

软件测试上机报告



第三次上机作业 - Lab 4 Major

学 院 智能与计算学部

专 业 软件工程

姓 名 郎文翀

学 号 **3019244247**

年 级 2019 级

班 级 软件工程 5 班

1. Experimental requirements

1. Install Major Mutation Framework. The instruction of how to install and use Major can be seen in <http://mutation-testing.org/doc/major.pdf>
2. Coding a program named 'UpgradedTriangle'. Given the length value (integer) of 3 sides of a triangle. Finish 2 functions respectively, (1) classifying the triangle and (2) calculating the area of valid triangle.

In function (1), given 3 length of sides(integers), output the shape of triangle made up by given sides. (Output a String, the shape could be "SCALENE","EQUILATERAL","ISOSCELES","INVALID".)

In function (2), given 3 length of sides(integers), if these 3 sides can make up a valid triangle, output the area of the triangle (double or float), otherwise, return 0. (reference : Heron's formula)

3. Write testing cases for 2 functions with Junit according to your previous study (MC/DC, boundary value, equivalence partitioning, etc.), guarantee the sufficiency and diversity of your test set. Each function should have at least 10 test cases. Then run mutants on the test sets with Major Mutation Framework.
4. Analyzing the report provided by Major. Calculate these values:
 - The number of mutants generated

- The number of mutants covered by the test suite
- The number of mutants killed by the test suite
- The number of live mutants
- The overall mutation score / adequacy of the test suite

Discuss and explain your results: (Here are some Viewpoints you could discuss)

- What do the results tell you about your test suite?
- Does the test suite exhibit weaknesses? How can it be improved?
- Does the test suite exhibit strengths? How do you recognize them?
- Do you have any other interesting insights or opinions on the experience?
- Among the generated mutants, If both killed and unkilld mutants were generated, what was the type of operator used? How was it applied to the code (how did the code change)?
- According to your mutation analysis result, which part of the source code need to be strengthened in further coding? Which test case in your suite are more important compared with others.

Requirements for the experiment:

1. Finish the tasks above individually.
2. Post your experiment report to “智慧树” , the following information should be included in your report:
 - a) The brief description that you install Major and its configuring

process.

- b) Steps for generating Mutants
- c) Steps for making test sets and running mutants.
- d) Your mutants results listed in task 4 (The number of live mutants, killed mutants, etc.)
- e) Your discussion based on provided report. (It is listed in Task4)

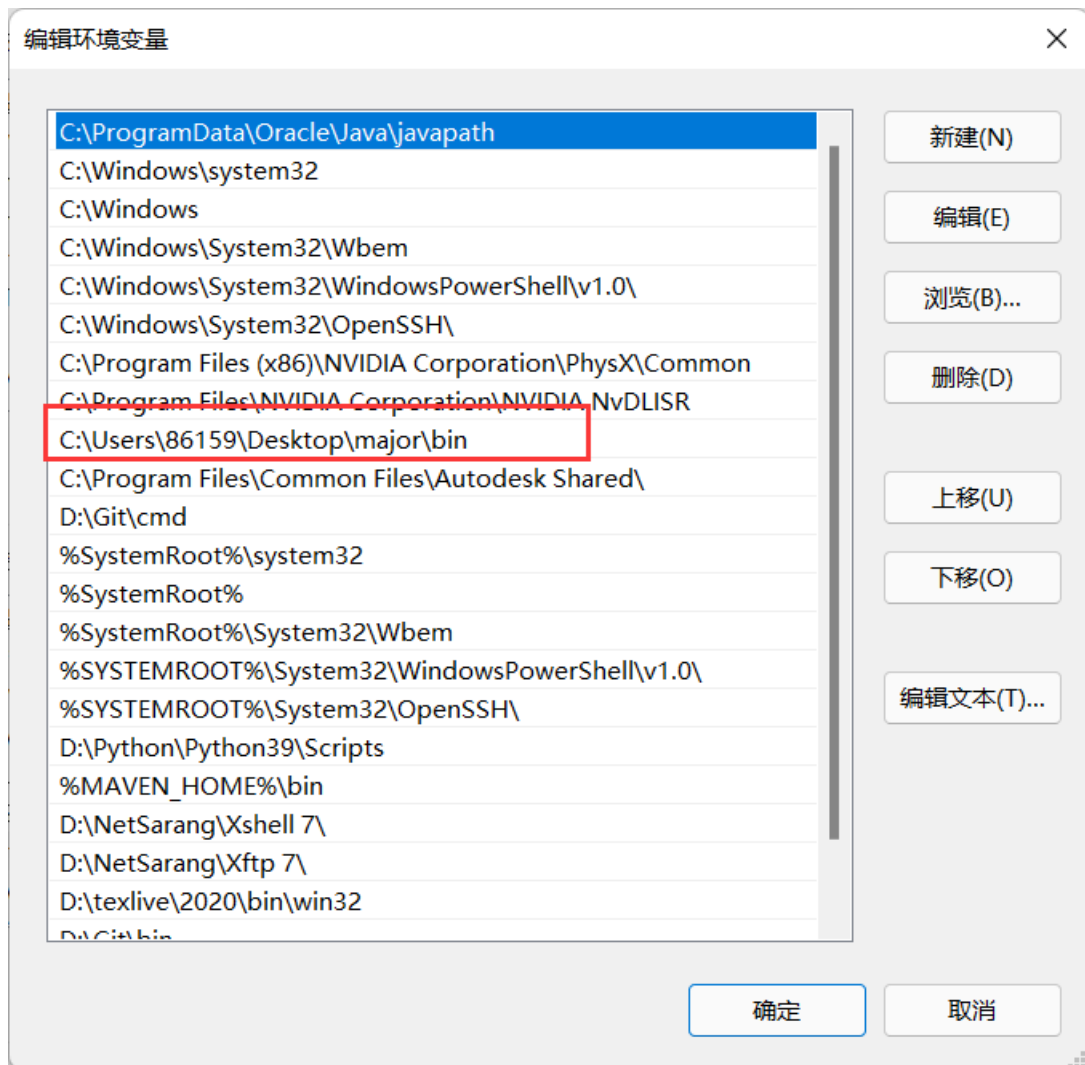
2. Configuration

2.1 实验前学习

首先我们解压缩 `major.zip` 文件得到一个 `major` 文件夹，这个文件夹就是用来进行变异体测试的，我们打开文档中的文档阅读以后可以知道他提供了两种方式进行变异体测试，分别用 `example` 中的 `standalone` 和 `ant` 进行了演示，并且还介绍了一些文件以及命令的使用，比如首先在 `bin` 文件夹下提供了需要使用的 `ant`, `javac`, `mml` 等指令集，然后我们再配置完成环境以后即可通过打开命令窗逐行输入命令进行变异体的生成、测试、分析结果导出等操作，但是也可以通过创建 `sh` 脚本文件一次性执行完所有相关的操作，并且将需要的结果分析文件导出进行分析。这里我选择了使用了 `ant` 来进行实验的变异体生成、测试以及结果分析。

