哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2018 年春季学期

计算机学院大二软件构造课程

Lab 4 实验报告

姓名	朱明彦
学号	1160300314
班号	1603003
电子邮件	1160300314@stu.hit.edu.cn
手机号码	18846082306

目录

1 实验目标概述

本次实验重点训练学生面向健壮性和正确性编程的技能，利用错误和异常处理、断言与防御式编程的技术、日志/断点等调试技术、黑盒测试编程技术，使程序可在不同健壮性/正确性需求下能恰当的处理各种例外与错误情况，在出错后可优雅的退出或继续执行，发现错误之后可有效的定位错误并作出修改。

主要需要使用以下技术进行改造，提高其健壮性和正确性：

- 错误处理
- 异常处理
- Assertion 和防御式编程
- 日志
- 调试技术
- 黑盒测试及代码覆盖度

2 实验环境配置

实验环境在之前的实验中对于 IDE 和 Java、git 环境都已经配置完毕。

在这里给出你的 GitHub Lab4 仓库的 URL 地址（Lab4-学号）。

https://github.com/ComputerScienceHIT/lab4-1160300314

3 实验过程

3.1 Error and Exception Handling

3.1.1 针对输入文本文件的异常/错误处理

对于输入文本错误的处理，主要有两种处理方案，分别是让用户重新选择输入文件或者是发现错误后进行简单处理但是程序继续运行，我将分开阐述对于不同的这两个方面的处理方法。

3.1.1.1 InputFileAgainException

对于这个自定义的 Exception 类主要用于处理所有需要重新输入文件的异常情况，下面将分别指出这些异常情况。

（1）文件中出现了不合乎语法的输入指令，如“Vertex = <“Word1”, “Word”>”这样缺少相应的“>”，违法了语法规则，做出的处理是，在 ParseCommandHelper 中直接抛出 IllegalGrammarTextException，在使用其解析输入文件的 GraphFactory 中同样不做处理直接抛出，并标明所发生的错误为语法错误。

测试的相关截图为：

```
[DEBUG] 2018-05-18 20:14:34,045 method:Main.main(Main.java:19)
1 :Input file has grammar errors Vertex = <"Word1", "Word"
exception.InputFileAgainException: 1 :Input file has grammar errors Vertex = <"Word1", "Word"
  at factory.GraphFactory.createGraph(GraphFactory.java:204)
  at Main.main(Main.java:17)
```

（2）文件在边中使用了未定义的顶点，同样属于需要退出重新选择文件的错误，在 GraphFactory 中直接抛出该异常，测试相关的截图如下：

测试使用的文件如下：

```
GraphType = "GraphPoet"
GraphName = "PoetGraph"

VertexType = "Word"
Vertex = <"Word1", "Word">

EdgeType = "WordNeighborhood"
Edge = <"edge", "WordNeighborhood", "1", "Word1", "Word2", "Yes">
```

可以看到其中 Edge 中使用了未定义的顶点“Word2”，所以测试的时候回抛出 UndefinedVertexException 表示出现了未定义的顶点，相关的测试的日志信息如下图所示：

```
[DEBUG] 2018-05-18 20:18:25,487 method:Main.main(Main.java:19)
Word2 is undefined!
exception.UndefinedVertexException: Word2 is undefined!
  at factory.GraphFactory.createGraph(GraphFactory.java:138)
  at Main.main(Main.java:17)
```

（3）对于不符合节点中属性定义的错误，例如在 Movie 中缺少部分属性值等情况，由于不能继续进行，所以仍然选择使用抛出异常重新选择文件输入。

相关的测试文件如下：

```
GraphType = "MovieGraph"
GraphName = "MyFavoriteMovies"

VertexType = "Movie", "Actor", "Director"
Vertex = <"TheShawshanRedemption", "Movie", <"1994", "USA">>
```

可以看到在输入顶点 Movie 时缺少相应的 IMDB 评分属性，所以直接抛出相关异常。测试结果截图如下：

```
[DEBUG] 2018-05-18 20:23:50,787 method:Main.main(Main.java:19)
Fill Movie vertex with 2 param(s)
exception.IllegalParamsNumberException: Fill Movie vertex with 2 param(s)
  at vertex.Movie.fillVertexInfo(Movie.java:94)
  at factory.VertexFactory.createVertex(VertexFactory.java:75)
  at factory.GraphFactory.createGraph(GraphFactory.java:100)
  at Main.main(Main.java:17)
```

（4）如果在某种节点中输入了不应该引入的顶点类型，比如在 GraphPoet 中引入了 Person 类型的数据，则会在测试中 GraphFactory 直接抛出 IllegalVertexTypeException，表示在这个图中不应该出现这种类型的顶点。相关的测试文件如下：

```
GraphType = "GraphPoet"
GraphName = "PoetGraph"

VertexType = "Word"
Vertex = <"Word1", "Person", <"M", "19">>
Vertex = <"Word2", "Word">
Vertex = <"Word3", "Word">
Vertex = <"Word4", "Word">
```

可以看到在 GraphPoet 中加入了 Person 类型的数据，那么会直接提示相关的错误，如下：

```
[DEBUG] 2018-05-18 20:33:23,649 method:Main.main(Main.java:19)
Person is an illegal vertex type in GraphPoet
exception.IllegalVertexTypeException: Person is an illegal vertex type in GraphPoet
  at factory.GraphFactory.createGraph(GraphFactory.java:95)
  at Main.main(Main.java:17)
```

在这个问题中有一个类似处理的问题就是，如果我使用了不对的顶点类型声明，尝试将 Person 类的数据加入 GraphPoet 中，同样会报告相关的错误。相关的测试数据如下：

```
GraphType = "GraphPoet"
GraphName = "PoetGraph"

VertexType = "Word", "Person"
Vertex = <"Word1", "Person", <"M", "19">>
Vertex = <"Word2", "Word">
Vertex = <"Word3", "Word">
Vertex = <"Word4", "Word">
```

可以看到我们尝试在 VertexType 中增加别的顶点类型的定义，比如我们后边想要用到的 Person，测试的结果如下：

```
[DEBUG] 2018-05-18 20:38:53,193 method:Main.main(Main.java:19)
[Word, Person] are not defined in GraphPoet!
exception.IllegalVertexTypeException: [Word, Person] are not defined in GraphPoet!
  at factory.GraphFactory.legalVertexType(GraphFactory.java:247)
  at factory.GraphFactory.createGraph(GraphFactory.java:87)
  at Main.main(Main.java:17)
```

