

哈尔滨工业大学计算机科学与技术学院

2017 年秋季学期《软件工程》

## Lab 4: 代码评审与程序性能优化

姓名	学号	联系方式
张宁	1150310607	3294349775@qq.com/17390605885
马玉坤	1150310618	mayukuner@126.com/18845895386

## 目 录

1	实验要求.....	1
2	在 Eclipse 中配置代码审查与分析工具 .....	1
2.1	Checkstyle .....	1
2.2	PMD .....	3
2.3	FindBugs .....	4
2.4	VisualVM .....	6
3	本次实验所评审的代码.....	9
4	代码 review 记录.....	10
5	Checkstyle 所发现的代码问题清单及原因分析 .....	11
6	PMD 所发现的代码问题清单及原因分析 .....	11
7	FindBugs 所发现的代码问题清单及原因分析 .....	12
8	VisualVM 性能分析结果 .....	14
8.1	执行时间的统计结果与原因分析.....	14
8.2	内存占用的统计结果与原因分析.....	16
8.3	代码改进之后的执行时间统计结果 .....	20
8.4	代码改进之后的内存占用统计结果.....	20
9	利用 Git/GitHub 进行协作的过程.....	21
10	评述.....	28
10.1	对代码规范方面的评述.....	28
10.2	对代码性能方面的评述.....	28
11	计划与实际进度.....	28
12	小结.....	29
13	Travis CI 的配置.....	30

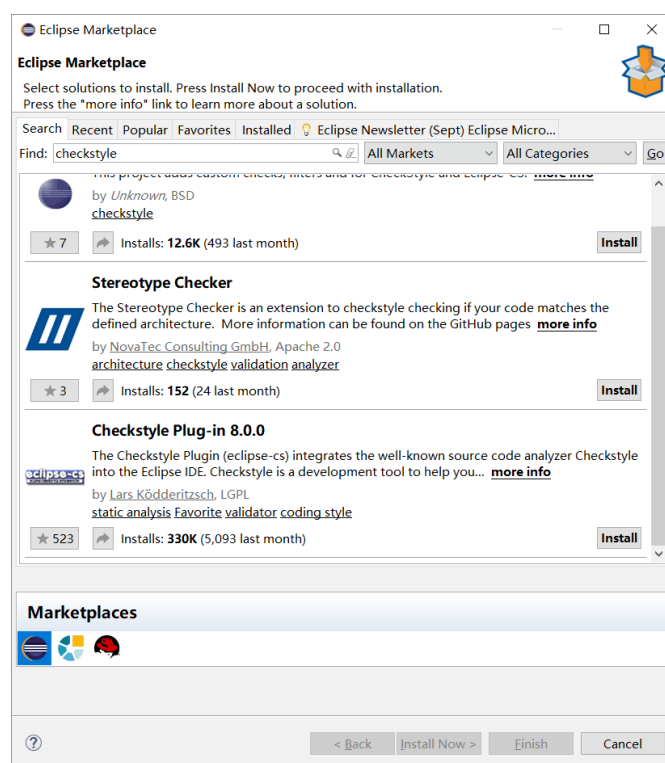
## 1 实验要求

- 1.1 根据老师发布在乐学网上的分组信息,找到我们对应的另一组同学在 github 上的项目地址。
- 1.2 将另外一组同学的 github 项目拷贝到本地。
- 1.3 在 eclipse 中配置 Checkstyle, FindBugs, PMD, VisualVM 等工具。
- 1.4 对另外一组的代码进行代码评审(走查)
- 1.5 使用 Checkstyle, FindBugs, PMD 对代码进行静态分析
- 1.6 通过 VisualVM 进行动态分析,找到性能的瓶颈,从性能角度对代码进行优化。

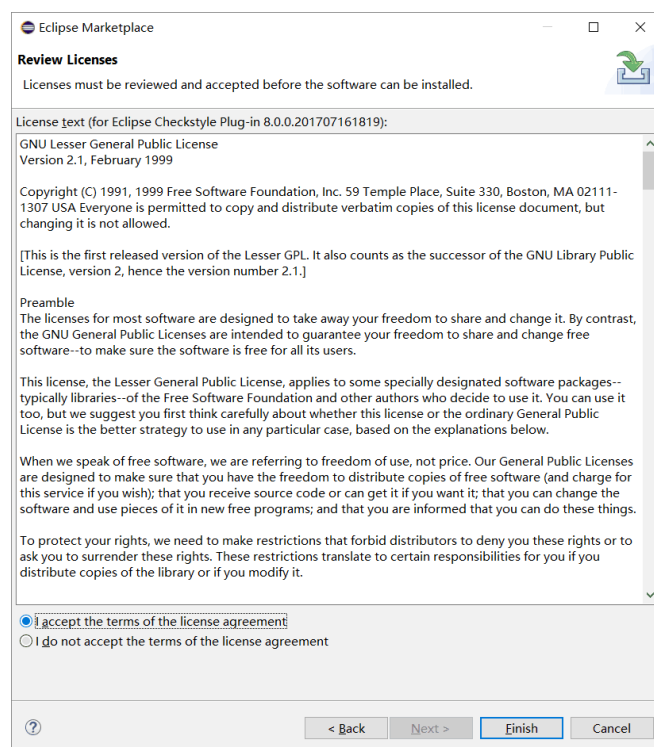
## 2 在 Eclipse 中配置代码审查与分析工具

### 2.1 Checkstyle

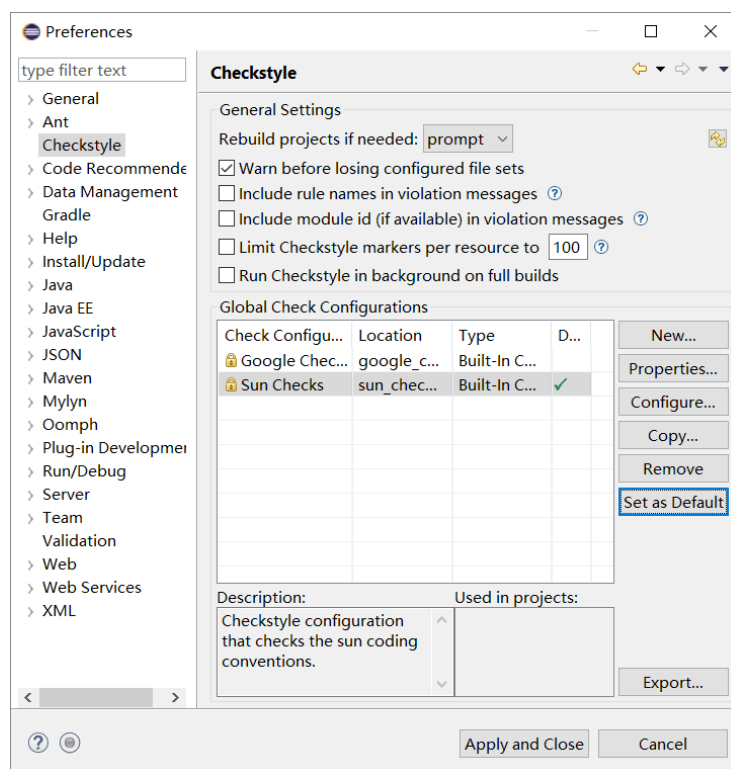
1. 在 Eclipse Marketplace 中搜索”checkstyle”, 点击 Install