2.2 配置实验环境

我们首先需要保证当前 `major` 下的 `javac` 环境是 `javac 1.7.0-Major-v1.3.5` 才能进行接下来的实验，这里我们本地 `win` 下装的 `javac` 环境是 `1.8` 不是要求的带有 `Major` 驱动的，因此需要去环境变量下重新新建一个 `Path` 来存储这个 `major/bin` 文件夹下的 `javac` 环境，同时还要注意将这个 `path` 进行优先级升级，至少要高于全局配置的 `javac1.8`，如下图所示：



同时我们还要保证本地命令窗使用 `javac` 进行源程序文件编译时不会出现乱码导致的编译失败问题，如果出现这种问题请自行百度，很好解决，如果出现下图所示则说明环境正常：

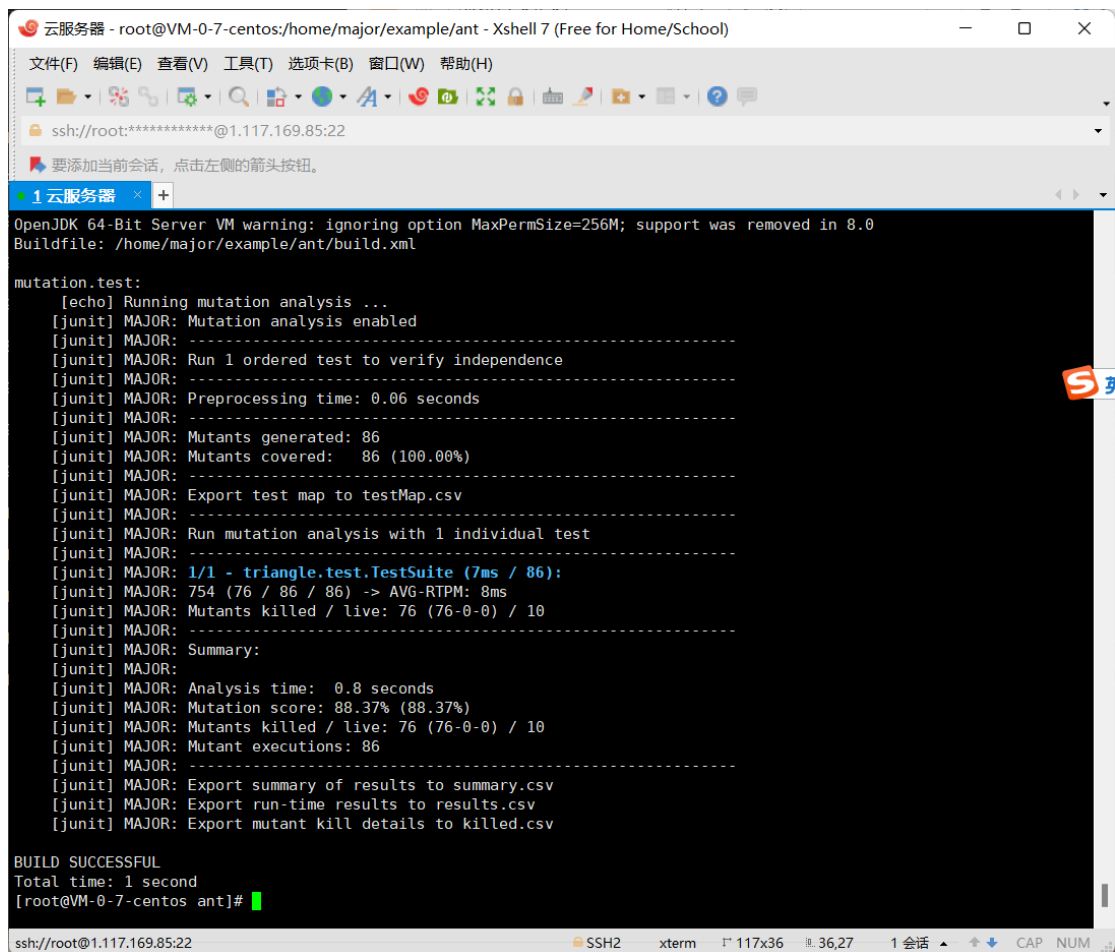
```
MINGW64:/c/Users/86159/Desktop/major
86159@种的大显示屏 MINGW64 ~/Desktop/major
$ javac
用法: javac <options> <source files>
其中, 可能的选项包括:
-g          生成所有调试信息
-g:none     不生成任何调试信息
-g:{lines,vars,source} 只生成某些调试信息
-nowarn     不生成任何警告
-verbose    输出有关编译器正在执行的操作的消息
-deprecation 输出使用已过时的 API 的源位置
-classpath <路径> 指定查找用户类文件和注释处理程序的位置
-cp <路径> 指定查找用户类文件和注释处理程序的位置
-sourcepath <路径> 指定查找输入源文件的位置
-bootclasspath <路径> 覆盖引导类文件的位置
-extdirs <目录> 覆盖所安装扩展的位置
-endorseddirs <目录> 覆盖签名的标准路径的位置
-proc:{none,only} 控制是否执行注释处理和/或编译。
-processor <class1>[,<class2>,<class3>...] 要运行的注释处理程序的名称; 绕过默认
的搜索进程
-processorpath <路径> 指定查找注释处理程序的位置
-d <目录> 指定放置生成的类文件的位置
-s <目录> 指定放置生成的源文件的位置
-implicit:{none,class} 指定是否为隐式引用文件生成类文件
```