表示 Word 和 Person 不能是同时定义在 GraphPoet 中。

（5）在某种类型的图中引入了不应出现的边的类型，如在 GraphPoet 中出现了 FriendTie 的边，相关的处理仍然和（4）相同，具体的测试见测试输入文件和测试结果：

```
GraphType = "GraphPoet"
GraphName = "PoetGraph"

VertexType = "Word"
Vertex = <"Word1", "Word">
Vertex = <"Word2", "Word">
Vertex = <"Word3", "Word">
Vertex = <"Word4", "Word">

EdgeType = "WordNeighborhood"
Edge = <"edge", "FriendTie", "1", "Word1", "Word2", "Yes">
```

可以看到在 Edge 中增加了 FriendTie 类型的边，但是在 GraphPoet 中这样的边是不合法的，相关的测试打印出的信息如下，

```
[DEBUG] 2018-05-18 20:46:00,460 method:Main.main(Main.java:19)
FriendTie is an illegal edge type!
exception.IllegalEdgeTypeException: FriendTie is an illegal edge type!
  at factory.GraphFactory.createGraph(GraphFactory.java:117)
  at Main.main(Main.java:17)
```

同样如果尝试在 EdgeType 中增加相关非法的边的类型的定义，仍然会与（4）一样报告相关的错误。

（6）在超边中包含的节点个数小于两个，由于在 MovieGraph 中使用的超边表示的是出演过同一部电影的演员的名单，所以在这里当顶点的个数少于两个的时候，将抛出 LackVerticesHyperEdgeException 异常重新选择文件。

测试文件如下：

```
GraphType = "MovieGraph"
GraphName = "MyFavoriteMovies"

VertexType = "Movie", "Actor", "Director"
Vertex = <"TheShawshanRedemption", "Movie", <"1994", "USA", "9.3">>
Vertex = <"FrankDarabont", "Director", <"M", "59">>
Vertex = <"MorganFreeman", "Actor", <"M", "81">>
Vertex = <"TimRobbins", "Actor", <"M", "60">>
Vertex = <"TheGreenMile", "Movie", <"1999", "USA", "8.5">>
Vertex = <"TomHanks", "Actor", <"M", "62">>

EdgeType = "MovieActorRelation", "MovieDirectorRelation", "SameMovieHyperEdge"
HyperEdge = <"ActorInSR", "SameMovieHyperEdge", {"TimRobbins"}>
```

仅有一个顶点

测试输出的结果将表示存在错误，具体如下：

```
[DEBUG] 2018-05-18 21:10:45,084 method:Main.main(Main.java:19)
HyperEdge only contains one vertex!
exception.LackVerticesHyperEdgeException: HyperEdge only contains one vertex!
  at factory.GraphFactory.createGraph(GraphFactory.java:194)
  at Main.main(Main.java:17)
```

（7）在带边权重未能给出权重，例如在 WordNeighborhood 中将边的权值赋为-1，表示该边没有权值，此处将会在 WordNeighborhood 中的抛出边权异常 IllegalEdgeParamsException，需要重新输入文件。

测试文件如下：

```
GraphType = "GraphPoet"
GraphName = "PoetGraph"

VertexType = "Word"
Vertex = <"Word1", "Word">
Vertex = <"Word2", "Word">
Vertex = <"Word3", "Word">
Vertex = <"Word4", "Word">

EdgeType = "WordNeighborhood"
Edge = <"edge", "WordNeighborhood", "-1", "Word1", "Word2", "Yes">
```

测试的结果如下：

```
[DEBUG] 2018-05-18 21:14:55,212 method:Main.main(Main.java:19)
Weight of WordNeighborhood is less than 0
exception.IllegalEdgeParamsException: Weight of WordNeighborhood is less than 0
  at edge.WordNeighborhood.<init>(WordNeighborhood.java:32)
  at factory.EdgeFactory.createEdge(EdgeFactory.java:39)
  at factory.GraphFactory.createGraph(GraphFactory.java:162)
  at Main.main(Main.java:17)
```

（8）类似于（7）在边中输入不符合要求的权值，例如在 SocialNetwork 中输入边权超过 1 的边等等，都违反了输入要求，需要重新选择输入的文件。

测试使用的文件如下：

```
GraphType = "SocialNetwork"
GraphName = "SOCIALNETWORK"

VertexType = "Person"
Vertex = <"Kevin", "Person", <"M", "19">>
Vertex = <"Amy", "Person", <"F", "18">>

EdgeType = "CommentTie", "FriendTie", "ForwardTie"
Edge = <"forwardTie", "ForwardTie", "2", "Kevin", "Amy", "Yes">
```

边权超过了1

测试的结果如下图：

```
[DEBUG] 2018-05-18 21:21:02,640 method:Main.main(Main.java:19)
Weight of Forward Connection is illegal
exception.IllegalEdgeParamsException: Weight of Forward Connection is illegal
  at edge.ForwardTie.<init>(ForwardTie.java:30)
  at factory.EdgeFactory.createEdge(EdgeFactory.java:49)
  at factory.GraphFactory.createGraph(GraphFactory.java:162)
  at Main.main(Main.java:17)
```

（9）对于边或者顶点的或者图的 label 的命名不符合[\w]+的命名规范，将抛出错误并要求用户重新选择输入文件。例如将 GraphName = “@..”这种格式在测试的时候就会提示这种异常。

```
[DEBUG] 2018-05-18 21:24:26,911 method:Main.main(Main.java:19)
2  :Input file has grammar errors GraphName = "@SOCIALNETWORK"
exception.InputFileAgainException: 2  :Input file has grammar errors GraphName = "@SOCIALNETWORK"
  at factory.GraphFactory.createGraph(GraphFactory.java:208)
  at Main.main(Main.java:17)
```

（10）对于在实验手册中要求 loop 的情况，其实这仍然属于一种输入的问题，并且出现了这种错误其实是非常严重的错误，所以在我的处理中选择将这种错误**作为一种需要重新输入文件的异常进行处理**，而不是仅仅将其作为一种可以继续执行的异常。