2. 接受用户协议, 点击 Finish

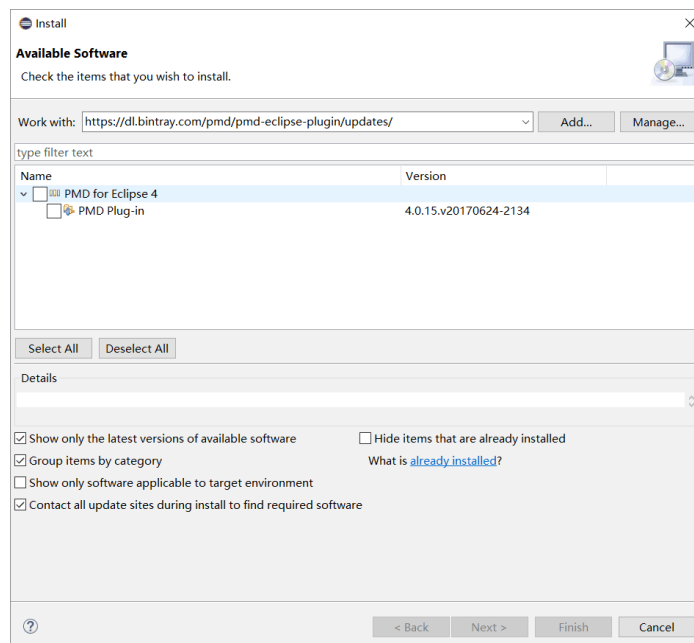


3. 重启之后, 打开 Window-Preference 窗口, 找到 Checkstyle, 在右侧将 Sun Checks 设为默认。

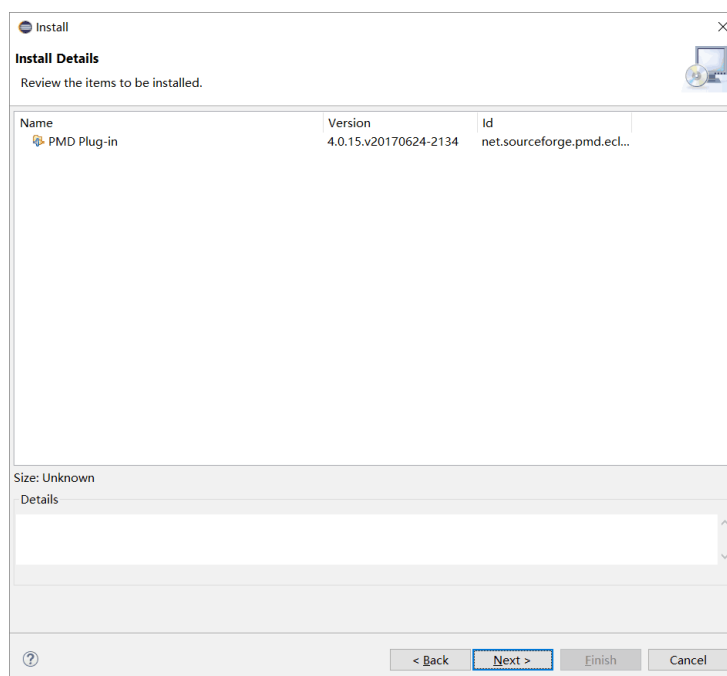


## 2.2 PMD

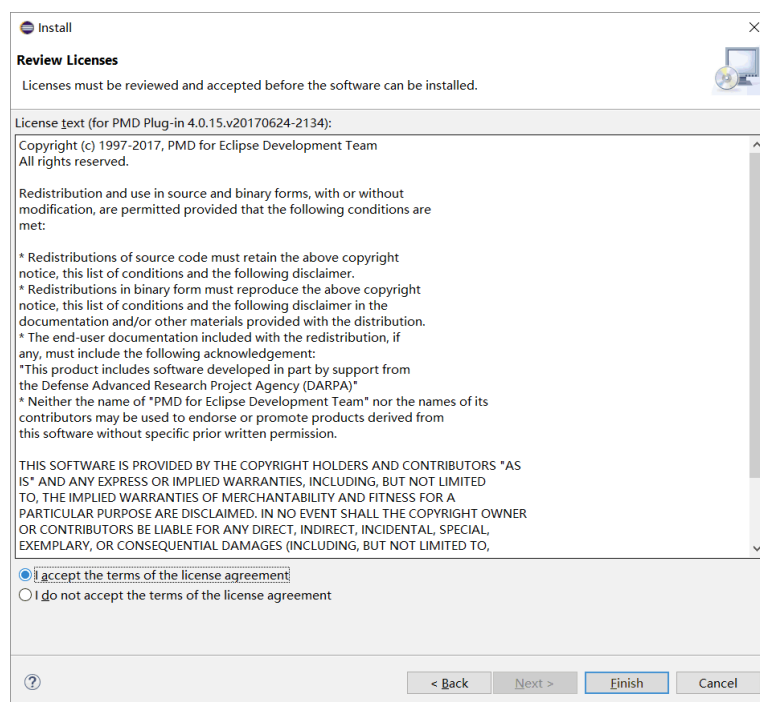
1. 打开 Eclipse，打开“Help” - “Install New Software”，在“Work with:”后输入“<https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/>”，点击 Add。在下方选择“PMD Plug-in”，然后点击“Next”。



2. 检查安装细节，点击“Next”。



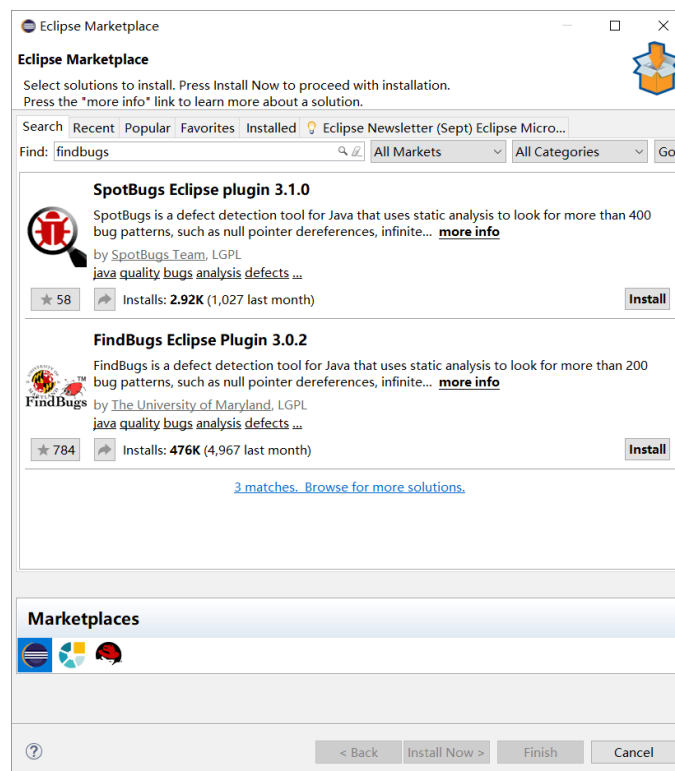
3. 接受用户协议之后，点击 Finish，安装成功。



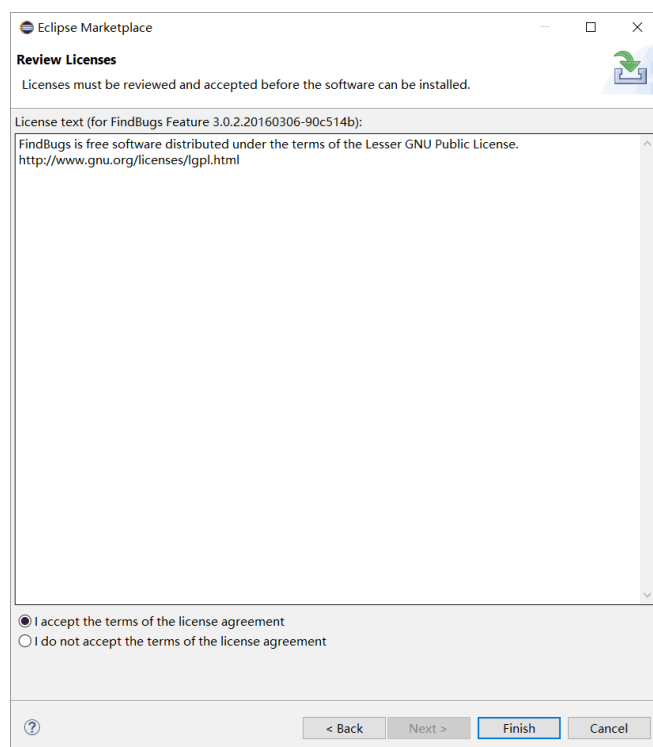
4. 重启 Eclipse 后即可使用，无需配置。

## 2.3 FindBugs

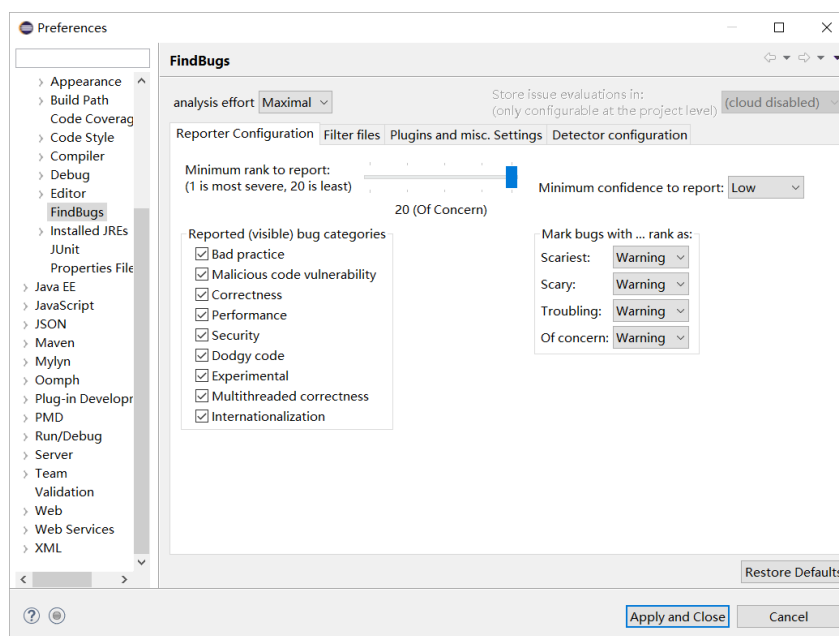
1. 在 Eclipse Marketplace 中搜索 “Findbugs”，点击 Install。