当然由于各种适配原因（比如 `javac` 在本地编译出现了启动不起来的问题）我最终选择了在自己的腾讯云云服务器上进行实验，操作系统为 `centOS7.6`, 打开 `xshell` 和 `xftp` 连接服务器以后我再 `/home/major` 下进行实验。自此我们已经成功配置好了实验所需要的带有 `major` 驱动的 `javac` 环境。在云服务器上不需要配置环境变量，只需要输入 `./bin/javac -version` 可以看到 `javac` 环境配置就是符合要求的：

```
BUILD SUCCESSFUL
Total time: 1 second
[root@VM-0-7-centos ant]# ../../bin/javac -version
javac 1.7.0-Major-v1.3.5
[root@VM-0-7-centos ant]#
```

2.3 实验项目创建

我们看完题意，知道了实际上就是让我们编写一个源程序文件要求有两个功能函数，一个是可以对合法的输入进行判断是何种类型的三角形，还有一个功能函数就是对合法的输入使用 `heron` 公式进行三角形面积的计算，当不符合要求是就返回 `0.0`。在 `example/ant` 下实际上已经给出了一个用 `ant` 测试三角形类型的函数变异体测试实例，我们只需要直接执行 `sh run.sh` 就可以查看到他这个示例的效果如下：



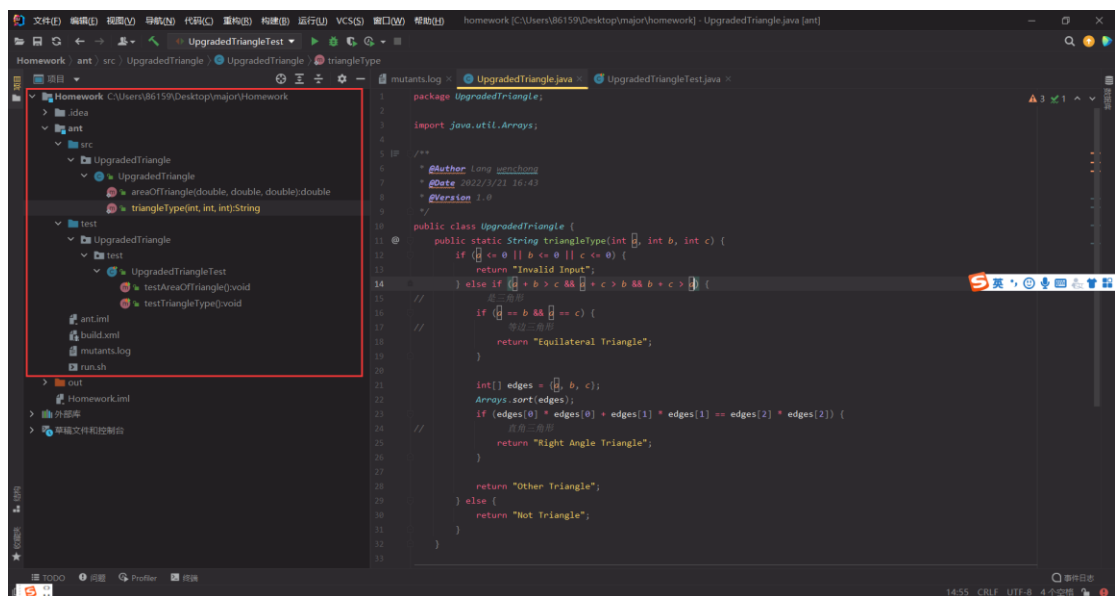
```
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=256M; support was removed in 8.0
Buildfile: /home/major/example/ant/build.xml

mutation.test:
[echo] Running mutation analysis ...
[junit] MAJOR: Mutation analysis enabled
[junit] MAJOR: -----
[junit] MAJOR: Run 1 ordered test to verify independence
[junit] MAJOR: -----
[junit] MAJOR: Preprocessing time: 0.06 seconds
[junit] MAJOR: -----
[junit] MAJOR: Mutants generated: 86
[junit] MAJOR: Mutants covered: 86 (100.00%)
[junit] MAJOR: -----
[junit] MAJOR: Export test map to testMap.csv
[junit] MAJOR: -----
[junit] MAJOR: Run mutation analysis with 1 individual test
[junit] MAJOR: -----
[junit] MAJOR: 1/1 - triangle.test.TestSuite (7ms / 86):
[junit] MAJOR: 754 (76 / 86 / 86) -> AVG-RTPM: 8ms
[junit] MAJOR: Mutants killed / live: 76 (76-0-0) / 10
[junit] MAJOR: -----
[junit] MAJOR: Summary:
[junit] MAJOR: -----
[junit] MAJOR: Analysis time: 0.8 seconds
[junit] MAJOR: Mutation score: 88.37% (88.37%)
[junit] MAJOR: Mutants killed / live: 76 (76-0-0) / 10
[junit] MAJOR: Mutant executions: 86
[junit] MAJOR: -----
[junit] MAJOR: Export summary of results to summary.csv
[junit] MAJOR: Export run-time results to results.csv
[junit] MAJOR: Export mutant kill details to killed.csv

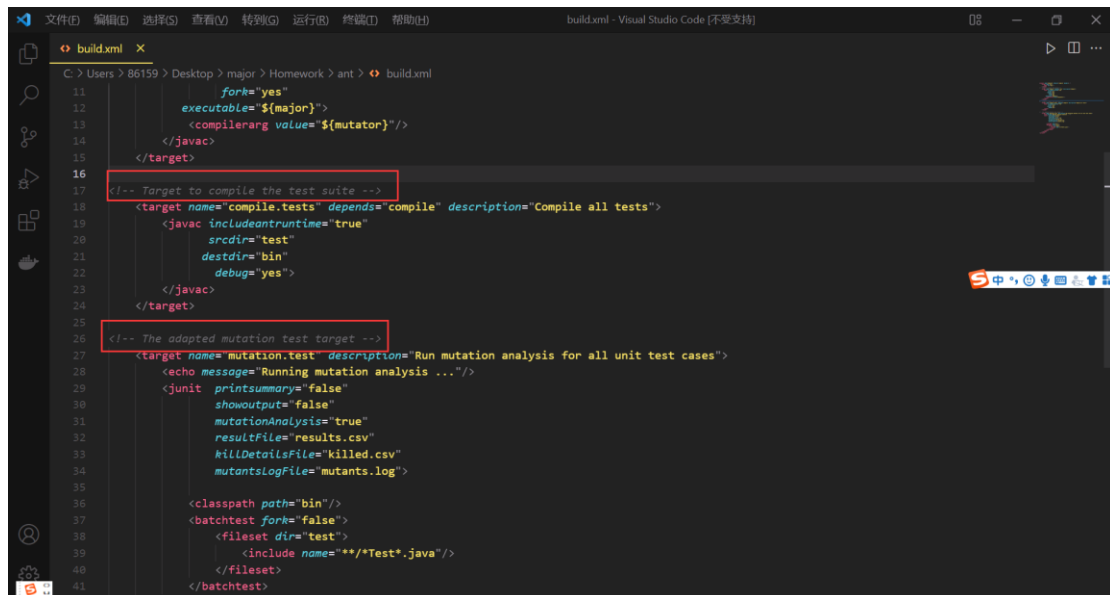
BUILD SUCCESSFUL
Total time: 1 second
[root@VM-0-7-centos ant]#
```