具体的测试文件如下：

```
GraphType = "SocialNetwork"
GraphName = "SOCIALNETWORK"

VertexType = "Person"
Vertex = <"Kevin", "Person", <"M", "19">>
Vertex = <"Amy", "Person", <"F", "18">>

EdgeType = "CommentTie", "FriendTie", "ForwardTie"
Edge = <"forwardTie", "ForwardTie", "1", "Kevin", "Kevin", "Yes">
```

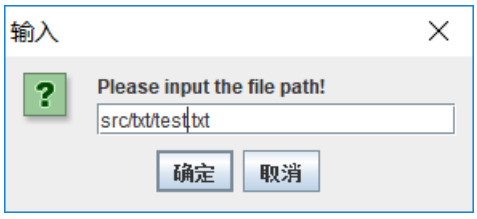
测试的结果如下：

```
[DEBUG] 2018-05-19 08:55:59,332 method:Main.main(Main.java:19)
ForwardTie can not be loop!
exception.IllegalEdgeParamsException: ForwardTie can not be loop!
  at edge.ForwardTie.addVertices(ForwardTie.java:57)
  at factory.EdgeFactory.createEdge(EdgeFactory.java:56)
  at factory.GraphFactory.createGraph(GraphFactory.java:165)
  at Main.main(Main.java:17)
```

并且如果是仅仅将输入文件中边的选项只选择一个顶点作为 loop 输入，则会有类似的错误，如下：

```
[DEBUG] 2018-05-19 09:02:09,439 method:Main.main(Main.java:19)
9  :Input file has grammar errors Edge = <"forwardTie", "ForwardTie", "1", "Kevin", "Yes">
exception.InputFileAgainException: 9  :Input file has grammar errors Edge = <"forwardTie", "ForwardTie", "1", "Kevin", "Yes">
  at factory.GraphFactory.createGraph(GraphFactory.java:208)
  at Main.main(Main.java:17)
```

对于所有以上提到的需要重新输入文件进行处理的异常，在 App 的测试时会表现为下面描述的情况：



首先输入文件，然后点击确定按键；

```
[DEBUG] 2018-05-19 09:07:23,286 method:application.GraphPoetApp.<init>(GraphPoetApp.java:40)
Read file from "src/txt/test.txt"
[ERROR] 2018-05-19 09:07:23,680 method:application.GraphPoetApp.<init>(GraphPoetApp.java:62)
11  :Input file has grammar errors Edge = <"edge", "WordNeighborhood", "1", "Word1", "Word2", "Yes"
exception.InputFileAgainException: 11  :Input file has grammar errors Edge = <"edge", "WordNeighborhood", "1", "Word1", "Word2", "Yes"
  at factory.GraphFactory.createGraph(GraphFactory.java:208)
  at application.GraphPoetApp.<init>(GraphPoetApp.java:44)
  at application.GraphPoetApp.main(GraphPoetApp.java:103)
```

在日志和 Terminal 中都会打印出类似于上面的错误信息，并重新弹出需要输入文件的对话框，要求用户重新输入文件（或者将文件修复后重新读入）。

如果输入正确就会得到相关的 INFO，否则会退出程序。

```
[INFO ] 2018-05-19 09:09:47,152 method:application.GraphPoetApp.<init>(GraphPoetApp.java:71)
Exchange file path to src/txt/inputGraphPoet.txt
[DEBUG] 2018-05-19 09:09:47,155 method:application.GraphPoetApp.<init>(GraphPoetApp.java:40)
Read file from "src/txt/inputGraphPoet.txt"
```

3.1.1.2 ContinueRunningException

对于这类自定义的异常，主要用于处理一些不是非常严重的错误，主要的处理方式就是将异常信息抛出，并将其处理的方式写出，然后不再进行其他的处理。

（1）单重图中存在了多重边，在处理的时候，仅仅将除了第一条边的其余边删除并记录相关的删除信息，然后继续向下执行。

相关的测试文件如下所示：

```
EdgeType = "WordNeighborhood"
Edge = <"edge", "WordNeighborhood", "1", "Word1", "Word2", "Yes">
Edge = <"edge1", "WordNeighborhood", "10", "Word1", "Word2", "Yes">
```

在 Word1 和 Word2 中出现了多重边，所以在测试得到的信息中会表现为：

```
[ERROR] 2018-05-19 09:15:44,584 method:factory.GraphFactory.createGraph(GraphFactory.java:177)
edge1 has been multi defined!
exception.ContinueRunningException: edge1 has been multi defined!
  at factory.GraphFactory.createGraph(GraphFactory.java:159)
  at Main.main(Main.java:17)
```

（2）如果在相关的 label 之间出现了重复，则选择在重复的边后增加序号，作为一种提示然后继续向下进行。

相关的测试文件如下：

```
GraphType = "GraphPoet"
GraphName = "PoetGraph"

VertexType = "Word"
Vertex = <"Word", "Word">
Vertex = <"Word", "Word">
Vertex = <"Word", "Word">
Vertex = <"Word", "Word">
```

在测试中，除了第一个“Word”的顶点，其余顶点的均属于重复定义的 label，所以在测试的时候回输出相关的测试信息如下：

```
[ERROR] 2018-05-19 09:20:46,400 method:factory.GraphFactory.modifyDuplicateLabels(GraphFactory.java:289)
Word has been used!
exception.DuplicateLabelsException: Word has been used!
  at factory.GraphFactory.modifyDuplicateLabels(GraphFactory.java:286)
  at factory.GraphFactory.createGraph(GraphFactory.java:100)
  at Main.main(Main.java:17)
[INFO ] 2018-05-19 09:20:46,414 method:factory.GraphFactory.modifyDuplicateLabels(GraphFactory.java:291)
We change it to Word001
```

（3）对于其余的一些在实验手册中标明的异常。如**在不应出现超边的图中出现了超边，这属于一种使用了不合法类型的边，在上面的需要用户重新输入文件的异常中已经做出了相应的处理。**

3.1.2 针对输入图操作指令的异常/错误处理（可选）

覆盖实验手册 3.1 节中(2)列出的各项任务。

3.2 Assertion and Defensive Programming

3.2.1 checkRep()检查 invariants

利用 checkRep 检查表示不变量，是一种比较常用的手段。在 debug 的阶段，将 checkRep 作为一种保障正确性的方式。主要的检查出现在 edge、vertex 和 Graph 类及其子类中，具体的实现类似，分别举一个例子进行说明。