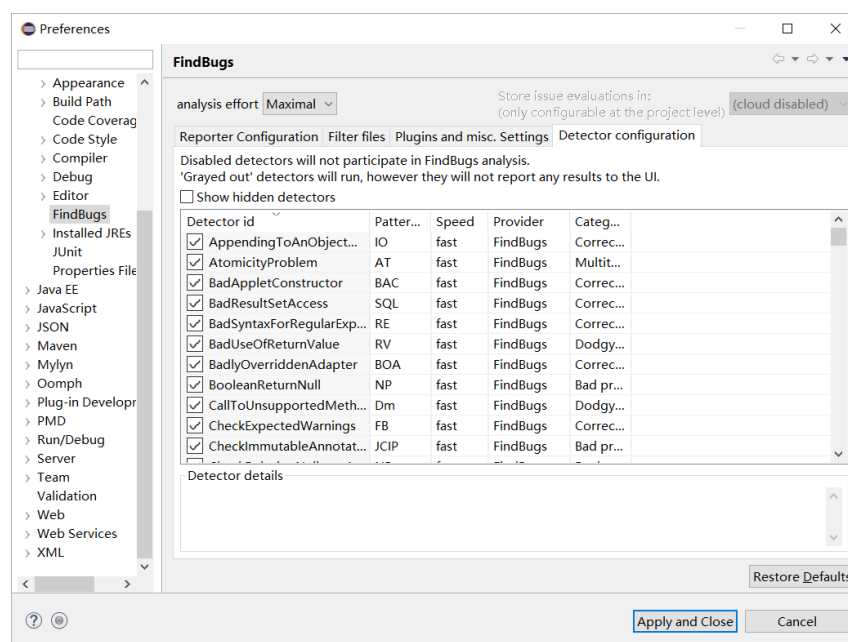


2. 接受用户协议，点击 Finish。



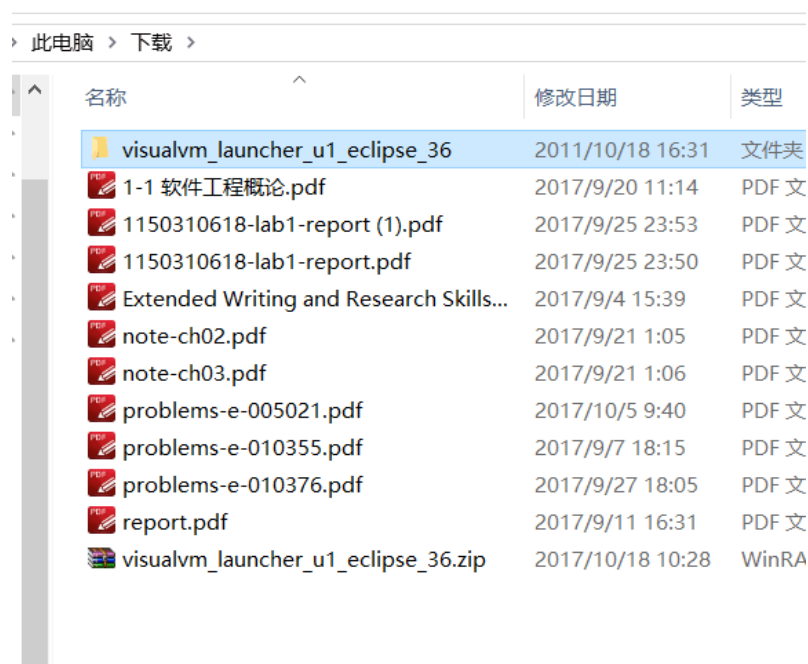
3. 安装完成后, 重启 Eclipse。在 Preference 窗口中, 找到 Java-Findbugs, 在右侧 analysis effort 选择 “Maximal”, 报告等级设为 20, confidence 设为 Low, 然后将检测项全部选择, 之后应用修改。





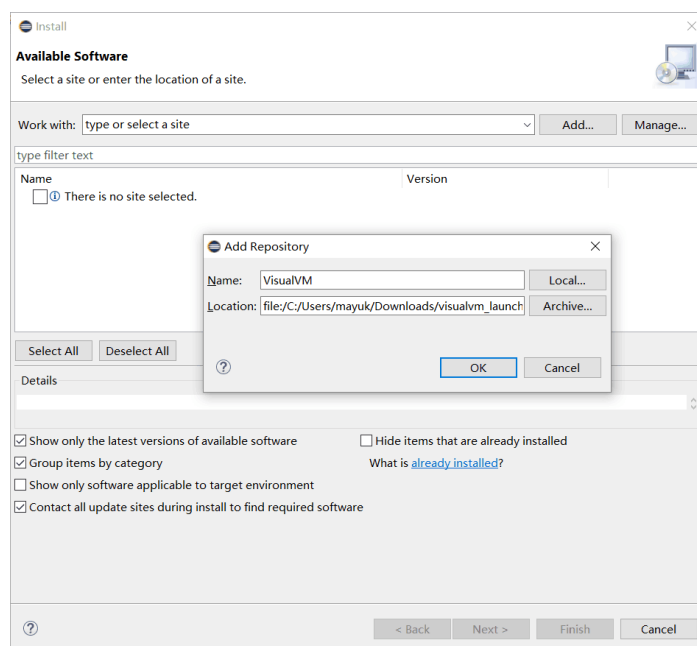
## 2.4 VisualVM

1. 在 <https://visualvm.github.io/idesupport.html> 页面上下载 Eclipse Launcher 到本地文件夹，并解压。

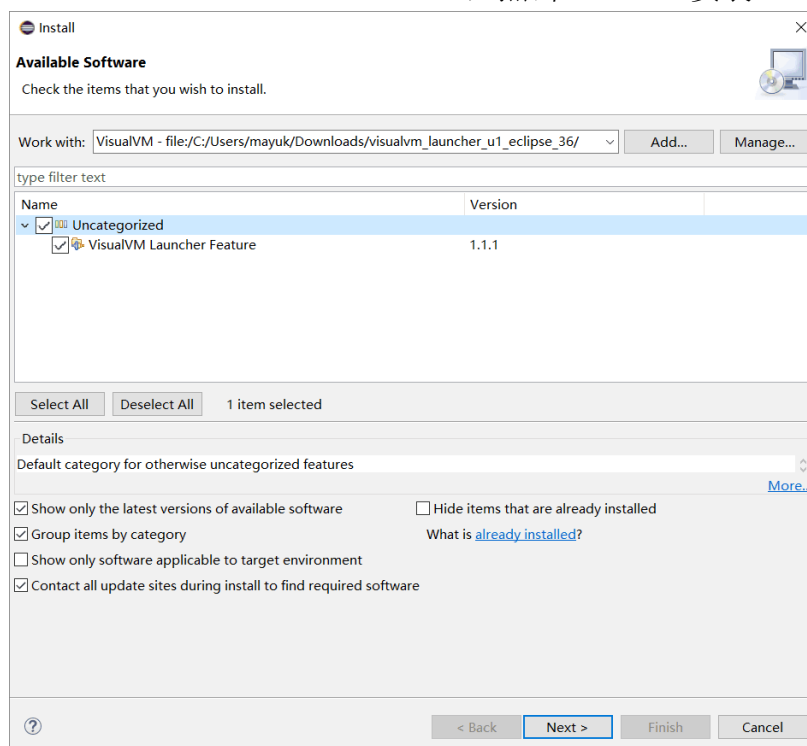


2. 打开 Eclipse，点击“Help”-“Install New Software”。点击“Add”添加 Repository。Name 任意，Location 填写下载并解压的 launcher 文件夹的路径，并点击“OK”。

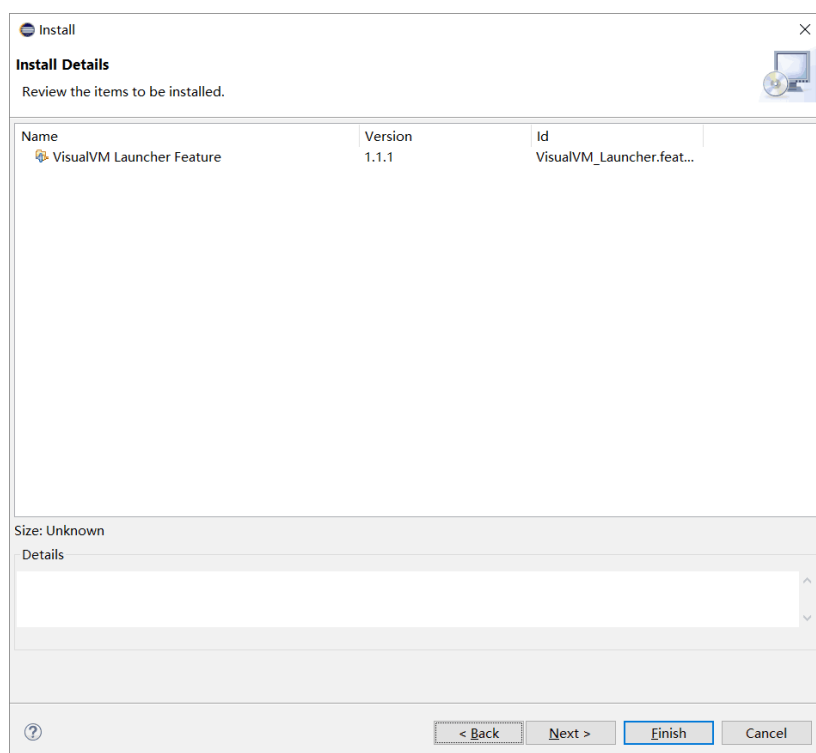




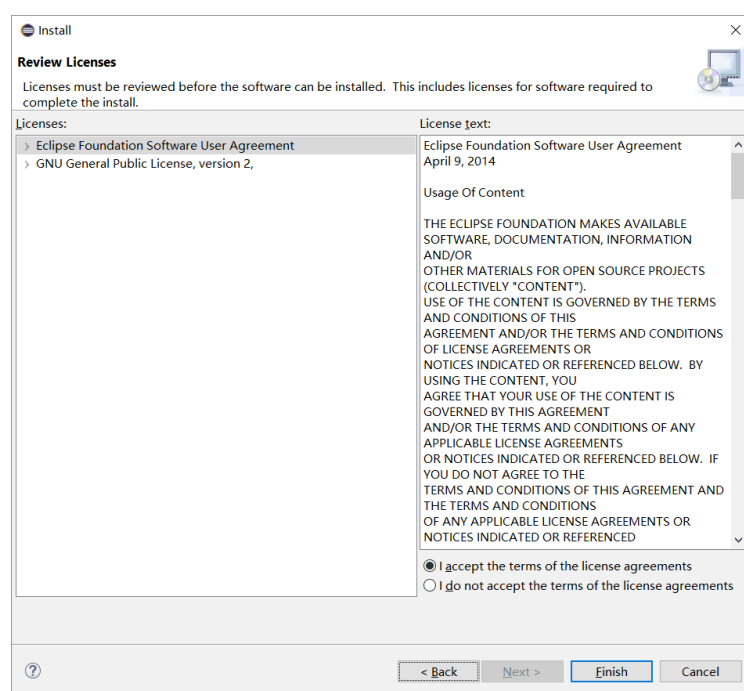
3. 选择“VisualVM Launcher Feature”，点击“Next”安装。