他一共生成了 76 个变异体，然后变异体覆盖率为 100%，即所有的变异体都进行了 junit 测试，同时杀死率为 88.37（杀死 76，存活 10）。我们查看他的文件可以看到他的测试样例有 33 个之多，因此杀死率很高了，样例具有一定的错误检测能力，再补充一下就可以达到需求了。

但是我并没有再其基础上添加我的修改代码，我是在 major 下新建了一个和 example 同级的 Homework 项目，同时使用 ant 进行测试，因此我的 Homework 项目结构和 example 很类似，如下所示：



具体的代码请查看第四部分，接下来我们编写好自己的源程序代码文件 UpgradedTriangle.java 和测试文件 UpgradedTriangleTest.java 以后我们就可以进行测试了。我们只需要如下操作：首先将示例代码中的 web.xml 和 run.sh 文件拷贝过来，因为我们也是会用类似的命令去生成我们的变异体，然后在进行测试。但是我们需要略加修改，我们需要将 web.xml 中指向的测试文件名改为 UpgradedTriangle 这样才会为我们编写的文件进行变异体生成。同时我们大概 web.xml 中进行分析可以看到他将不同的操作都封装成了一个 target:



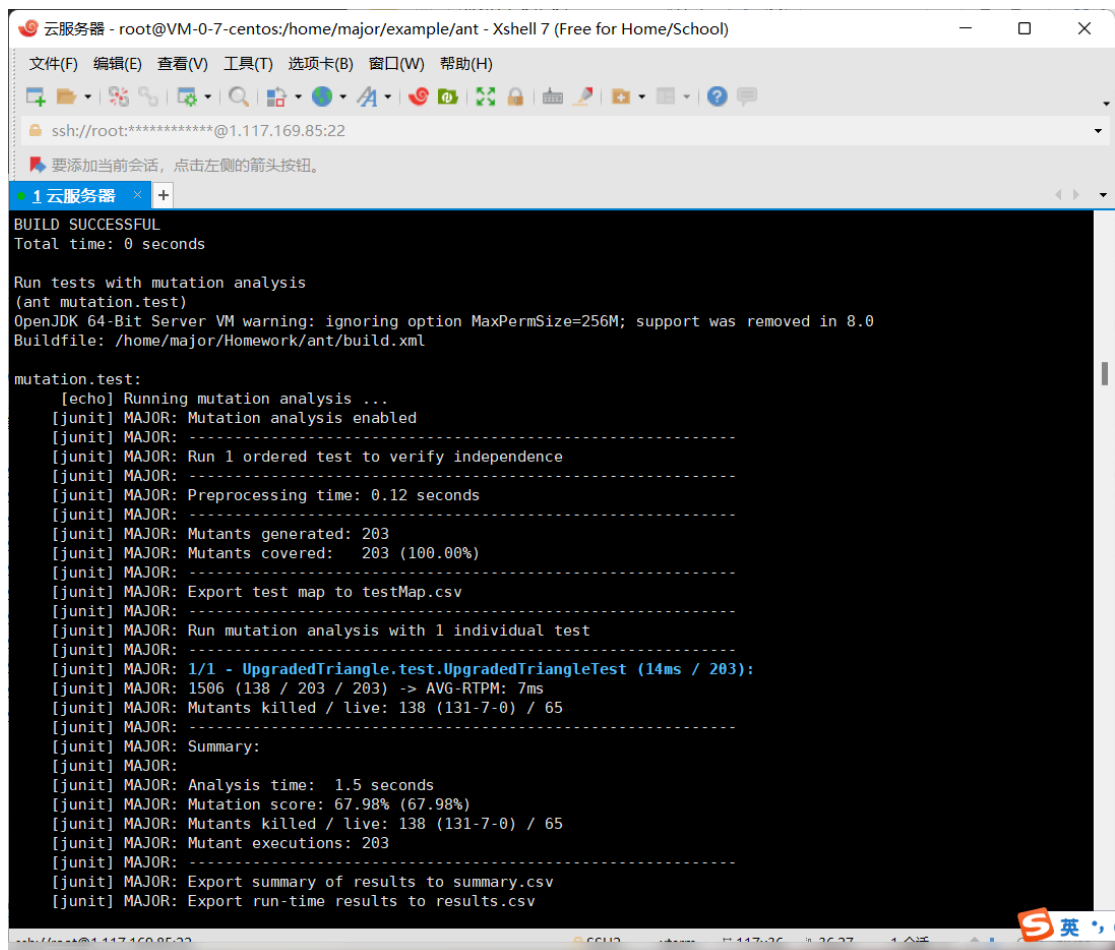
然后将不同的 target 取了一个别名，紧接着我们再打开 run.sh 脚本文件可以看到实际上脚本文件中的命令就是在使用 ant 指令顺序的调用这几个 target 从而实现自动化的完成删除之前的分析结果-编译源程序-生成变异体-编译测试文件-将源程序进行测试-变异体进行测试-可视化数据导出的全流程操作。因此接下来里我们也对我们的项目进行这些操作。