3.2.1.1 vertex 类及其子类的 checkRep

此处的实现是基于 vertex 抽象类的 checkRep，在最基础的 vertex 的抽象类仅仅检查 vertex 的 label 是否符合要求（不能为 null 或者由[^\w]的字符组成），继承 vertex 抽象类的 Person 则将 checkRep 表现为：

```
@Override
protected void checkRep() throws IllegalVertexParamsException {
  super.checkRep();
  assert gender != null && (gender.equals("M") || gender.equals("F"));
  if (gender == null || (!gender.equals("M") && !gender.equals("F")) || age <= 0) {
    throw new IllegalVertexParamsException("Params of Person is wrong!");
  }
  //    Gender of Person instance is 'M' or 'F'.
  assert age > 0;
}
```

即检查性别（Gender）和年龄（age）是否符合表示不变量。

3.2.1.2 edge 类及其子类的 checkRep

在 edge 的抽象类中，checkRep 仅仅检查 label 是不是符合要求以及 weight 是否为正数或者-1（表示无权边），对于更加细节的 checkRep 则由相应的继承 edge 的子类中覆写函数实现，如下面在 WordNeighborhood 中仅需要检查是否为权值为正：

```
@Override
protected void checkRep() throws InputFileAgainException {
  assert this.getWeight() > 0;
  super.checkRep();
}
```

3.2.1.3 Graph 类及其子类的 checkRep

在 ConcreteGraph 中可以检查的不变量很少，仅仅可以检查是否为合法的 label 表示。下面以 NetworkTopology 为例：

```
@Override
protected void checkRep() {
  super.checkRep();
  Set<NetworkConnection> edgeSet = this.edges();
  for (Edge edge : edgeSet) {
    List<Vertex> source = new ArrayList<>(edge.sourceVertices());
    assert source.size() == 2;
    List<Vertex> target = new ArrayList<>(edge.targetVertices());
    assert target.size() == 2;
    assert !source.get(0).equals(target.get(1));
  }
}
```

测试所有的边均是符合要求的无向无 loop 边。

3.2.2 Assertion 保障 pre-/post-condition

对于实验手册上的要求需要利用 assert 来保证 Pre/Post-condition，由于本身在程序员中间就存在分歧，我是比较赞同于不使用 assert 来作为保障 Pre/Post-condition 的，这样的缺点在于我仅仅能在 debug 阶段保证我所有的函数中的前置和后置条件满足，但真正在 release 版本中并不能保障函数的正确性，所以在这一个方面我**更加倾向于使用 Exception 作为一种保证函数正确性和健壮性的方式**。

基于这种考虑，我在大部分违背要求的函数中增加了 throws 作为一种保障机制。下面将分别说一下其中的一些处理。

（1）在 Vertex 类中将 label 的格式是否符合作为 pre-condition，在具体实现中表现为：

```
public Word(String label) throws IllegalVertexParamsException {
  super(label);
  super.checkRep();
}
```

在此处没有直接抛出异常，而是作为 super.checkRep 中的测试可以抛出，具体在 checkRep 中的表现如下：

```
protected void checkRep() throws IllegalVertexParamsException {
  if (label == null) {
    throw new IllegalVertexParamsException("Label is null!");
  }
  assert label != null;
  assert label.length() != 0;
  Pattern pattern = Pattern.compile("^[\\w]+$");
  // All label consists of [A-Za-z_0-9]
  Matcher matcher = pattern.matcher(label);
  assert matcher.matches();
  if (label.length() == 0 || !matcher.matches()) {
    throw new IllegalVertexParamsException("Label is illegal!");
  }
}
```

（2）在 edge 类特别是其子类中，对于 addVertices 函数需要保证所有加入的顶点都是符合边的类型定义并且不能有违背其定义的（如不能有 loop）

下面以 ForwardTie 中的 addVertices 为例说明。

```
public boolean addVertices(List<Vertex> vertices) throws InputFileAgainException {
  if (vertices == null) {
    throw new IllegalVertexParamsException("Add vertices into ForwardTie with null
vertices!");
  }
  if (vertices.size() != 2) {
    throw new IllegalVertexParamsException(
        "Add vertices into ForwardTie with " + vertices.size() + " vertices");
  }
  assert vertices.size() == 2;
  if (vertices.get(0).equals(vertices.get(1))) {
    throw new IllegalEdgeParamsException("ForwardTie can not be loop!");
  }
  boolean answer = true;
  for (Vertex vertex : vertices) {
    answer = answer && vertex instanceof Person;
    if (!answer) {
      throw new IllegalVertexTypeException(
          "Add " + (vertex == null ? "null" : vertex.getClass().getSimpleName())
              + " vertex into ForwardTie!");
    }
  }
  return answer && super.addVertices(vertices);
}
```

在加入 Vertices 中如果其为 null 或者由不是两个顶点组成的 list 作为参数，都是会抛出相关的异常。并且由于 ForwardTie 是不允许有 loop（所有的在 SocialNetwork 中的边均不允许），所以如果其中出现 loop 也是会抛出相关的异常。

（3）在相关的 Graph 中的一些不变量被违反，比如在 Graph 中增加 null 类型的顶点，这是违反了其中对于参数的要求，所以会抛出相关的异常。

```
public boolean addVertex(L v) {
  //       LOGGER.debug("Attempt to add Vertex " + v.getLabel());
  if (v == null) {
    throw new RuntimeException("Can not add null into vertex set in Graph!");
  }
  return vertexSet.add(v);
}
```

（4）另外一些防御式的编程的方式，没有使用 assert 或者是 throws Exception 的方式。

在第三章的时候我们讲过防御式拷贝的内容，对于可变的数据类型，所有需要返回或者作为参数传入需要使用的地方均使用了防御式拷贝的方法，防止调用者可以利用 reference 将内部的值做改变，相关的一些例子：

```
public Set<L> vertices() {
  return new HashSet<>(vertexSet);
}

//截取自 GraphFactory 中的片段，在所有的 edge 中增加的所有 vertex 都是复制得到的
for (Vertex vertex : vertexSet) {
  if (vertex.getLabel().equals(list.get(4))) {
    vertex1 = vertex.clone();
  }
  if (vertex.getLabel().equals(list.get(5))) {
    vertex2 = vertex.clone();
  }
}
```

