 SQLObject 的 _SO_set_number() 方法。

清单 13. PhoneNumber 的 _set_number () 方法

```
   import re
   def _set_number(self, value):
      if not re.match('\([0-9]{3}\) [0-9]{3}-[0-9]{4}', value):
         #It's not in the format we expect.
         if re.match('[0-9]{10}', value):
            #It's totally unformatted; add the formatting.
            value = "(%s) %s-%s" % (value[:3], value[3:6],
value[6:])
         else:
            raise ValueError, 'Not a phone number: %s' % value
      self._SO_set_number(value)
```

定义表之间的关系

到目前为止，看到的所有操作都针对单个表：phone_number。但真正的数据库应用程序通常具有多个相关表。SQLObject 允许将表之间的关系定义为外键。作为演示，我们将一个小的数据库规范化（normalization）应用于上一示例，将 PhoneNumber 的

owner 字段分割到单独的 person 表中。清单 14 所示的代码保存在名为 PhoneNumberII.py 的文件中。

清单 14. PhoneNumberll.py 的代码

```
import sqlobject
from Connection import conn
class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    number = sqlobject.StringCol(length=14, unique=True)
    owner = sqlobject.ForeignKey('Person')
    lastCall = sqlobject.DateTimeCol(default=None)
class Person(sqlobject.SQLObject):
    _idName='fooID'
    _connection = conn
    name = sqlobject.StringCol(length=255)
    #The SQLObject-defined name for the "owner" field of PhoneNumber
    #is "owner_id" since it's a reference to another table's primary
    #key.
    numbers = sqlobject.MultipleJoin('PhoneNumber',
joinColumn='owner_id')
Person.createTable(ifNotExists=True)
PhoneNumber.createTable(ifNotExists=True)
```

该 PhoneNumber 类具有与旧类相同的成员，但它的 owner 成员是对 person 表的主键的引用，而不是对 phone_number 表中字符串列的引用。这使得表示具有两个电话号码的个人成为可能：

清单 15. 表示具有两个电话号码的个人

```
>>> from PhoneNumberII import PhoneNumber, Person
>>> me = Person(name='Leonard Richardson')
>>> work = PhoneNumber(number="(650) 555-1212", owner=me)
>>> cell = PhoneNumber(number="(415) 555-1212", owner=me)
```

Person 的 numbers 成员，一个 SQLObject MultipleJoin，使得基于 person 到 phone_number 的连接进行查询变得容易：

清单 16. 基于 person 到 phone_number 的连接的查询

```
>>> for phone in me.phoneNumbers:
...     print phone.number
...
(650) 555-1212
(415) 555-1212
```

同样，SQLObject 允许使用 MultipleJoin 类进行多对多连接的查询。

将 SQLObject 用于现有表

SQLObject 的一个常见用途是为另一个应用程序创建的数据库提供 Python 接口。SQLObject 有多个特性可用于实现这一点。

数据库内省

如果正在使用数据库中已经存在的表，则不需要在 Python 中定义列。SQLObject 可以通过数据库内省来提取它需要的信息。例如，清单 17 中的代码保存在 PhoneNumberIII.py 中。

清单 17. PhoneNumberlll.py 的代码

```
import sqlobject
from Connection import conn
class PhoneNumber(sqlobject.SQLObject):
    _connection = conn
    _fromDatabase = True
```

该类将使用现有 phone_number 数据表的属性。您可以与它交互，就好像已经手动定义了该类及其所有列一样，如前面的示例所示。使用 SQLObject，只需要编写表定义一次 —— 用 SQL 还是用 Python 编写就取决于您了。

但是，该特性又带来了数据库的选择问题。例如，该特性完全不能用于 SQLite。它基本上能用于 MySQL，但不能提取外键关系。如果使用的是 MySQL，而且想要为表定义外键，则需要在从数据库中加载模式之后，编写代码定义这些字段。

命名约定

上一部分中的代码假设现有表符合 SQLObject 的命名约定（例如，表的主键字段名为 id，且列名中的词用下划线分隔）。表的命名约定在 Style 类中定义。

SQLObject 提供了一些与常见数据库命名约定对应的 Style 类。例如，如果列名类似 likeThis 而非 like_this，则可以使用 MixedCaseStyle：

清单 19. 使用 MixedCaseStyle

```
import sqlobject
from sqlobject.styles import MixedCaseStyle
from Connection import conn
class PhoneNumber(sqlobject.SQLObject):
```

```
_connection = conn
_fromDatabase = True
_style = MixedCaseStyle
```

如果没有预包装的 Style 类符合您的需要，那么您可以定义 Style 基类的子类，并定义自己的命名约定。在最坏的情况下，如果表的字段名分配得毫无道理，则可以逐个命名每个字段。