2.4 项目运行结果

但是当我执行 sh ru.sh 以后发现并没有出现预期的结果，发现变异体生成的数量居然为 0，这显然是除问题了。原因就出现在了 run.sh 脚本文件中，这里我取出问题代码：

```
echo
echo "Compiling and mutating project"
echo "(ant -DmutOp=\"=\${MAJOR_HOME}/mml/tutorial.mml.bin\" clean compile)"
echo
$MAJOR_HOME/bin/ant -DmutOp=\"=\${MAJOR_HOME}/mml/tutorial.mml.bin\"
clean compile
```

我们发现他指向变异体生成的指令配置选项使用的是 tutorial.mml.bin，实际上这个是专门为 example/ant 下的 Triangle.java 使用的变异体测试配置，这里我们需要进行修改，我们是对 UpgradedTriangle.java 中的所有方法都进行变异体测试，同时选择所有的变异算子，因此我们应该将-DmutOp 后面的配置项改为 ALL 即可，修改完保存退出，然后我们再重新执行 sh run.sh 启动我们的脚本文件就可以看到正确的结果了，如下是我的运行截图：



```
云服务器 - root@VM-0-7-centos:/home/major/example/ant - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://root:*****@1.117.169.85:22
要添加当前会话，点击左侧的箭头按钮。
1 云服务器
BUILD SUCCESSFUL
Total time: 0 seconds

Run tests with mutation analysis
(ant mutation.test)
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=256M; support was removed in 8.0
Buildfile: /home/major/Homework/ant/build.xml

mutation.test:
[echo] Running mutation analysis ...
[junit] MAJOR: Mutation analysis enabled
[junit] MAJOR: -----
[junit] MAJOR: Run 1 ordered test to verify independence
[junit] MAJOR: -----
[junit] MAJOR: Preprocessing time: 0.12 seconds
[junit] MAJOR: -----
[junit] MAJOR: Mutants generated: 203
[junit] MAJOR: Mutants covered: 203 (100.00%)
[junit] MAJOR: -----
[junit] MAJOR: Export test map to testMap.csv
[junit] MAJOR: -----
[junit] MAJOR: Run mutation analysis with 1 individual test
[junit] MAJOR: -----
[junit] MAJOR: 1/1 - UpgradedTriangle.test.UpgradedTriangleTest (14ms / 203):
[junit] MAJOR: 1506 (138 / 203 / 203) -> AVG-RTPM: 7ms
[junit] MAJOR: Mutants killed / live: 138 (131-7-0) / 65
[junit] MAJOR: -----
[junit] MAJOR: Summary:
[junit] MAJOR: -----
[junit] MAJOR: Analysis time: 1.5 seconds
[junit] MAJOR: Mutation score: 67.98% (67.98%)
[junit] MAJOR: Mutants killed / live: 138 (131-7-0) / 65
[junit] MAJOR: Mutant executions: 203
[junit] MAJOR: -----
[junit] MAJOR: Export summary of results to summary.csv
[junit] MAJOR: Export run-time results to results.csv
```

3. Result analysis

自此我们算是初步完成了一次变异体测试，但是这个杀死率不太理想（也太低了），说明我们的测试样例过于单调，对很多变异算子生成的变异体都未能检测出错误进行杀死，因此我们肯定是需要进一步增强我们的测试样例的。首先我们可以查看一下 `mutants.log` 中具体的生成变异体，一共有 203 个变异体生成如下所示是部分截图：

```
*mutants.log - 记事本
文件 编辑 查看

dedTriangle.UpgradedTriangle@triangleType(int,int,int):12:0 |==> 1
dedTriangle.UpgradedTriangle@triangleType(int,int,int):12:0 |==> -1
(int,int):UpgradedTriangle.UpgradedTriangle@triangl

eType(int,int,int):12:a <=
=(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:a <= 0 |==> a == 0
RUE(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:a <= 0 |==> true
dedTriangle.UpgradedTriangle@triangleType(int,int,int):12:0 |==> 1
dedTriangle.UpgradedTriangle@triangleType(int,int,int):12:0 |==> -1
(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:b <= 0 |==> b < 0
=(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:b <= 0 |==> b == 0
TRUE(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:b <= 0 |==> true
boolean)!= (boolean,boolean):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:a <= 0 || b <= 0 |==> a <= 0 != b <=
boolean):LHS(boolean,boolean):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:a <= 0 || b <= 0 |==> a <= 0
boolean):RHS(boolean,boolean):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:a <= 0 || b <= 0 |==> b <= 0
boolean):TRUE(boolean,boolean):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:a <= 0 || b <= 0 |==> true
radedTriangle.UpgradedTriangle@triangleType(int,int,int):12:0 |==> 1
radedTriangle.UpgradedTriangle@triangleType(int,int,int):12:0 |==> -1
<(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:c <= 0 |==> c < 0
=(int,int):UpgradedTriangle.UpgradedTriangle@triangleType(int,int,int):12:c <= 0 |==> c == 0

行 7, 列 85 100% Unix (LF) UTF-8
```

我们可以看到他罗列出了每一个变异体生成时使用的变异算子以及是如何具体修改代码进行变异的，因此我们为了能够进一步提高我们的测试样例，我们可以对比 `mutants.log` 中的变异体生成过程以及导出的 `killed.csv` 查看哪些生成的变异体未能杀死，未能杀死原因也可以一目了然：

```
killed [4].csv - 记事本
文件 编辑 查看

MutantNo,[FAIL | TIME | EXC | LIVE]
1,LIVE
2,LIVE
3,LIVE
4,LIVE
5,FAIL
6,LIVE
7,LIVE
8,LIVE
9,LIVE
10,FAIL
11,FAIL
12,LIVE
13,LIVE
14,FAIL
15,LIVE
16,LIVE
17,LIVE
18,LIVE
19,FAIL
20,FAIL

行 9, 列 7 100% Unix (LF) UTF-8
```

我们可以看到 1、2、3、4、6、7 等居然都没有杀死，当然有可能是因为这几个变异体都是等价变异体就不可能被样例杀死，但是更大概率应该是不等价的变异体未能被测试样例识别出错误从而存活了，因此我们查看一下我们的测试样例：