另外还有一些对于 mutable 的数据类型，vertex 和 edge 在与 Mr.Wang 交流的时候，如果我直接将 mutable 类型的数据作为 HashSet 中的元素，并且覆写了 hashCode 方法，那么**每次修改 vertex 或者是 edge 的内容就会导致原本存在于集合中的元素，由于更改了 hash 值变为不存在于集合中。**为了避免这种情况，我没有给任何 vertex、edge 提供 setter 方法，并且所有的修改处都是做不了，如果尝试修改任意一个顶点，均需要将原有的顶点信息进行拷贝，在新的顶点对象中修改。

3.3 Logging

3.3.1 写日志

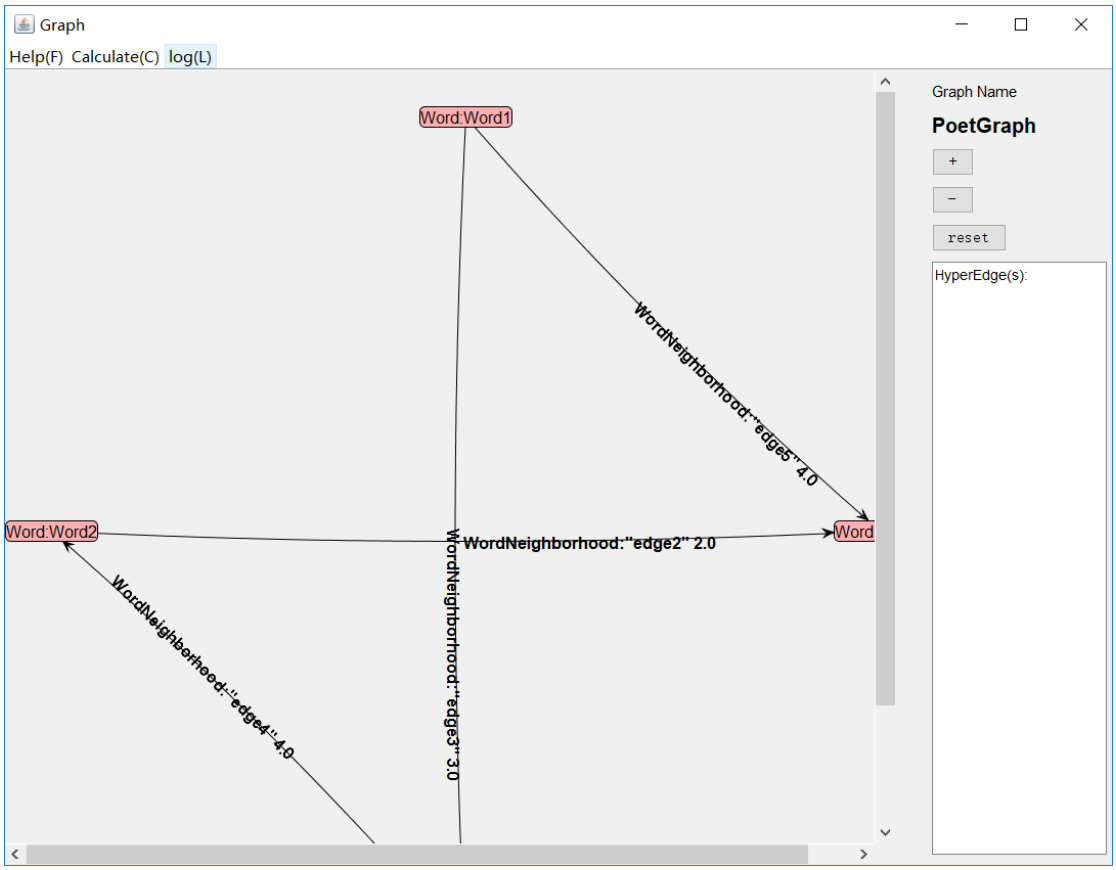
在这里我没有使用 Logging（JDK 自带的日志工具），而是选择使用 Log4j 作为主要的日志记录工具，相关的配置文件摘要如下：

```
log4j.rootLogger=debug,stdout,D,E
### 输出信息到控制台 ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS}
method:%l%n%m%n
### 输出 DEBUG 级别以上的日志到文件./log/debug.log ###
log4j.appender.D=org.apache.log4j.FileAppender
log4j.appender.D.File=./log/debug.log
log4j.appender.D.Append=true
log4j.appender.D.Threshold=DEBUG
log4j.appender.D.layout=org.apache.log4j.PatternLayout
log4j.appender.D.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}  [ %c ] - [ %p ] -
[ %l ] %m%n
### 输出 ERROR 级别以上的日志到文件./log/error.log ###
log4j.appender.E=org.apache.log4j.FileAppender
log4j.appender.E.File=./log/error.log
log4j.appender.E.Append=true
log4j.appender.E.Threshold=ERROR
log4j.appender.E.layout=org.apache.log4j.PatternLayout
log4j.appender.E.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}  [ %c ] - [ %p ] -
[ %l ] %m%n
```

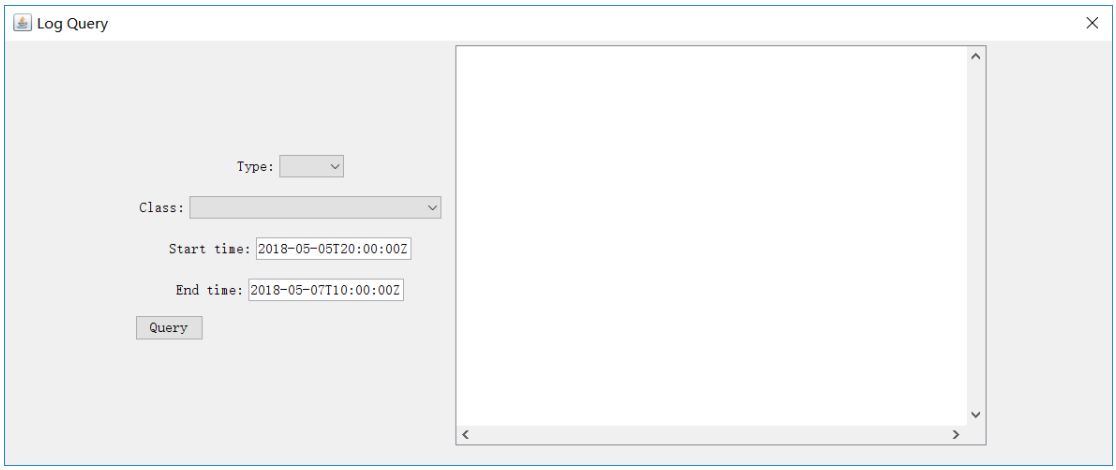
此处进行一下说明，**所有的 log 文件均存在于 log 文件夹下**。对于 log 的级别，所有的异常抛出均以[ERROR]进行记录；相关的异常处理以[INFO]级别进行记录；相应的读取操作信息以[DEBUG]级别进行记录。