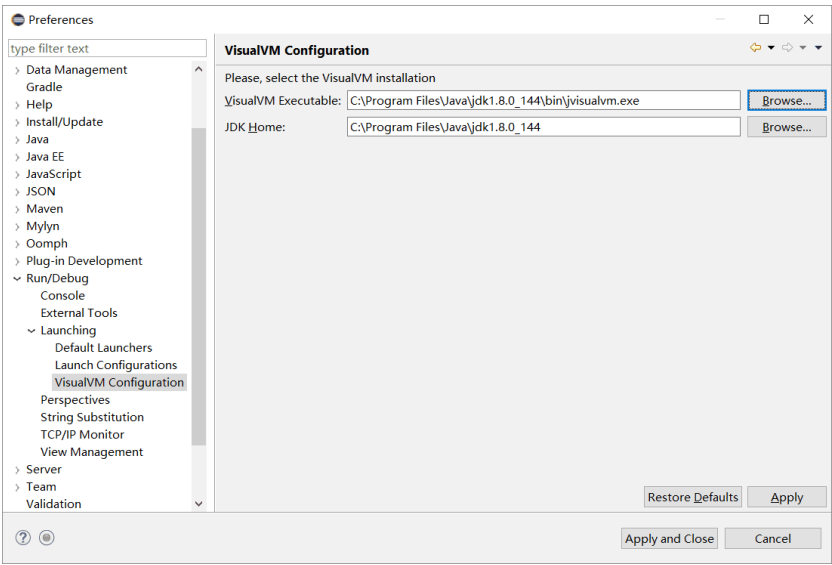
4. 依赖下载完成后，点击 Finish。



5. 接受用户协议，点击 Finish 完成安装。

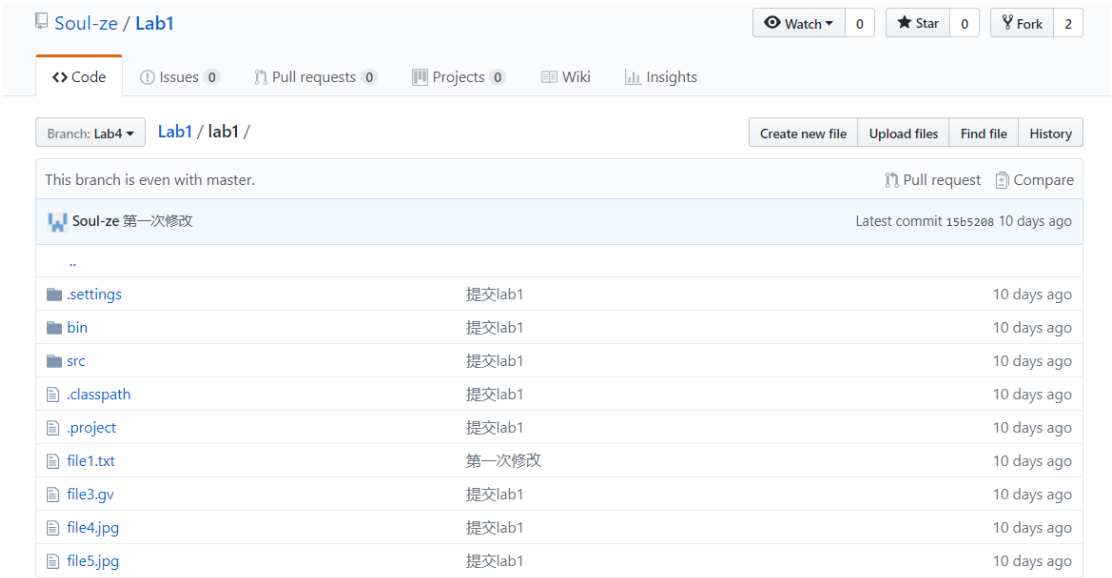


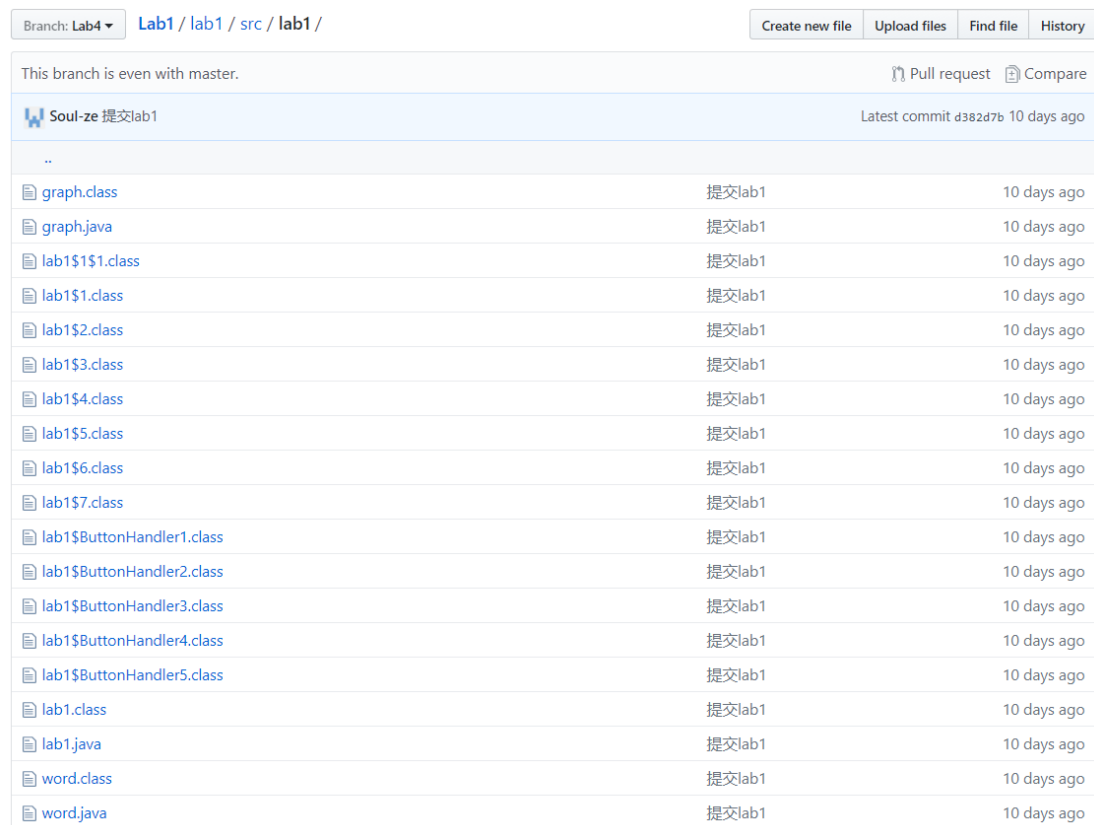
6. 重启 Eclipse 之后，打开 Preference 窗口，找到 “Run/Debug” – “Launching” – “VisualVM Configuration”，在右侧 “VisualVM Executable” 中填入 VisualVM 可执行文件的路径（在 jdk 的 bin 文件夹下）。



3 本次实验所评审的代码

姓名	学号	GitHub 地址
唐铭泽	1150310523	<a href="https://github.com/Soul-ze/Lab1">https://github.com/Soul-ze/Lab1</a>
姚铖栋	1150310504	<a href="https://github.com/The-water-tree/Lab1">https://github.com/The-water-tree/Lab1</a>





## 4 代码 review 记录

问题描述	类型	所在代码行号	修改方式
类名小写开头	类命名问题	Lab1.java:12 Graph.java:5 Word.java:6	将类名及类被引用的部分全部进行修改（eclipse 有自动重构功能）
引入了代码不需要的包	引用问题	Word.java:4	删除 import 语句
Vertical1 和 Horizontal1 两个变量名首字母大写	变量命名问题	lab1.java:17	将变量首字母改为小写，并使用替换功能，将使用到两个变量的代码全部进行修改
使用 dot 绘制图像的命令错误，使用了绝对路径，在评审人计算机上无法正常运行	文件路径问题	Lab1.java:384	使用代码 <code>System.getProperty("user.dir")</code> 获得项目的路径，获得正确的绝对路径