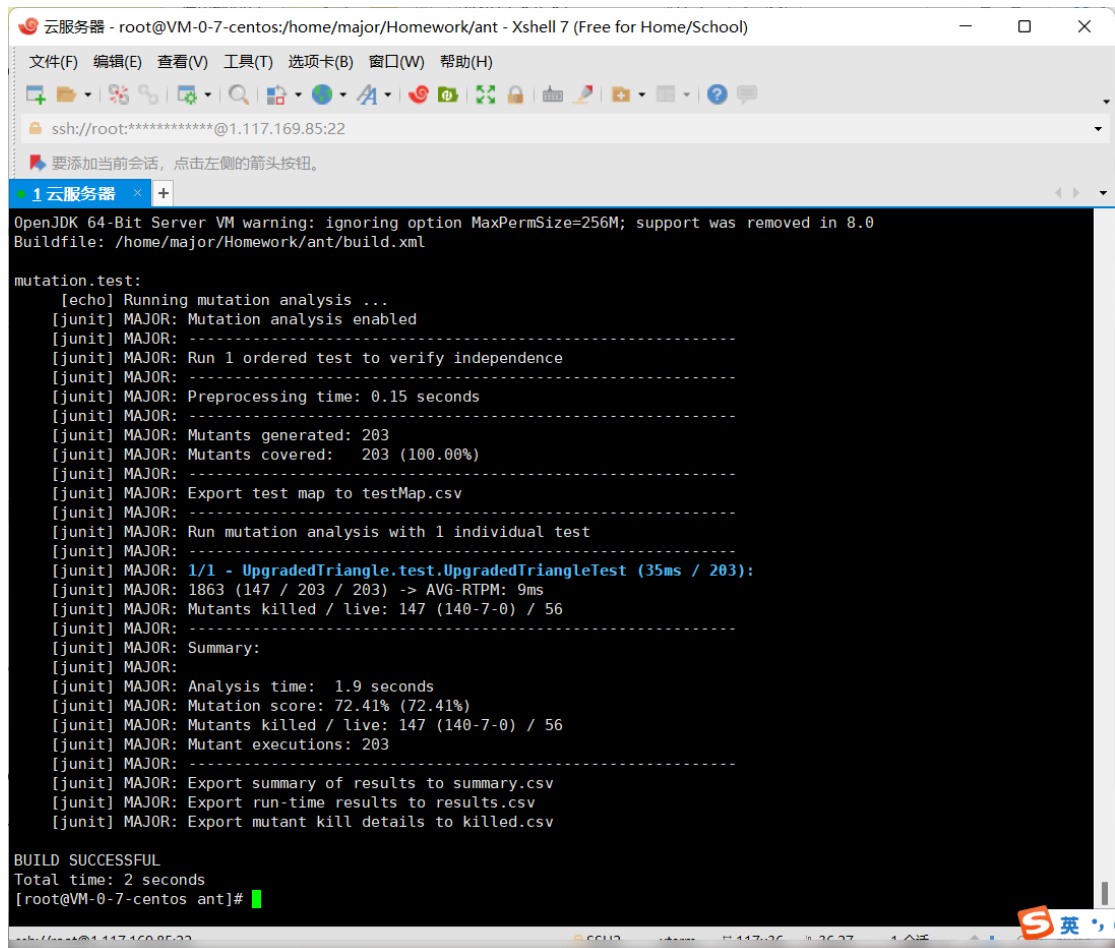
```
UpgradedTriangleTest [2].java - 记事本
文件 编辑 查看
public class UpgradedTriangleTest {

    @Test
    public void testTriangleType() {
        Map<int[], String> testMap = new HashMap<int[], String>() {
            {
                put(new int[]{-1, -1, 2}, "Invalid Input");
                put(new int[]{0, 0, 0}, "Invalid Input");
                put(new int[]{4, 4, 8}, "Not Triangle");
                put(new int[]{10, 10, 25}, "Not Triangle");
                put(new int[]{6, 6, 6}, "Equilateral Triangle");
                put(new int[]{10, 10, 10}, "Equilateral Triangle");
                put(new int[]{3, 4, 5}, "Right Angle Triangle");
                put(new int[]{5, 12, 13}, "Right Angle Triangle");
                put(new int[]{6, 6, 7}, "Other Triangle");
                put(new int[]{3, 4, 6}, "Other Triangle");
            }
        };
        for (int[] arr : testMap.keySet()) {
            String ans = UpgradedTriangle.triangleType(arr[0], arr[1], arr[2]);
        }
    }
}
```

我们分析以后会发现这个样例过于简单了因此，对于第一、二个变异体他们是将非法输入的判断表达式的临界值进行了修改改成了 $a \leq 1 || b \leq 1 || c \leq 1$ 以及 $a \leq -1 || b \leq -1 || c \leq -1$ ，对于前者确实无法使用测试样例杀死变异体，但是对于后者这种变异体很明显是不应该存活的，因为对于 $a=0, b=1, c=2$ 或者 $a=1, b=0, c=2$ 或者 $a=1, b=2, c=0$ 时应该是非法输入但是却并未能检测出来，因此我们可以借此增强我们的样例，加入新的样例测试如下所示：

```
@Test
public void testTriangleType() {
    Map<int[], String> testMap = new HashMap<int[], String>() {
        {
            put(new int[]{-1, -1, 2}, "Invalid Input");
            // 新增强化样例
            put(new int[]{0, 1, 2}, "Invalid Input");
            put(new int[]{1, 0, 2}, "Invalid Input");
            put(new int[]{1, 2, 0}, "Invalid Input");
            put(new int[]{0, 0, 0}, "Invalid Input");
            put(new int[]{4, 4, 8}, "Not Triangle");
            put(new int[]{10, 10, 25}, "Not Triangle");
            put(new int[]{6, 6, 6}, "Equilateral Triangle");
            put(new int[]{10, 10, 10}, "Equilateral Triangle");
            put(new int[]{3, 4, 5}, "Right Angle Triangle");
            put(new int[]{5, 12, 13}, "Right Angle Triangle");
            put(new int[]{6, 6, 7}, "Other Triangle");
            put(new int[]{3, 4, 6}, "Other Triangle");
        }
    };
    for (int[] arr : testMap.keySet()) {
        String ans = UpgradedTriangle.triangleType(arr[0], arr[1], arr[2]);
        assertEquals(testMap.get(arr), ans);
    }
}
```

