《软件系统设计》- 迭代三

截止时间–2024.5.12 23:59:59

1. 总览

在迭代一和迭代二中，大家已经实现了一个初步的在线评测系统（Online Judge），并已经实现了部分功能。本次迭代的重点是对迭代二中的编程题评测进一步扩展，要求实现一种代码性能评测限制指标**【时间复杂度】**和一种代码风格评测指标**【圈复杂度】**。

本次依旧是以压缩包形式提交，其中应该包含有**项目代码**、三次迭代的**设计文档**和 Git **提交记录（.git文件夹）**，其中每次设计文档中需要包含**类图**，本次迭代希望大家给出尽可能完整的类图。另外，在最终提交前，我们也强烈建议你综合所学设计思想，对三次迭代中的设计加以反思，你可以修改前两次迭代中不恰当的设计。

2. 功能需求

2.1. 时间限制

在代码（编程题作答结果）的执行阶段，需要对运行时间进行计算，对**超出时间限制的作答结果判0分**。对于具体时间限制，在第一次迭代中，编程题题目理论上预留了对应的timeLimit字段，单位为ms。同时，对于超出时间限制的执行线程，你需要采用合适的方式将其**终止**，避免因为潜在的死循环等情况影响整个评测系统。对于超时的具体输出，我们不做要求。

2.2. 圈复杂度

圈复杂度直接关联到程序源代码的复杂性，它通过评估程序中的决策点（如条件判断、循环等）来计算。一般来说，圈复杂度越高，程序的维护成本和出错概率通常也越高。另外，圈复杂度也能够指导测试用例的编写，好的设计经验通常是创建数量与被测代码圈复杂度值相等的测试用例，确保分支覆盖率。

在本次迭代中，我们不要求基于控制流图的严谨实现，而是采用**节点判定法**，因为圈复杂度直观上就是判定条件数量的反映。具体来说，你可以借助 Javaparser、antlr 等工具精确识别判定节点，其中仅需要你实现的基本判定规则如下：

1. if 语句，对于多分支的 if – else if – else 结构，每个 else if 语句，都应该算为一个判定节点。
2. while 语句、do–while 语句和 for 语句。
3. 三元运算符。
4. and(&&) 和 or(||) 布尔运算。
5. 最终圈复杂度是判定节点总数+1。

另外，在相对复杂的编程题目中，一般不会只包含单个函数实现。因此，我们要求以**相对统一**的方式分别实现单个函数和整个类的圈复杂度实现。在本次迭代中，我们简单规定整个类的圈复杂度为其内部所有函数的圈复杂度之和。

下面是一个例子：

- main函数圈复杂度：1(if) + 1(||) + 1 = 3
- divide函数圈复杂度：1(while) + 1 = 2
- 总圈复杂度：3 + 2 = 5

圈复杂度案例 Java

```java
public static void main(String[] args) {
    assert args.length == 2;
    int dividend = Integer.valueOf(args[0]);
    int divisor = Integer.valueOf(args[1]);
    if (dividend == 0 || divisor == 1) {
        System.out.println(dividend);
    } else {
        System.out.println(divide(dividend, divisor));
    }
}

private static int divide(int dividend, int divisor) {
    int quotient = 0;
    while (dividend >= divisor) {
        dividend -= divisor;
        quotient++;
    }
    return quotient;
}
```

3. 测试用例

关于测试用例部分，迭代一和迭代二中的规定仍然不变：

> 在迭代一中，我们将提供给你一个文件夹路径，其中包含两个子文件夹，分别是exams与answers，其中exams文件夹中包含若干考试文件，格式为JSON或XML，answers中包含若干回答文件，格式为JSON，你需要在读取exams文件夹内所有考试文件后为answers文件夹内的每一份回答文件打分，并将打分结果输出为csv文件。
>
> 关于编程题测试用例的一些额外规定：
>
> - 编程题的测试用例在exams文件夹中的考试文件中，具体为编程题的**samples属性**，测试用例只包含**input**和**output**两项，input为需要传递给编程题代码的参数，output则为期望的输出。
> - 编程题的答案保存在文件中，文件名则被包含在学生的答案中，为了统一和简化，**编程题的答案文件会被放置在answers/code-answers文件夹下，类名为Solution{xx}，只包含main函数**，所需参数由main函数的参数传入。

本次迭代依旧需要在上次的代码上迭代开发，因此本次迭代我们仍然只提供测试用例，大家只需要将本次的测试用例替换掉迭代二中的即可，也就是把解压后的 test 文件夹替换掉src/test。

4. 用例评分

编程题需要测试用例全部通过，任一一个测试用例不过或超时均为0分，圈复杂度**不影响**具体编程题的评分。对于圈复杂度，你需要输出一个和之前评分结果相似的 csv 文件，命名为 output_complexity.csv，这个 csv 文件中应包含四列，分别是 examId、stuId、qId（题目序号）与 complexity（本题作答总圈复杂度），你同样需要为 csv 文件输出表头。当编译不通过时，圈复杂度默认-1。

其他评分的具体细则和迭代一和迭代二保持一致。

5. 提交方式

1. 为了尽可能符合设计原则，请不要将本地路径硬编码在代码中，如果你需要放一些文件，请放在测试目录下。
2. 你需要使用git来管理你的代码，请在开发的过程中遵守git的提交规范，我们会通过git提交记录跟踪你的设计过程，每次迭代至少需要有一次提交记录，并且将每次迭代的最终版本的git提交信息设置为**iter{n}_finish**（如第一次迭代为iter1_finish）。
3. 每下一次迭代需要在上一次迭代的最终版上继续更改，在三次迭代结束后我们会检查完整的三次迭代的git提交信息，你可以对前面迭代中不恰当的设计或实现加以修改。
4. 除了本地的git追踪外，你还需要在https://git.nju.edu.cn/上**创建git仓库进行远程的备份**，仓库命名为**software_design_学号**，我们会对项目实现、设计文档和提交记录进行综合检查。
5. 本次迭代依旧是以压缩包形式提交，请确保其中包含有**项目代码**、三次迭代的**设计文档**和 Git **提交记录**。
6. 为了控制提交大小，**请务必通过 `.gitignore` 过滤build、target等产物目录**。