## 5 Checkstyle 所发现的代码问题清单及原因分析

编号	问题描述	类型	所在代码行号	修改策略
1	'else' 结构必须使用大括号 '{}'	条件分支格式问题	word.java: 41	在 else 后面手动添加 {}
2	',' 后应有空格。	语句格式问题	lab1.java: 217	使用全局替换,在部分 “,” 前面添加空格
3	'(' 后不应有空格。	表达式格式问题	lab1.java: 89	手动将 “(” 后的空格删除
4	'class def modifier' 缩进了 4 个缩进符, 应为 2 个。作者使用 Tab 进行了缩进	代码缩进问题	lab1.java: 44	对 Eclipse 的 Formatter 进行设置, 将缩进符修改为 4 个空格, 并使用 eclipse 的 Formatter 对代码进行格式化
5	'10' 是一个魔术数字 (直接常数)。	常量使用问题	lab1.java: 209	定义一个常量, 赋值为 10, 并将 10 出现的位置修改为常量
6	本行字符数 170 个, 最多: 80 个。	行字数限制问题	lab1.java: 385	只需要原语句使用换行断开即可, 将一条语句拆成 3 行
7	'while' 结构必须使用大括号 '{}'	循环结构格式问题	lab1.java: 412	在 while 标识符后手动添加大括号
8	'{' 后应有空格。	代码块格式问题	graph.java: 16	使用替换功能, 向部分不满足格式的 “{” 符号后添加空格

## 6 PMD 所发现的代码问题清单及原因分析

优先级	问题描述	违反的规则集	所在代码行号	修改策略
Important	作者将两个变量的初始	<i>Controversial</i>	Lab1.java:540	将两个变量

t	化放在了同一行.	<i>OneDeclarationPerLine</i>		的初始化 放在两行进行.
Warning	变量重复初始化 <code>String[] stringBuffer = null;</code> <code>stringBuffer = null;</code>	<i>Controversial</i> <i>DataflowAnomalyAnalysis</i>	Lab1.java:334	删除第 335 行代码
Urgent	注释写的位置不符合代码规范,把注释放在了方法名字的后面 <code>void</code> <code>showDirectedGraph(graph G) { // 图形化展示有向图</code>	<i>Comments</i> <i>CommentDefault</i> <i>AccessModifier</i>	Lab1.java:373	将注释放在方法命名的前面
Urgent	”Avoid using final local variables” 作者在一个方法中声明了一个 final 变量	<i>Controversial</i> <i>AvoidFinalLocalVariable</i>	Graph.java:88	将 final 变量的声明移到方法外
Urgent	If else 语句中条件为真时,没有可执行语句,只用了一个; <code>if (sign == 1)</code> <code>; // 判断图的边是否出现过</code>	<i>Empty code</i> <i>EmptyIfStmt</i>	Graph.java:58	通过将判断条件取反,将 ifelse 语句转化为 if 语句
Urgent	变量命名短,只有一个字母	<i>Naming</i> <i>ShortVariable</i>	Graph.java:133	将变量重新命名,全部替换
Important	一个库函数被重复导入	<i>Import statements</i> <i>DuplicateImports</i>	Lab1.java:10	删除重复同一个库的代码
Urgent	使用了 <code>Buffer.size() &gt; 0</code> 这样的表达式	<i>Design</i> <i>UseCollectionIsEmpty</i>	Lab1.java:367	替换为 <code>!Buffer.empty()</code>
Urgent	字符串变量 string3 定义后未被使用	<i>UnusedLocalVariable</i>	lab1.java:36	删除 string3 的定义语句

## 7 FindBugs 所发现的代码问题清单及原因分析

问题描述	类型	所在代码行号	修改策略
初始化 <code>InputStreamReader</code>	默认编码依赖	lab1.java:33	显式声明所用编码, 将

<p>时未限定编码，而是使用环境默认编码</p> <pre>InputStreamReader dis = new InputStreamReader(fis);</pre>	<p>(reliance on default encoding)</p>	8	<p>InputStreamReader 所用编码设为 UTF-8</p> <pre>InputStreamReader dis = new InputStreamReader(fis, "UTF-8");</pre>
<p>调用 String 的空构造函数，浪费空间</p> <pre>String s = new String();</pre>	<p>无效 String()构造函数调用 (inefficient new String() constructor)</p>	graph.java:72	<p>删除调用 String 空构造函数的语句。</p> <pre>String s;</pre>
<p>对同一变量连续执行两次赋值语句</p> <pre>wordName = new String(); wordName = name;</pre>	<p>双赋值语句 (double assignment of field)</p>	word.java:17	<p>删除第一个赋值语句。</p> <pre>wordName = name;</pre>
<p>使用 “+” 运算符，执行多次 String 变量后接字符串的操作</p> <pre>String s = ""; for (int i = 0; i &lt; field.length; ++i) {     s = s + field[i]; }</pre>	<p>在循环体中用+连接字符串 (concatenates strings using + in a loop)</p>	graph.java:75	<p>改使用 StringBuffer 以及 append、toString 方法</p>
<p>对变量进行无用的初始化（变量在其后会被重新赋值）</p> <pre>Vector&lt;String&gt; bridgeWords = new Vector&lt;String&gt;();</pre>	<p>本地变量存储了闲置不用的对象 (dead store to local variable)</p>	lab1.java:472	<p>删除对象构造的语句即可。</p> <pre>Vector&lt;String&gt; bridgeWords;</pre>
<p>添加监听事件时，匿名对象调用了外部类的私有成员变量</p> <pre>private Graph G;</pre>	<p>内层类方法调用了外层类的私有成员(method accesses to a private member variable of owning class)</p>	lab1.java:112	<p>将被调用成员从 private 类型修改为 public 类型。</p> <pre>public Graph G;</pre>
<p>使用 toLowerCase 函数时，未指定所使用的区域（语言），可能会因为环境的默认区域改变而导致代码的运行结果出现</p>	<p>使用了未指定区域(语言)的 toLowerCase (use of non-localized)</p>	lab1.java:345	<p>在 toLowerCase 的参数中，加入 Locale.US，规定区域为美国。</p> <pre>tmp = tmp.toLowerCase(Locale.U</pre>

错误。 tmp = tmp.toLowerCase();	tring.toLowerCase ())		S);
------------------------------------	--------------------------	--	-----

## 8 VisualVM 性能分析结果

### 8.1 执行时间的统计结果与原因分析

#### 1. 性能测试方案

- 对“展示有向图”、“查询桥接词”、“根据 bridge word 生成文本”、“计算两个单词的最短路径”、“一个单词到其他单词的最短路径”以及“随机游走”六个功能分别调用三次
- 使用 VisualVM 中的 Profiler 进行统计
- 将每个函数按照“自用时间”、“总时间”、“调用次数”分别进行排序分析

#### 2. 测试结果

- 按照自用时间排序

热点 - 方法	自... ▾	自用时间	总时间	调用
lab1.Lab1.showDirectedGraph (lab1.Graph)		16,006 ms (54.5%)	16,013 ms	43
lab1.Lab1\$.actionPerformed (java.awt.eve...		9,541 ms (32.5%)	9,721 ms	1
lab1.Lab1.calcShortestPath (lab1.Graph, S...		1,022 ms (3.5%)	8,373 ms	34
lab1.Lab1\$.actionPerformed (java.awt.eve...		906 ms (3.1%)	1,128 ms	1
lab1.Lab1\$.actionPerformed (java.awt.event.ActionEvent)		66.4 ms (0.2%)	292 ms	1
lab1.Lab1.randomWalk (lab1.Graph)		39.7 ms (0.1%)	273 ms	1
lab1.Graph.getMatrix ()		29.7 ms (0.1%)	32.0 ms	35
lab1.Lab1\$.actionPerformed (java.awt.eve...		24.7 ms (0.1%)	24.7 ms	1
lab1.Lab1\$.actionPerformed (java.awt.eve...		21.4 ms (0.1%)	242 ms	1
lab1.Lab1\$.actionPerformed (java.awt.eve...		17.5 ms (0.1%)	245 ms	1
lab1.Lab1\$.actionPerformed (java.awt.eve...		17.2 ms (0.1%)	17.3 ms	1
lab1.Lab1.createDirectedGraph (String)		14.9 ms (0.1%)	29.1 ms	4
lab1.Graph.createGraph (String, String)		13.1 ms (0%)	14.0 ms	160
lab1.Lab1\$.ButtonHandler4.actionPerformed ...		9.93 ms (0%)	8,314 ms	1
lab1.Word.graphInf ()		5.7 ms (0%)	5.7 ms	1,408
lab1.Lab1\$.ButtonHandler1.actionPerformed ...		4.20 ms (0%)	4.37 ms	1
lab1.Word.judgeOfWord (String)		2.21 ms (0%)	2.20 ms	38,115
lab1.Lab1\$.ButtonHandler5.actionPerformed ...		1.88 ms (0%)	274 ms	1
lab1.Graph.getCommand ()		1.76 ms (0%)	6.83 ms	43
lab1.Lab1\$.ButtonHandler3.actionPerformed ...		1.28 ms (0%)	289 ms	1
lab1.Word.addLinkList (String)		0.621 ms (0%)	0.620 ms	147
lab1.Graph.getWordIndex (String)		0.568 ms (0%)	0.568 ms	456