然后我们再执行 `sh run.sh` 发现确实杀死率增强了，即将刚刚不合理存活的变异体进行了错误检测及时杀死了，也就增强了我们的测试样例，更加贴近测试需求：



```
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=256M; support was removed in 8.0
Buildfile: /home/major/Homework/ant/build.xml

mutation.test:
[echo] Running mutation analysis ...
[junit] MAJOR: Mutation analysis enabled
[junit] MAJOR: -----
[junit] MAJOR: Run 1 ordered test to verify independence
[junit] MAJOR: -----
[junit] MAJOR: Preprocessing time: 0.15 seconds
[junit] MAJOR: -----
[junit] MAJOR: Mutants generated: 203
[junit] MAJOR: Mutants covered: 203 (100.00%)
[junit] MAJOR: -----
[junit] MAJOR: Export test map to testMap.csv
[junit] MAJOR: -----
[junit] MAJOR: Run mutation analysis with 1 individual test
[junit] MAJOR: -----
[junit] MAJOR: 1/1 - UpgradedTriangle.test.UpgradedTriangleTest (35ms / 203):
[junit] MAJOR: 1863 (147 / 203 / 203) -> AVG-RTPM: 9ms
[junit] MAJOR: Mutants killed / live: 147 (140-7-0) / 56
[junit] MAJOR: -----
[junit] MAJOR: Summary:
[junit] MAJOR: -----
[junit] MAJOR: Analysis time: 1.9 seconds
[junit] MAJOR: Mutation score: 72.41% (72.41%)
[junit] MAJOR: Mutants killed / live: 147 (140-7-0) / 56
[junit] MAJOR: Mutant executions: 203
[junit] MAJOR: -----
[junit] MAJOR: Export summary of results to summary.csv
[junit] MAJOR: Export run-time results to results.csv
[junit] MAJOR: Export mutant kill details to killed.csv

BUILD SUCCESSFUL
Total time: 2 seconds
[root@VM-0-7-centos ant]#
```

类似的，如果我们先要进一步提高杀死率，增强我们的测试样例，只需要不断地重复上述步骤即可。这里我仅以上面的 72.41 杀死率的测试结果进行接下来的分析。首先我们前面说到了

问题 1：生成的变异体

生成的变异体数量为 203，这其中包括了对三角类型判断和三角面积求解两个方法的变异。

问题 2：变异体测试覆盖率

如上图所示给出的结果分析为 100%，所有的变异体都进行了测试。

问题 3：被测试样例杀死的变异体数量

如上面所示共杀死了 147 个变异体

问题 4： 存活的变异体数量

如上图所示共存活了 56 个变异体

问题 5： 计算一下 mutation score

我们参考可见给出的定义，他是杀死的变异体/存活的非等价变异体，但是我们知道这 56 个存活的变异体中肯定有一定数量的等价变异体永远不可能被杀死，因此我们只能给出一个下界：

至少当前情况下 得分 \geq 杀死的变异体/存活的所有变异体 $=147/56=2.625$ ：

问题 6： 测试的结果可以告诉我们哪些信息

通过 mutants.log 我们可以看到每一个变异体的生成过程，然后在结合上面的分析图，以及导出的 killed.csv,summary.csv 等测试结果，我们可以分析出每一个变异体在测试样例中的表现，以及现在测试样例题有哪些需要进行补充增强的方面，及时进行增强类似于我上面的演示步骤从而提升我们的测试样例对变异体的杀死率。

问题 7： 测试套件是否表现出弱点？ 如何改进？

测试样例总是不够完善的，我们需要进行多次操作和改进样例不断地尝试提升增强我们的样例。如上面步骤所示我就对我的测试样例进行了一次简单的改进从而增强了测试样例的杀死率，从 68->73。这个步骤主要是首先通过 mutants.log 查看每一个变异体生成的原因，然后再通过对比 killed.csv 查看哪些变异体没能够被杀死，会看我们的测试样例即可以分析出如何增强样例。但是我们一定要注意某些变异体是等价变异体无论如何也不可能被杀死的，我们只需要扩展测试样例的考察方面杀死哪些非等价的存活变异体即可。

问题 8： 测试套件是否表现出优势？ 如何发现的？

其实一个更加优秀的测试样例条件，应该可以尽可能考察多个方面，同时杀死多种的非等价变异体，我们可以先罗列出简单的多个单方面考察的测试样例，然后再结合测试的分析结果尝试进行具有相同点的测试套件进行融合即可生成更多的优秀测试，从而提升我们的测试效率。

问题 9： 在这些生成的变异体中，有没有一些变异算子可能产生的变异体集部分被杀死、部分存活？如果有这种变异算子可能是如何应用到代码中的呢？

其实有一个很暴力但是操作麻烦的办法我们可以找到这种变异算子。比如我们可以从存活的变异体入手，对比那些被杀死的变异体，看看是否有和存活变异体相同变异算子生成的变异体，如果有，那么这个变异算子肯定就是符合这种要求的了，而且我们可以通过生成的 `mutants.log` 的文件中就可以清晰的看出这种变异算子如何应用到代码中来生成变异体的。但是我再对比了一些变异算子以及上面自己进行增强的过程，发现有一种变异算子就可能导致上面这种情况的出现：

首先修改算术规则的变异算子一般可能不会产生上面这种效果，他修改了程序核心计算公式会很容易造成计算结果的错误。但是对于那些改变判断条件临界值的变异算子就可能生成题目描述的变异体集合，比如将 `<=` 判断条件改为了 `<` 的变异体就可能继续通过测试，但是变异成 `>` 或者 `!=` 的变异体就可能会被杀死。