3.3.2 日志查询

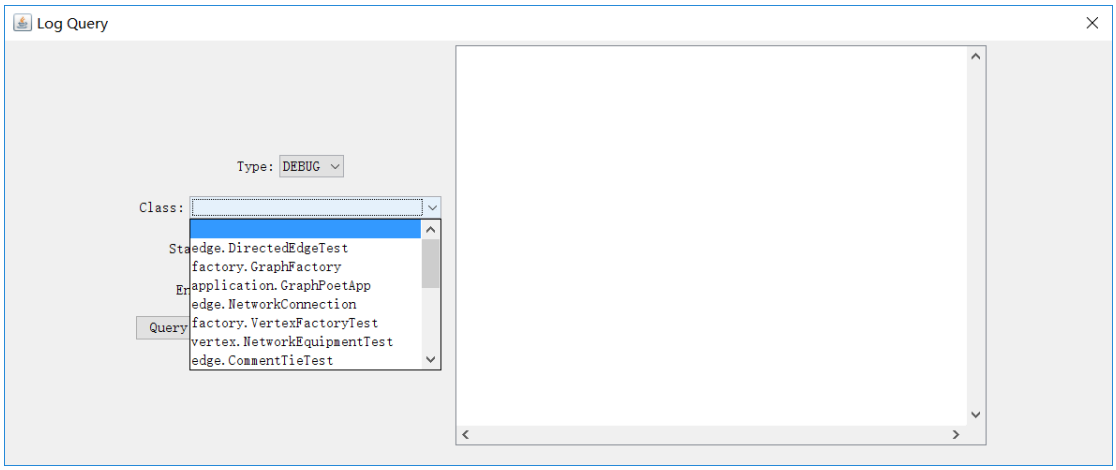
日志查询的实现是基于 GUI 界面完成，在内部可以按照类、异常类型和时间进行筛选，并对原本的日志信息进行处理后在界面输出，相关的操作如下所示：



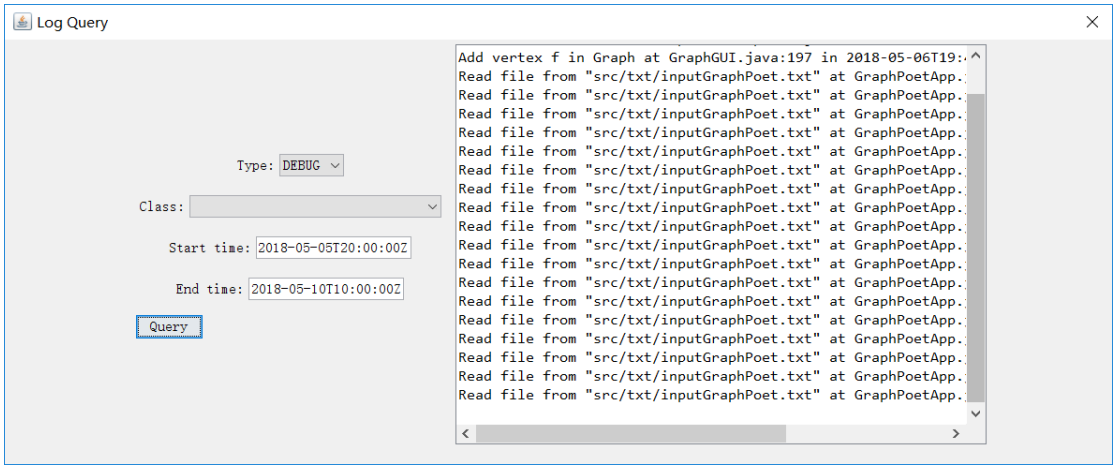
log 下可以进入查询界面。



可以选择相关的 Type（级别）和 class 类



也可以选择起始时间进行查询：



查询结果的显示如上图。

3.4 Testing for Robustness and Correctness

3.4.1 Testing strategy

测试策略，分别对不同的类进行测试的时候进行说明。

3.4.1.1 Graph 的 Test Strategy

Graph 中定义了多个函数，分别对其中的每个函数进行测试，运用等价类划分的方法进行和最小覆盖的方法进行。

（1）addVertex 函数增加的顶点分别是 null 类型的顶点和 normal 类型的顶

点。在具体的 Graph 的测试中（比如 GraphPoet 等）还需要测试 normal correct vertex class 和 normal incorrect vertex class 的顶点进行测试。

（2）对于 removeVertex 函数需要测试的就是 null、在图中的顶点没有边连接，在图中顶点有普通边连接，在图中有超边连接和不在图中分别进行测试。

（3）对于 Vertices 和 edges 函数的测试则主要测试在有顶点（边）在图中的和没有顶点（边）在图中的时候可以得到正确的结果。

（4）对于 sources 和 targets 函数，则主要测试顶点没有边连接，顶点有一条边连接和顶点有多条边连接的情况，可以得到正确的结果。

（5）对于 addEdge 和 removeEdge 的测试策略和 addVertex 及 removeVertex 的测试方法相同。

3.4.1.2 Vertex 的测试策略

对于 Vertex 类主要的测试函数就是 fillVertexInfo，对于其余的 equals、hashCode 和 toString 方法并不是不测试，而是由于其功能的单一性，只进行对其正确性的测试。

（1）fillVertexInfo 函数的测试添加的 args 参数，主要分为 null，不合法的参数个数，正确的参数个数但是非法的参数，以及正确的参数列表几种情况。

（2）对于 equals 的测试，主要测试其满足自反性、传递性和对称性。

3.4.1.3 Edge 的测试策略

（1）对于 Edge 主要的测试的函数就是 addVertices，对于其参数 Vertices

的 list，主要划分为以下几部分 null，不合法个数的顶点个数，合法的顶点个数但是不合法的顶点类型的 list 和合法的数据。

（2）对于其余的函数则主要是测试正确性，vertices，sourceVertices、targetVertices 和继承于 Object 的覆写函数，均测试其正确性。