- 按照总时间排序



热点 - 方法	自... ▾	自用时间	总时间	调用
lab1.Lab1.showDirectedGraph (lab1.Graph)		16,006 ms (54.5%)	16,013 ms	43
lab1.Lab1\$1.actionPerformed (java.awt.eve...		9,541 ms (32.5%)	9,721 ms	1
lab1.Lab1.calcShortestPath (lab1.Graph, S...		1,022 ms (3.5%)	8,373 ms	34
lab1.Lab1\$3.actionPerformed (java.awt.eve...		906 ms (3.1%)	1,128 ms	1
lab1.Lab1\$2.actionPerformed (java.awt.event.ActionEvent)		66.4 ms (0.2%)	292 ms	1
lab1.Lab1.randomWalk (lab1.Graph)		39.7 ms (0.1%)	273 ms	1
lab1.Graph.getMatrix ()		29.7 ms (0.1%)	32.0 ms	35
lab1.Lab1\$6.actionPerformed (java.awt.eve...		24.7 ms (0.1%)	24.7 ms	1
lab1.Lab1\$5.actionPerformed (java.awt.eve...		21.4 ms (0.1%)	242 ms	1
lab1.Lab1\$4.actionPerformed (java.awt.eve...		17.5 ms (0.1%)	245 ms	1
lab1.Lab1\$7.actionPerformed (java.awt.eve...		17.2 ms (0.1%)	17.3 ms	1
lab1.Lab1.createDirectedGraph (String)		14.9 ms (0.1%)	29.1 ms	4
lab1.Graph.createGraph (String, String)		13.1 ms (0%)	14.0 ms	160
lab1.Lab1\$ButtonHandler4.actionPerformed ...		9.93 ms (0%)	8,314 ms	1
lab1.Word.graphInf ()		5.7 ms (0%)	5.7 ms	1,408
lab1.Lab1\$ButtonHandler1.actionPerformed ...		4.20 ms (0%)	4.37 ms	1
lab1.Word.judgeOfWord (String)		2.21 ms (0%)	2.20 ms	38,115
lab1.Lab1\$ButtonHandler5.actionPerformed ...		1.88 ms (0%)	274 ms	1
lab1.Graph.getCommand ()		1.76 ms (0%)	6.83 ms	43
lab1.Lab1\$ButtonHandler3.actionPerformed ...		1.28 ms (0%)	289 ms	1
lab1.Word.addLinkList (String)		0.621 ms (0%)	0.620 ms	147
lab1.Graph.getWordIndex (String)		0.568 ms (0%)	0.568 ms	456

## c) 按照调用次数排序

热点 - 方法	自用时...	自用时间	总时间	调用 ▾
lab1.Word.judgeOfWord (String)		2.21 ms (0%)	2.20 ms	38,115
lab1.Word.getWeight (int)		0.101 ms (0%)	0.100 ms	2,030
lab1.Word.graphInf ()		5.7 ms (0%)	5.7 ms	1,408
lab1.Word.colorCleaned ()		0.261 ms (0%)	0.260 ms	1,188
lab1.Lab1.findMinCost (int, int[], boolea...		0.377 ms (0%)	0.377 ms	1,122
lab1.Graph.getWordNum ()		0.065 ms (0%)	0.064 ms	851
lab1.Graph.getWordIndex (String)		0.568 ms (0%)	0.568 ms	456
lab1.Word.getColor (String)		0.144 ms (0%)	0.143 ms	207
lab1.Word.changecolor (int, int)		0.078 ms (0%)	0.076 ms	203
lab1.Word.indexOfEdge (String)		0.057 ms (0%)	0.056 ms	203
lab1.Graph.changeEdgeColor (int, int, int)		0.129 ms (0%)	0.264 ms	203
lab1.Graph.createGraph (String, String)		13.1 ms (0%)	14.0 ms	160
lab1.Word.addLinkList (String)		0.621 ms (0%)	0.620 ms	147
lab1.Word.<init> (String)		0.153 ms (0%)	0.152 ms	88
lab1.Lab1\$1\$1.accept (java.io.File, String)		0.174 ms (0%)	7,376 ms	74
lab1.Word.searchLink (String)		0.114 ms (0%)	0.212 ms	73
lab1.Lab1.showDirectedGraph (lab1.Graph)		16,006 ms (54.5%)	16,013 ms	43
lab1.Graph.getCommand ()		1.76 ms (0%)	6.83 ms	43
lab1.Graph.colorClean ()		0.372 ms (0%)	0.632 ms	36
lab1.Graph.queryExist (String)		0.059 ms (0%)	0.059 ms	35
lab1.Graph.getMatrix ()		29.7 ms (0.1%)	32.0 ms	35
lab1.Lab1.calcShortestPath (lab1.Graph, S...		1,022 ms (3.5%)	8,373 ms	34

## 3. 测试结果分析

通过对每个函数的占用时间进行分析，我们能够发现以下结论：

- Lab1.showDirectedGraph()*函数的自用时间及总时间都很大，是整个程序运行效率的瓶颈
- Word.judgeOfWord()*、*Word.getWeight()*等方法调用次数最多，原因是其为有向图基本操作需调用的方法（比如判断两点是否有边，获得边权等等），但是由于结构简单，因此执行时间较小

#### 4. 改进方法

- 对 `Lab1.showDirectedGraph()` 方法以及调用 `Lab1.showDirectedGraph()` 方法的 `Lab1.calcShortestPath()` 与 `Lab1.singleSource()` 方法进行静态分析后, 我们发现执行 `Lab1.singleSource()` 方法求解图中一点与其他各点的最短路过程中, 用于在 GUI 中展示有向图 `Lab1.showDirectedGraph()` 会被执行若干次, 执行次数与有向图中的点数相同。尽管 `Lab1.showDirectedGraph()` 会被执行若干次, 但只有最后一次执行的结果对 GUI 展示的图像有效。
- 由于只有最后一次执行 `Lab1.showDirectedGraph()` 的结果有效, 因此可以设一个变量对图像是否需要重新绘制进行标记, 当必需重新绘制时, 调用 `Lab1.showDirectedGraph()`, 否则不对有向图对应的图像进行绘制。该标记变量会在 `Lab1.singleSource()` 方法被修改。

## 8.2 内存占用的统计结果与原因分析

### 1. 性能测试方案

- 分别执行“展示有向图”、“查询桥接词”、“根据 bridge word 生成文本”、“计算两个单词的最短路径”、“一个单词到其他单词的最短路径”以及“随机游走”六个功能
- 使用 VisualVM 的 Profiler 记录每个功能执行时的内存占用
- 对每个功能执行时的内存占用进行分析比较

### 2. 测试结果

- 展示有向图

类名 - 活动的分配对象	活... ▼	活动字节	活动对象	年代数
java.lang.Object[]		769,736 B (16.8%)	16,982 (16.1%)	97
java.util.HashMap		418,704 B (9.1%)	8,723 (8.3%)	97
char[]		384,504 B (8.4%)	4,323 (4.1%)	8
java.util.TreeMap\$Entry		322,160 B (7%)	8,054 (7.6%)	3
java.util.Hashtable\$Entry[]		264,832 B (5.8%)	4,138 (3.9%)	97
java.util.Vector		264,608 B (5.8%)	8,269 (7.8%)	97
byte[]		256,176 B (5.6%)	1,324 (1.3%)	8
java.io.ObjectStreamClass\$WeakClassKey		208,256 B (4.5%)	6,508 (6.2%)	2
java.util.Hashtable		199,152 B (4.3%)	4,149 (3.9%)	97
java.security.ProtectionDomain		168,040 B (3.7%)	4,201 (4%)	97
int[]		153,320 B (3.3%)	557 (0.5%)	5
java.security.CodeSource		132,096 B (2.9%)	4,128 (3.9%)	97
javax.management.remote.rmi.RMIConnectionImp...		100,584 B (2.2%)	1,397 (1.3%)	97
javax.management.remote.rmi.RMIConnectionImp...		99,936 B (2.2%)	1,388 (1.3%)	97
com.sun.jmx.remote.util.OrderClassLoaders		99,360 B (2.2%)	1,380 (1.3%)	97
java.security.Principal[]		66,992 B (1.5%)	4,187 (4%)	97
java.util.HashSet		66,816 B (1.5%)	4,176 (4%)	97
java.security.ProtectionDomain\$Key		65,952 B (1.4%)	4,122 (3.9%)	97
java.lang.String		51,984 B (1.1%)	2,166 (2.1%)	5
java.util.TreeMap\$KeyIterator		43,840 B (1%)	1,370 (1.3%)	2
java.util.TreeMap		43,248 B (0.9%)	901 (0.9%)	3