4. Source code

实验程序已存储到 github: <https://github.com/Langwenchong/SoftwareTestingLabs/tree/lab4>

1. UpgradedTriangle.java

```
package UpgradedTriangle;

import java.util.Arrays;

/**
 * @Author Lang wenchong
 * @Date 2022/3/21 16:43
 * @Version 1.0
 */
public class UpgradedTriangle {
    public static String triangleType(int a, int b, int c) {
        if (a <= 0 || b <= 0 || c <= 0) {
            return "Invalid Input";
        } else if (a + b > c && a + c > b && b + c > a) {
            // 是三角形
            if (a == b && a == c) {
                // 等边三角形
                return "Equilateral Triangle";
            }

            int[] edges = {a, b, c};
            Arrays.sort(edges);
```

```

        if (edges[0] * edges[0] + edges[1] * edges[1] == edges[2] * edges[2]) {
//            直角三角形
            return "Right Angle Triangle";
        }

        return "Other Triangle";
    } else {
        return "Not Triangle";
    }
}

public static double areaOfTriangle(double a, double b, double c) {
    if (a <= 0 || b <= 0 || c <= 0) {
        return 0;
    } else if (a + b > c && a + c > b && b + c > a) {
//        是三角形
        double p = (a + b + c) / 2;
        return ((int)Math.sqrt(p * (p - a) * (p - b) * (p - c)*10000))/100.0;
    } else {
        return 0;
    }
}
}

```

2. UpgradedTriangleTest.java

```

package UpgradedTriangle.test;

import UpgradedTriangle.UpgradedTriangle;
import org.junit.Test;

import java.util.HashMap;
import java.util.Map;

import static org.junit.Assert.assertEquals;

/**
 * @Author Lang wenchong
 * @Date 2022/3/21 17:00
 * @Version 1.0
 */
public class UpgradedTriangleTest {

    @Test
    public void testTriangleType() {
        Map<int[], String> testMap = new HashMap<int[], String>() {
            {
                put(new int[]{-1, -1, 2}, "Invalid Input");
//                新增强化样例
            }
        };
    }
}

```

```

        put(new int[] {0, 1, 2}, "Invalid Input");
        put(new int[] {1, 0, 2}, "Invalid Input");
        put(new int[] {1, 2, 0}, "Invalid Input");
        put(new int[] {0, 0, 0}, "Invalid Input");
        put(new int[] {4, 4, 8}, "Not Triangle");
        put(new int[] {10, 10, 25}, "Not Triangle");
        put(new int[] {6, 6, 6}, "Equilateral Triangle");
        put(new int[] {10, 10, 10}, "Equilateral Triangle");
        put(new int[] {3, 4, 5}, "Right Angle Triangle");
        put(new int[] {5, 12, 13}, "Right Angle Triangle");
        put(new int[] {6, 6, 7}, "Other Triangle");
        put(new int[] {3, 4, 6}, "Other Triangle");
    }

};

for (int[] arr : testMap.keySet()) {
    String ans = UpgradedTriangle.triangleType(arr[0], arr[1], arr[2]);
    assertEquals(testMap.get(arr), ans);
}

}

@Test
public void testAreaOfTriangle() {
    Map<int[], Double> testMap = new HashMap<int[], Double>() {
        {
            put(new int[] {-1, -1, 2}, 0.00);
            // 新增强化样例
            put(new int[] {0, 1, 2}, 0.00);
            put(new int[] {1, 0, 2}, 0.00);
            put(new int[] {1, 0, 2}, 0.00);
            put(new int[] {0, 0, 0}, 0.00);
            put(new int[] {4, 4, 8}, 0.00);
            put(new int[] {10, 10, 25}, 0.00);
            put(new int[] {6, 6, 6}, 15.58);
            put(new int[] {10, 10, 10}, 43.30);
            put(new int[] {3, 4, 5}, 6.00);
            put(new int[] {5, 12, 13}, 30.00);
            put(new int[] {6, 6, 7}, 17.05);
            put(new int[] {3, 4, 6}, 5.33);
        }
    };

    for (int[] arr : testMap.keySet()) {
        Double ans = UpgradedTriangle.areaOfTriangle(arr[0], arr[1], arr[2]);
        assertEquals(testMap.get(arr), ans);
    }
}

```



```
}
```

3. Build.xml

```
<project name="homework" default="compile" basedir=". ">
    <target name="init">
        <mkdir dir="bin"/>
    </target>
    <!-- Target to compile the project -->
    <target name="compile" depends="init" description="Compile">
        <javac includeantruntime="true"
            srcdir="src"
            destdir="bin"
            debug="yes"
            fork="yes"
            executable="${major}">
            <compilerarg value="${mutator}"/>
        </javac>
    </target>

    <!-- Target to compile the test suite -->
    <target name="compile.tests" depends="compile" description="Compile all
tests">
        <javac includeantruntime="true"
            srcdir="test"
            destdir="bin"
            debug="yes">
        </javac>
    </target>

    <!-- The adapted mutation test target -->
    <target name="mutation.test" description="Run mutation analysis for all unit test
cases">
        <echo message="Running mutation analysis ..."/>
        <junit printsummary="false"
            showoutput="false"
            mutationAnalysis="true"
            resultFile="results.csv"
            killDetailsFile="killed.csv"
            mutantsLogFile="mutants.log">

            <classpath path="bin"/>
            <batchtest fork="false">
                <fileset dir="test">
                    <include name="**/*Test*.java"/>
                </fileset>
            </batchtest>
        </junit>
    </target>
</project>
```

4. Run.sh

```
#!/bin/sh

MAJOR_HOME="./.."

echo
echo "Compiling and mutating project"
echo "(ant -DmutOp=\"=\$MAJOR_HOME/mml/tutorial.mml.bin\" clean compile)"
echo
$MAJOR_HOME/bin/ant -DmutOp="=\$MAJOR_HOME/mml/tutorial.mml.bin"
clean compile

echo
echo "Compiling tests"
echo "(ant compile.tests)"
echo
$MAJOR_HOME/bin/ant compile.tests

echo
echo "Run tests without mutation analysis"
echo "(ant test)"
$MAJOR_HOME/bin/ant test

echo
echo "Run tests with mutation analysis"
echo "(ant mutation.test)"
$MAJOR_HOME/bin/ant mutation.test
```

5. 实验心得总结

实验四耗时较长，主要是自己阅读英文文档有很多细节地方未能及时掌握导致踩了许多坑，但是回过来看付出是有回报的，经过多日的摸索，我对 **major mutants** 测试工具的使用以及原理有了初步的了解，同时也通过实验三和实验四亲自使用了不同的工具进行了变异体测试，所谓实践出真知，只有亲自动手实验以后才能真的理解透一个知识点，相信经过本次课程可以对软件测试有了更加深刻的认识！