3.4.2 测试用例设计

测试用例的设计主要根据上述提到的测试策略进行，保证每一组等价类划分至少有一组测试用例包含，并且为了减少赘余的测试用例，没有采用笛卡尔乘积的测试用例设计方式。

具体的设计的测试用例见 test 文件夹内的测试。

3.4.3 测试运行结果与覆盖度报告

用 maven 的测试的结果如下：

```
[INFO] Results:
[INFO]
[INFO] Tests run: 122, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 21.475 s
[INFO] Finished at: 2018-05-20T10:37:57+08:00
[INFO] Final Memory: 29M/303M
[INFO] ------------------------------------------------------------------------
```

测试覆盖度报告见 Lab4-1160300314/doc/reports

3.5 FindBugs tool（可选）

发现了哪些错误，每种错误代表什么不良的编程习惯
对代码修改，消除这些错误。

发现的错误主要有这么几类

（1）依赖于平台编码的 bug，由于不同的平台使用的编码不同所以直接使用类似于以下语句

```
InputStreamReader inputStreamReader
        = new InputStreamReader(new FileInputStream(fileName));
```

当使用不同的平台的时候就有可能产生完全不同的结果。因此需要改为以下的方式进行 IO：

```
InputStreamReader inputStreamReader
      = new InputStreamReader(new FileInputStream(fileName), "utf-8");
```

（2）有定义但未使用的代码存在，所有的变量定义后必须使用，否则也会被作为 findbugs 处理。所以将定义了但没有使用的变量删除。

3.6 Debugging

3.6.1 待调试程序

CalculatorGUI 测试

geometryProcessor 测试

textProcesser 测试

webDownloader 测试

3.6.2 理解待调试程序的过程

3.6.2.1 CalculatorGUI 测试

根据实验手册 Spring2018_HITCS_SC_Lab4 中的要求，需要 Calculator 显示一个计算器的界面，在其中显示 0-9 和+-*/四种运算的界面，并实现基本的四则运算。

3.6.2.2 geometryProcesser 测试

根据实验手册实现相应的随机效果，输出几行面积长度颜色的信息。

3.6.2.3 textProcesser 测试

按照顺序输出一棵 trie 树

3.6.2.4 webDownloader 测试

在一个指定网址上爬下来一些文件

3.6.3 发现并定位错误的过程

3.6.3.1 CalculatorGUI 测试

首先非常容易发现，所有的四则运算给出的运算符都是混乱的，简单修改一下就行。还有循环变量都没有用 for-each 的做，其中甚至有的地方定义了循环变量 i，但是没有用到。

```
if (!init && !buffer.equals("")) {
    result = Float.parseFloat(buffer);
    txtResult.setText("" + result);
    buffer = "";
}
```

另外在截取的这部分没有使用初始化 init = true，所以导致每一次循环都会导致其进入这个分支，导致运算的逻辑变化。

3.6.3.2 geometryProcesser 测试

可以很明显的看到，在一些需要计算面积的地方给出的公式是错误的，简单修改圆、正方形、三角形的面积计算公式即可。循环变量的写法也出错了，简单

修改。

3.6.3.3 textProcesser 测试

在测试中经常出现空指针引用，在 Trie 中。所以定位到 pointer 指针没有初始化便被引用。

3.6.3.4 webDownloader 测试

在其中的很多 while 循环直接使用了不可能到达的变量值，所以无法进入循环进行选择。并且所有的文件的后缀（“.txt”，“.pdf”，“.mp3”）都是 4 个字符长度，但在使用的时候作为三个进行的测试。

3.6.4 如何修正错误

3.6.4.1 CalculatorGUI 测试

修正循环变量，修改运算的逻辑，在上面提到的分支里面增加初始化 init 的部分。

3.6.4.2 geometryProcesser 测试

修改面积计算公式和循环变量输出即可。

3.6.4.3 textProcesser 测试

将 pointer 在未初始化的时候，指向根节点即可。

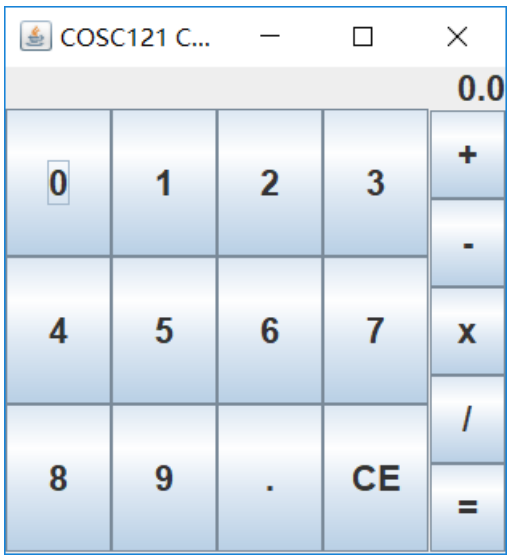
3.6.4.4 webDownloader 测试

将循环变量更改，修改后缀的长度使用，并且将下载设置为在文件下载失败的

时候不立即退出，而是将所有选择的文件都进行下载后再退出，打印错误信息。

3.6.5 结果

3.6.5.1 CalculatorGUI 测试



3.6.5.2 geometryProcesser 测试

debug 后的输出结果如下所示

Printing all shapes grouped by array

Shape: Circle, Name: Circle00, Area: 17.47156183683782, Colour: Red
Shape: Square, Name: Square01, Area: 1.0879094758226022, Colour: Green
Shape: Triangle, Name: Triangle02, Area: 11.089214856132438, Colour: Blue
Shape: Circle, Name: Circle10, Area: 41.981303219130474, Colour: Red
Shape: Square, Name: Square11, Area: 90.85230222222698, Colour: Green
Shape: Triangle, Name: Triangle12, Area: 13.400886308699352, Colour: Blue
Shape: Circle, Name: Circle20, Area: 35.96334670901794, Colour: Red
Shape: Square, Name: Square21, Area: 30.09168790473432, Colour: Green
Shape: Triangle, Name: Triangle22, Area: 4.5844606245816, Colour: Blue

Printing out shapes grouped by type...