- 查询桥接词

类名 - 活动的分配对象	活... ▼	活动字节	活动对象	年代数
java.lang.Object[]		867,496 B (16.8%)	19,039 (16.1%)	106
java.util.HashMap		454,224 B (8.8%)	9,463 (8%)	106
char[]		447,056 B (8.7%)	5,041 (4.3%)	8
java.util.TreeMap\$Entry		356,640 B (6.9%)	8,916 (7.5%)	3
byte[]		293,592 B (5.7%)	1,527 (1.3%)	8
java.util.Vector		286,496 B (5.6%)	8,953 (7.6%)	106
java.util.Hashtable\$Entry[]		285,976 B (5.6%)	4,469 (3.8%)	106
java.io.ObjectStreamClass\$WeakClassKey		243,104 B (4.7%)	7,597 (6.4%)	2
java.util.Hashtable		214,656 B (4.2%)	4,472 (3.8%)	106
java.security.ProtectionDomain		181,520 B (3.5%)	4,538 (3.8%)	106
int[]		164,224 B (3.2%)	641 (0.5%)	5
java.security.CodeSource		142,752 B (2.8%)	4,461 (3.8%)	106
javax.management.remote.rmi.RMIConnectionImp...		108,936 B (2.1%)	1,513 (1.3%)	106
javax.management.remote.rmi.RMIConnectionImp...		108,216 B (2.1%)	1,503 (1.3%)	106
com.sun.jmx.remote.util.OrderClassLoaders		107,064 B (2.1%)	1,487 (1.3%)	106
java.util.HashSet		72,976 B (1.4%)	4,561 (3.9%)	106
java.security.Principal[]		72,288 B (1.4%)	4,518 (3.8%)	106
java.security.ProtectionDomain\$Key		71,248 B (1.4%)	4,453 (3.8%)	106
java.lang.String		59,976 B (1.2%)	2,499 (2.1%)	5
java.util.TreeMap\$KeyIterator		51,168 B (1%)	1,599 (1.4%)	2
java.io.ObjectStreamClass		50,128 B (1%)	482 (0.4%)	2

c) 根据 bridge word 生成文本

类名 - 活动的分配对象	活... ▼	活动字节	活动对象	年代数
java.lang.Object[]		730,568 B (17.1%)	14,981 (15.1%)	107
java.util.HashMap		446,400 B (10.4%)	9,300 (9.4%)	107
java.util.Hashtable\$Entry[]		286,944 B (6.7%)	4,485 (4.5%)	107
java.util.Vector		286,944 B (6.7%)	8,967 (9%)	107
char[]		281,216 B (6.6%)	3,132 (3.2%)	8
java.util.TreeMap\$Entry		253,920 B (5.9%)	6,348 (6.4%)	3
java.util.Hashtable		215,088 B (5%)	4,481 (4.5%)	107
byte[]		186,928 B (4.4%)	944 (1%)	8
java.security.ProtectionDomain		181,840 B (4.3%)	4,546 (4.6%)	107
java.security.CodeSource		143,136 B (3.3%)	4,473 (4.5%)	107
java.io.ObjectStreamClass\$WeakClassKey		142,176 B (3.3%)	4,443 (4.5%)	2
int[]		112,320 B (2.6%)	400 (0.4%)	5
javax.management.remote.rmi.RMIConnectionImp...		109,152 B (2.6%)	1,516 (1.5%)	107
javax.management.remote.rmi.RMIConnectionImp...		108,432 B (2.5%)	1,506 (1.5%)	107
com.sun.jmx.remote.util.OrderClassLoaders		107,208 B (2.5%)	1,489 (1.5%)	107
java.security.Principal[]		72,352 B (1.7%)	4,522 (4.6%)	107
java.util.HashSet		72,240 B (1.7%)	4,515 (4.6%)	107
java.security.ProtectionDomain\$Key		71,264 B (1.7%)	4,454 (4.5%)	107
java.lang.String		37,944 B (0.9%)	1,581 (1.6%)	5
java.util.TreeMap		33,984 B (0.8%)	708 (0.7%)	3
java.io.ObjectStreamClass		30,784 B (0.7%)	296 (0.3%)	2

d) 计算两个单词的最短路径

类名 - 活动的分配对象	活...	活动字节	活动对象	年代数
<b>char[]</b>		124,592 B (18.7%)	1,437 (9.4%)	3
<b>java.lang.Object[]</b>		91,000 B (13.6%)	3,140 (20.6%)	3
<b>java.util.TreeMap\$Entry</b>		77,320 B (11.6%)	1,933 (12.7%)	1
<b>java.io.ObjectStreamClass\$WeakClassKey</b>		76,448 B (11.5%)	2,389 (15.7%)	1
<b>byte[]</b>		73,288 B (11%)	431 (2.8%)	1
<b>int[]</b>		43,032 B (6.4%)	181 (1.2%)	2
<b>java.lang.String</b>		17,208 B (2.6%)	717 (4.7%)	3
<b>java.util.TreeMap\$KeyIterator</b>		15,552 B (2.3%)	486 (3.2%)	1
<b>java.io.ObjectStreamClass</b>		13,728 B (2.1%)	132 (0.9%)	1
<b>java.util.TreeMap</b>		10,416 B (1.6%)	217 (1.4%)	1
<b>java.lang.StringBuilder</b>		9,072 B (1.4%)	378 (2.5%)	1
<b>java.util.HashMap</b>		8,592 B (1.3%)	179 (1.2%)	4
<b>java.util.TreeMap\$EntryIterator</b>		8,224 B (1.2%)	257 (1.7%)	1
<b>java.io.SerialCallbackContext</b>		7,104 B (1.1%)	296 (1.9%)	1
<b>java.util.Collections\$UnmodifiableCollecti...</b>		6,264 B (0.9%)	261 (1.7%)	1
<b>java.lang.Long</b>		5,664 B (0.8%)	236 (1.6%)	1
<b>javax.management.openmbean.CompositeDataSupport</b>		5,184 B (0.8%)	216 (1.4%)	1
<b>java.util.HashMap\$Node[]</b>		4,144 B (0.6%)	74 (0.5%)	2
<b>java.util.TreeMap\$EntrySet</b>		3,488 B (0.5%)	218 (1.4%)	1
<b>java.util.TreeMap\$KeySet</b>		3,472 B (0.5%)	217 (1.4%)	1
<b>java.util.HashMap\$Node</b>		3,456 B (0.5%)	108 (0.7%)	2

e) 一个单词到其他单词的最短路径

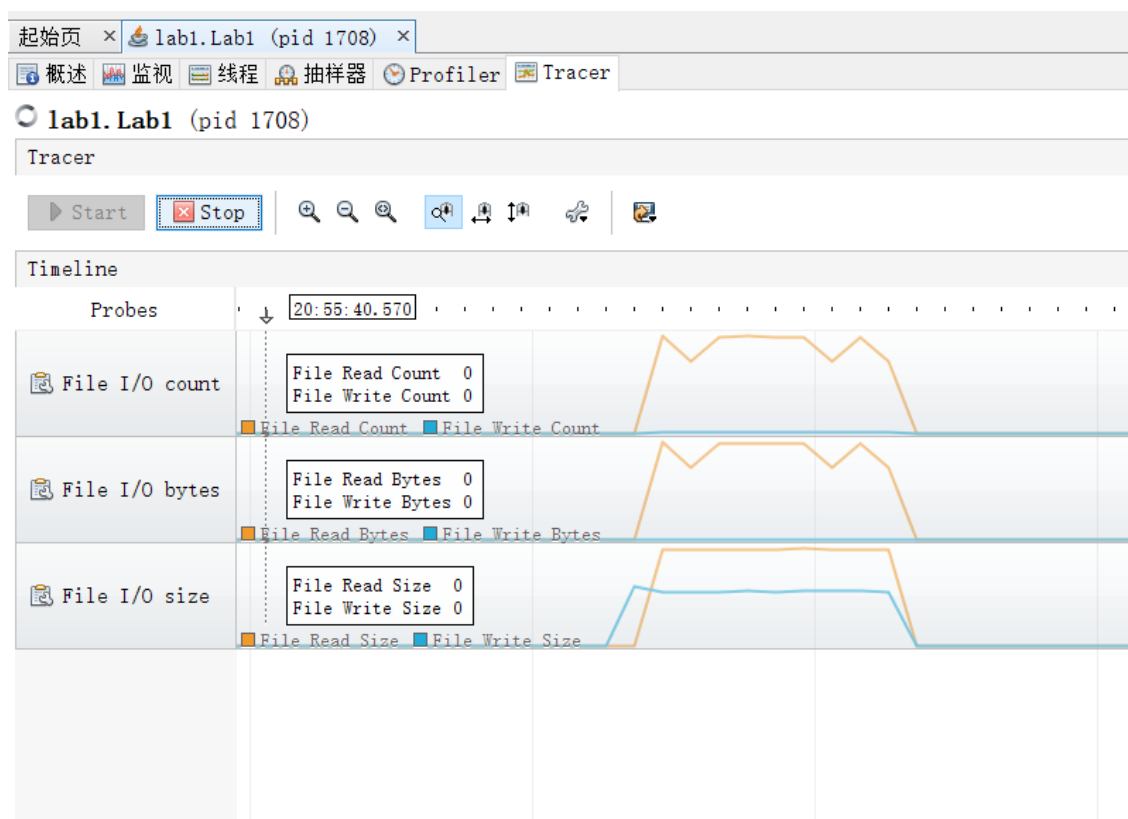
类名 - 活动的分配对象	...	活动字节	活动对象	年代数
<b>byte[]</b>		6,715,880 B (98.3%)	270 (8.7%)	1
<b>char[]</b>		20,440 B (0.3%)	262 (8.5%)	4
<b>java.lang.Object[]</b>		17,680 B (0.3%)	460 (14.9%)	15
<b>java.util.TreeMap\$Entry</b>		9,840 B (0.1%)	246 (7.9%)	1
<b>java.io.ObjectStreamClass\$WeakClassKey</b>		9,440 B (0.1%)	295 (9.5%)	1
<b>int[]</b>		6,504 B (0.1%)	26 (0.8%)	4
<b>java.util.HashMap</b>		4,224 B (0.1%)	88 (2.8%)	15
<b>java.lang.String</b>		3,432 B (0.1%)	143 (4.6%)	3
<b>java.util.Vector</b>		2,496 B (0%)	78 (2.5%)	17
<b>java.util.Hashtable\$Entry[]</b>		2,264 B (0%)	35 (1.1%)	11
<b>java.util.TreeMap\$KeyIterator</b>		1,920 B (0%)	60 (1.9%)	1
<b>java.lang.StringBuilder</b>		1,848 B (0%)	77 (2.5%)	1
<b>java.util.Hashtable</b>		1,776 B (0%)	37 (1.2%)	11
<b>java.io.ObjectStreamClass</b>		1,664 B (0%)	16 (0.5%)	1
<b>java.security.ProtectionDomain</b>		1,360 B (0%)	34 (1.1%)	8
<b>java.util.TreeMap</b>		1,296 B (0%)	27 (0.9%)	1
<b>java.security.CodeSource</b>		1,120 B (0%)	35 (1.1%)	8
<b>java.util.TreeMap\$EntryIterator</b>		1,056 B (0%)	33 (1.1%)	1
<b>java.util.HashMap\$Node</b>		1,056 B (0%)	33 (1.1%)	4
<b>javax.swing.plaf.metal.MetalScrollbar</b>		976 B (0%)	2 (0.1%)	2
<b>com.sun.jmx.remote.util.OrderClassLoaders</b>		936 B (0%)	13 (0.4%)	5