关于 SQLObject 限制

SQLObject 想让您用面向对象的方式而非关系方式进行思考。这有利于您的理解和您的编程生产率，但不利于性能。毕竟，数据库仍是关系型的。如何标记呼叫过的每个电话号码？使用 SQL，您将使用单个 UPDATE 命令。使用 SQLObject，您需要迭代通过整个结果集，并修改每个对象的 last_call 成员，这是非常低效的。

SQLObject 为开发人员时间牺牲了处理器时间。这通常是好的交易，但甚至在简单的应用程序中，您也可能需要下降一个级别到达 Python 数据库接口，为一些关键路径的操作编写原始 SQL。

结束语

本文广泛介绍了 SQLObject，但并不十分深入。它是一个万能工具，具有许多方便的小特性。它的限制易于理解，但可以通过编写需要的 SQL 来绕过它们。SQLObject 针对关系数据库编程而 Python 针对应用程序编程：一种用很少的时间完成工作的方便方法。

北 京 邮 电 大 学
本科毕业设计（论文）开题报告

学院	计算机学院	专业	网络工程	班级	14
学生姓名	马庆元	学号	08211543	班内序号	6
指导教师姓名	房鸣	所在单位	实验中心	职称	副教授
设计（论文）题目	基于控制流分析的代码反抄袭系统				

毕业设计（论文）开题报告内容：

现在，计算机科学已经由多年前的纯理论研究发展到今天，成为了一项实用性非常强的应用科技。计算机科学教学顺应该趋势，渐渐加强对实践教学的重视。在计算机科学实践教学中，程序设计联系是重要的一环。随着实践类教学比重的逐渐增加，程序设计类作业相对传统作业所占的比例越来越大，但由于学生提交程序多为电子版，抄袭变得前所未有的方便，并且由于程序代码的智力密集性，助教难以利用较少的劳动和时间发现抄袭行为，这在一定程度上对影响了实践环节的教学效果。因而，设计一个可以帮助助教分析学生的作业代码，快速搜寻抄袭证据的系统，对于揭发抄袭行为，吓阻抄袭意图，保证实践教学的效果而言，非常重要。本次研究旨在提供一套完整的解决方案，通过在控制流层面上分析学生提交的作业程序代码，帮助助教发现可能的抄袭行为。

目前，学界已经研发了一系列反抄袭系统，这些系统可以有效解决相当一部分抄袭问题。但是上述通用系统在处理非常具体的反抄袭任务时，往往不能充分利用任务的特殊性质和额外资源，从而难以达到期望的反抄袭效果。本次研究将着力发掘利用程序设计实践中的特殊性，针对其设计算法，充分利用这些性质，缩减计算资源需求，提高准确性和召回率。

另外，针对目前已有的各种反抄袭系统的弱点，学生很容易设计出易于操作的反-反抄袭策略，削弱反抄袭系统的效用。本次研究将调查学生常见的代码反-反抄袭的策略，有针对性的加强反抄袭系统，同时对已有的反抄袭算法本身进行算法理论分析，找出其弱点，尽量在新系统中予以解决，使新系统可以作为已有系统的补充选项。

在项目的具体实现上，本项目需要设计高精确度，高召回率的基于控制流分析的代码比对算法，要求可以抵抗常见的反-反抄袭手段，将抄袭成本提高到抄袭收益之上。同时方便易用的接口工具和 API，可以连接现有的实践平台，方便助教人员提取、整理分析数据，为检举抄袭提供充分依据。

指导教师签字		日期	2012 年 2 月　日

注：可根据开题报告的长度加页。

北 京 邮 电 大 学
本科毕业设计（论文）中期进展情况检查表

学院	计算机学院	专业	网络工程	班级	2008211314
学生姓名	马庆元	学号	08211543	班内序号	6
指导教师姓名	房鸣	职称	副教授		
设计（论文）题目	基于控制流分析的代码反抄袭系统				

目前已完成任务

主要内容:（毕业设计（论文）进展情况，字数一般不少于 1000 字）

1. 已经完成了反抄袭系统的输入数据库的 schema 的设计。并且构建了一个从配置文件，代码库和测试数据自动生成反抄袭系统运行所需的数据包的工具。
2. 已经完成了对 GCC 工具 GCOV(GNU 覆盖率测试工具)的修改，使其可以自动化地从 GCC 编译的包含覆盖率测试的二进制中提取该反作弊系统工作所需的代码控制流数据。
3. 已经设计并实现了控制流图化简和预处理的算法，减少控制流信息中与相似度匹配无关的各种噪音。
4. 已经设计和实现 2 种算法，将控制流图的信息量化，生成适合相似度计算的中间数据。
5. 已经设计并实现了 2 个控制流相似度计算算法：BASIC 和 ADVANCED，BASIC 利用一次类似 LCA 的动态规划和一次简单的距离计算来确定相似度，ADVANCED 引入了更多信息，利用控制流的前向分支信息 2 次类似 LCA 的动态规划确定相似度。BASIC 算法较快，ADVANCED 算法在复杂的控制流上有更佳的效果。
6. 已经完成了基本的报告输出模块。
7. 已经完成了系统的自动化运行控制工具，只要一个指令就可以执行完整的运行-分析-报告过程。