Shape: Circle, Name: Circle00, Area: 17.47156183683782, Colour: Red
Shape: Circle, Name: Circle10, Area: 41.981303219130474, Colour: Red
Shape: Circle, Name: Circle20, Area: 35.96334670901794, Colour: Red
Shape: Square, Name: Square01, Area: 1.0879094758226022, Colour: Green
Shape: Square, Name: Square11, Area: 90.85230222222698, Colour: Green
Shape: Square, Name: Square21, Area: 30.09168790473432, Colour: Green
Shape: Triangle, Name: Triangle02, Area: 11.089214856132438, Colour: Blue
Shape: Triangle, Name: Triangle12, Area: 13.400886308699352, Colour: Blue
Shape: Triangle, Name: Triangle22, Area: 4.5844606245816, Colour: Blue

3.6.5.3 textProcesser 测试

debug 后的输出结果（由于没有测试中给出的文件，所以增加了 testText.TXT 文件作为测试用的文件，里面只有一句话“Hello, world!”）

Node Value: debug.textProcesser.Node@3b9a45b3, Children: H

Node Value: debug.textProcesser.Node@7699a589, Children: e

Node Value: debug.textProcesser.Node@58372a00, Children: l

Node Value: debug.textProcesser.Node@4dd8dc3, Children: l

Node Value: debug.textProcesser.Node@6d03e736, Children: o

Node Value: debug.textProcesser.Node@568db2f2, Children: ,

Node Value: debug.textProcesser.Node@378bf509, Children: *
w

Node Value: debug.textProcesser.Node@5fd0d5ae, Children: , isLeaf: true, hashcode: debug.textProcesser.Node@5fd0d5ae
Node Value: debug.textProcesser.Node@2d98a335, Children: o

Node Value: debug.textProcesser.Node@16b98e56, Children: r

Node Value: debug.textProcesser.Node@7ef20235, Children: l

Node Value: debug.textProcesser.Node@27d6c5e0, Children: d

Node Value: debug.textProcesser.Node@4f3f5b24, Children: !

Node Value: debug.textProcesser.Node@15aeb7ab, Children: *

Node Value: debug.textProcesser.Node@7b23ec81, Children: , isLeaf: true, hashcode: debug.textProcesser.Node@7b23ec81

{Hello,=[0], world!=[0]}

3.6.5.4 webDownloader 测试

debug 后的测试结果

```
1
https://people.ok.ubc.ca/rlawrenc/teaching/304/Notes/304_5_SQL_answers.pdf
Download 304_5_SQL_answers.pdf failed
e:\test\refs\rulesfordatastorage.pdf (系统找不到指定的路径。)
Download refs/rulesfordatastorage.pdf failed
Execution time = 379 seconds
47 file(s) downloaded
Program exiting, goodbye!
```

4 实验进度记录

请尽可能详细的记录你的进度情况。

日期	时间段	计划任务	实际完成情况
5.6		完成 debug 任务	完成
5.8-5.18		增加 test 用例	完成
5.18-5.20		写实验报告	完成

5 实验过程中遇到的困难与解决途径

实验中 findbugs 测试出的 bug 很多都是来自于依赖平台编码的 IO 错误，但是由于对于 Java 的 IO 相关的类不够了解所以在 findbugs 时候上网查阅了相关的官方文档（已经不更新的 findbugs）和博客。

对于 Exception 的相关内容，借鉴了 Effective Java（2nd edition）中的对于 Exception 编写的一些准则。

6 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

(1) 健壮性和正确性，二者对编程中程序员的思路有什么不同的影响？

(2) 为了应对 1%可能出现的错误或异常，需要增加很多行的代码，这是否划算？

(3) “让自己的程序能应对更多的异常情况”和“让客户端/程序的用户承担确保正确性的职责”，二者有什么差异？你在哪些编程场景下会考虑遵循前者、在哪些场景下考虑遵循后者？

(4) 过分谨慎的“防御”（excessively defensive）真的有必要吗？

(5) 通过调试发现并定位错误，你自己的编程经历中有总结出一些有效的方法吗？请分享之。Assertion 和 log 技术是否会帮助你更有效的定位错误？

(6) 怎么才是“充分的测试”？代码覆盖度 100%是否就意味着 100%充分的测试？

(7) 关于本实验的工作量、难度、deadline。

(8) 到目前为止你对《软件构造》课程的评价和建议。

（1）对于正确性编程的程序员，可以参考相关飞行器的程序员，他们对于正确性的要求远高于其余场合，所以他们对于错误的输入和抛出的异常必须处于零容忍。而一般的应用级别的程序员对于健壮性的要求更高，需要优雅的退出。

（2）划算的。首先放结论，因为 exception 的作用就是保证整个程序可以应对更多的情况。在开发的时候可以考虑更多的错误想信息，就能够降低错误来临的时候将整个系统高崩溃的概率。

（3）前者更多的是商用的面向于普通大众的程序，这样的程序不能时不时就蓝屏，这样产生的影响会造成用户对于不好的影响。而对于高级别的用户，比如用户就是程序员群体，有能力去处理其中的问题并且对于这样的问题容忍度也更高，就有可能将错误让用户去处理。

（4）这个需要辩证的看吧，首先如果安全性特别高（国防航天领域等等），可能答案就是再高也不过分；但是在追求效率的领域，过分谨慎的防御式编程就对应于效率的下降，这可能是得不偿失的。

（5）我觉得最后的办法还是 stack trace 可能有着最丰富的信息和最好的调试帮助。直接去看调用栈，然后利用分部调试的方法，去查看中间变量可能是对于我最好的调试方法，能够最快的发现问题。

（6）充分的测试，应该是能够覆盖所有的分支和圈。100%不一定意味着测试充分，因为大部分的测试覆盖度软件生成的都是对于语句的覆盖度。可能有很多的分支并没有达到，还是可以达到一个接近于 100%的覆盖度。同样有些分支由于函数 signature 和编译器对于异常的要求，可能就有语句覆盖无法达到的地方，这样的一些问题可能与强行达到百分之百的测试覆盖度没有什么可比性。

（7）工作量与所占的分数有点不成比例，工作量在整个学期都很大，就拿一个报告来说，可能很多童鞋的报告都可以平均达到 30 页以上，这样的量可以说相当大了。

（8）建议对于实验，可以有老师或者 TA 在实验前能够进行一下回归测试，这样对于其中的一些小问题可能发现的更早也有利于实验手册的完善。