是否符合任务书要求进度 **是**

<table>
<tr><td rowspan="2">尚需完成的任务</td><td colspan="2">1. 设计一个从 PHP+MySQL 系统中自动生成反抄袭系统输入数据库的参考程序。
2. 设计匹配算法 INSANE，利用匈牙利算法代替 ADVANCED 算法中的第二次类 LCA 算法来计算相似度。</td></tr>
<tr><td colspan="2">能否按期完成设计（论文） 是</td></tr>
<tr><td rowspan="2">存在问题和解决办法</td><td>存在问题</td><td>1. ADVANCED 算法和即将设计的 INSANE 算法非常慢，对于一对 100 行左右的代码，ADVANCED 算法需要大约 2 秒才能给出结果，预计 INSANE 算法会更慢。

2. 目前的反抄袭系统不能良好的对很简单的程序给出相似度，因为逻辑过于简单，几乎所有可以达到目标的实现的确采用了非常相似的算法。基于控制流的算法不能分辨“诚信的”代码间的区别从而误判。</td></tr>
<tr><td>拟采取的办法</td><td>1. 优化算法的代码，如果必要，加入分布式设计，将负载分散在集群中。
2. 将更多的特征引入 ADVANCED 和 INSANE 算法，一旦检测到程序过于简单就启用这个算法，只有发现文本过于相似才给出警告。</td></tr>
<tr><td>指导教师签字</td><td></td><td>日期</td><td>2012 年 月 日</td></tr>
<tr><td>检查小组意见</td><td colspan="3">负责人签字： 2012 年 月 日</td></tr>
</table>

注：可根据长度加页；一式二份，学院、学生各一份。

北 京 邮 电 大 学

本科毕业设计（论文）信息卡

<table>
<tr><td>学院</td><td>计算机学院</td><td>专业</td><td>网络工程</td><td>班级</td><td>2008211314</td></tr>
<tr><td>学生姓名</td><td>马庆元</td><td>学号</td><td>08211543</td><td>班内序号</td><td>6</td></tr>
<tr><td>指导教师姓名</td><td>房鸣</td><td>职称</td><td colspan="3">副教授</td></tr>
<tr><td colspan="6">第 1－2 周记录：

收集样例数据，理解需求。</td></tr>
<tr><td>指导教师签字</td><td colspan="2"></td><td>日期</td><td colspan="2">年 月 日</td></tr>
<tr><td colspan="6">第 3－4 周记录：

调研现有的反抄袭系统，观察其运作原理。</td></tr>
<tr><td>指导教师签字</td><td colspan="2"></td><td>日期</td><td colspan="2">年 月 日</td></tr>
</table>

注：每 2 周内容记录在一个表格中，双面打印。

北 京 邮 电 大 学

本科毕业设计（论文）信息卡

学院	计算机学院	专业	网络工程	班级	2008211314
学生姓名	马庆元	学号	08211543	班内序号	6
指导教师姓名	房鸣	职称	副教授		

第 5－6 周记录：

设计系统框架，尝试将旧系统嵌入，进行部分测试。

指导教师签字		日期	年 月 日

第 7－8 周记录：

修改 GCOV 使其符合项目需求。

指导教师签字		日期	年 月 日

注：每 2 周内容记录在一个表格中，双面打印。

北 京 邮 电 大 学

本科毕业设计（论文）信息卡

学院	计算机学院	专业	网络工程	班级	2008211314
学生姓名	马庆元	学号	08211543	班内序号	6
指导教师姓名	房鸣	职称	副教授		

第 9－10 周记录：

编写输入预处理模块。

指导教师签字		日期	年 月 日

第 11－12 周记录：

编写自动化控制流图抽取模块。

指导教师签字		日期	年 月 日

注：每 2 周内容记录在一个表格中，双面打印。

北 京 邮 电 大 学

本科毕业设计（论文）信息卡

<table>
<tr><td>学院</td><td>计算机学院</td><td>专业</td><td>网络工程</td><td>班级</td><td>2008211314</td></tr>
<tr><td>学生姓名</td><td>马庆元</td><td>学号</td><td>08211543</td><td>班内序号</td><td>6</td></tr>
<tr><td>指导教师姓名</td><td>房鸣</td><td>职称</td><td colspan="3">副教授</td></tr>
<tr><td colspan="6">第 13－14 周记录：

编写相似度检查器框架，设计，实现相似度检查算法。</td></tr>
<tr><td>指导教师签字</td><td colspan="2"></td><td>日期</td><td colspan="2">年 月 日</td></tr>
<tr><td colspan="6">第 15－16 周记录：

完成撰写论文。</td></tr>
<tr><td>指导教师签字</td><td colspan="2"></td><td>日期</td><td colspan="2">年 月 日</td></tr>
</table>

注：每 2 周内容记录在一个表格中，双面打印。