f) 随机游走

类名 - 活动的分配对象	... 活动字节	活动对象	年代数
<b>byte[]</b>	637,144 B (86.6%)	239 (9.7%)	1
<b>char[]</b>	17,088 B (2.3%)	195 (7.9%)	4
java.lang.Object[]	13,632 B (1.9%)	303 (12.4%)	16
java.util.TreeMap\$Entry	4,880 B (0.7%)	122 (5%)	1
java.io.ObjectStreamClass\$WeakClassKey	4,864 B (0.7%)	152 (6.2%)	1
<b>int[]</b>	4,824 B (0.7%)	17 (0.7%)	3
java.util.HashMap	4,752 B (0.6%)	99 (4%)	16
java.util.Vector	3,072 B (0.4%)	96 (3.9%)	18
java.util.Hashtable\$Entry[]	2,840 B (0.4%)	44 (1.8%)	12
java.lang.String	2,688 B (0.4%)	112 (4.6%)	3
java.util.Hashtable	2,304 B (0.3%)	48 (2%)	12
java.security.AccessControlContext	1,640 B (0.2%)	41 (1.7%)	5
java.security.ProtectionDomain	1,640 B (0.2%)	41 (1.7%)	9
java.lang.StringBuilder	1,464 B (0.2%)	61 (2.5%)	1
java.security.CodeSource	1,408 B (0.2%)	44 (1.8%)	9
com.sun.jmx.remote.util.OrderClassLoaders	1,152 B (0.2%)	16 (0.7%)	6
javax.management.remote.rmi.RMIConnectio...	1,008 B (0.1%)	14 (0.6%)	5
javax.swing.plaf.metal.MetalScrollbar	976 B (0.1%)	2 (0.1%)	2
java.util.HashMap\$Node	960 B (0.1%)	30 (1.2%)	3
java.io.ObjectStreamClass	936 B (0.1%)	9 (0.4%)	1
java.util.TreeMap\$KeyIterator	928 B (0.1%)	29 (1.2%)	1

### 3. 测试结果分析

在六种查询中，空间消耗最大的是求解一个单词到其他单词的最短路径的查询。而消耗空间最大的对象为 byte[]。对程序代码进行分析，我们发现，byte[] 对象的产生实际上是由文件 IO 造成的。通过使用 VisualVM 的插件 “Tracer-IO Probes”，我们发现在执行该查询时，程序确实执行了大量的文件 IO 操作，如下图。



### 4. 改进方法

由于内存瓶颈在于一个单词到其他单词的最短路径的查询，而该查询的内存瓶颈在



于大量文件 IO 造成的，因此我们尝试减少了文件 IO 的次数，通过减少调用图像绘制的语句，理论上能大幅减少 IO 调用，从而减少内存的占用。

## 8.3 代码改进之后的执行时间统计结果

### 1. 重新运行 VisualVM，获取的 CPU 性能分析结果

热点 - 方法	自用时间 [%] ▼	自用时间	总时间	调用
lab1.Lab1.showDirectedGraph (lab1.Graph)		2,051 ms (53.2%)	2,054 ms	9
lab1.Lab1\$3.actionPerformed.		258 ms (6.7%)	533 ms	1
lab1.Lab1.calcShortestPath.		84.2 ms (2.2%)	514 ms	34
lab1.Lab1.randonWalk (lab...)		31.5 ms (0.8%)	252 ms	1
lab1.Lab1\$5.actionPerformed.		24.1 ms (0.6%)	486 ms	2
lab1.Graph.getMatrix ()		16.6 ms (0.4%)	21.0 ms	35
lab1.Lab1\$6.actionPerformed.		15.6 ms (0.4%)	15.7 ms	1
lab1.Lab1\$4.actionPerformed.		12.5 ms (0.3%)	235 ms	1
lab1.Lab1\$7.actionPerformed.		10.0 ms (0.3%)	10.0 ms	1
lab1.Lab1\$ButtonHandler4.actio		8.4 ms (0.2%)	481 ms	1
lab1.Lab1\$2.actionPerformed.		6.43 ms (0.2%)	256 ms	1
lab1.Word.judgeOfWord (St...		4.22 ms (0.1%)	4.22 ms	38,115
lab1.Lab1\$ButtonHandler1.actio		3.91 ms (0.1%)	4.10 ms	1
lab1.Lab1\$ButtonHandler5.actio		2.90 ms (0.1%)	255 ms	1
lab1.Word.graphInf ()		1.72 ms (0%)	1.71 ms	297
lab1.Lab1\$ButtonHandler3.actio		1.18 ms (0%)	259 ms	1
lab1.Graph.getWordIndex (...)		0.804 ms (0%)	0.803 ms	456
lab1.Graph.getCommand ()		0.747 ms (0%)	2.46 ms	9
lab1.Lab1.findMinCost (in...		0.679 ms (0%)	0.678 ms	1,122
lab1.Graph.colorClean ()		0.398 ms (0%)	0.692 ms	36
lab1.Lab1\$ButtonHandler2.actio		0.320 ms (0%)	0.529 ms	1
lab1.Word.colorCleaned ()		0.294 ms (0%)	0.293 ms	1,188

### 2. 改进分析

- 经过改进之后，`Lab1.showDirectGraph()`方法的 CPU 时间明显下降，调用次数也从 43 次降到了 9 次，自用时间从 16006ms 降到了 2051ms，效率有了明显的提升。自用时间下降了 87.2%，调用次数下降了 79.1%。

### 3. 进一步的改进

- 进一步的改进可以着眼于计算最短路算法的改进，实际上，求解最短路算法执行了 n 次，但求解单源最短路实际上只需要运行一次最短路算法，这样可以使 `Lab1.calcShortestPath()`方法的 CPU 时间大大下降。

## 8.4 代码改进之后的内存占用统计结果

### 1. 重新运行 VisualVM，单点到其他各点最短路查询时获取的内存性能分析结果

类名 - 活动的分配对象	活动字节 [%]	活动字节	活动对象	...
byte[]		706,264 B (53.9%)	546 (3.5%)	1
char[]		116,632 B (8.9%)	1,264 (8.1%)	1
java.lang.Object[]		78,232 B (6%)	2,749 (17.6%)	1
java.util.TreeMap\$Entry		69,480 B (5.3%)	1,737 (11.1%)	1
java.io.ObjectStreamClass\$WeakCla...		65,824 B (5%)	2,057 (13.2%)	1
int[]		44,640 B (3.4%)	153 (1%)	1
java.security.AccessControlContext		19,400 B (1.5%)	485 (3.1%)	1
java.util.ArrayList\$Itr		15,552 B (1.2%)	486 (3.1%)	1
java.lang.String		15,120 B (1.2%)	630 (4%)	1
java.util.TreeMap\$KeyIterator		13,760 B (1%)	430 (2.8%)	1
java.io.ObjectStreamClass		10,920 B (0.8%)	105 (0.7%)	1
java.util.TreeMap		9,312 B (0.7%)	194 (1.2%)	1
java.lang.StringBuilder		8,472 B (0.6%)	353 (2.3%)	1
java.util.TreeMap\$EntryIterator		7,136 B (0.5%)	223 (1.4%)	1
java.util.HashMap		6,432 B (0.5%)	134 (0.9%)	1
java.io.SerialCallbackContext		6,096 B (0.5%)	254 (1.6%)	1
java.util.Collections\$Unmodifiabl...		5,664 B (0.4%)	236 (1.5%)	1
java.lang.ref.WeakReference		4,896 B (0.4%)	153 (1%)	1
javax.management.openbean.Composit...		4,752 B (0.4%)	198 (1.3%)	1
javax.management.openbean.Composit...		4,648 B (0.4%)	243 (1.6%)	1
java.lang.Long		4,416 B (0.3%)	184 (1.2%)	1

## 2. 改进分析

- 改进之后，求单点到其他各点最短路所需的内存大小大大下降，从 6.715MB 下降到了 0.706MB，内存占用减少了 89.5%。

## 3. 进一步的改进

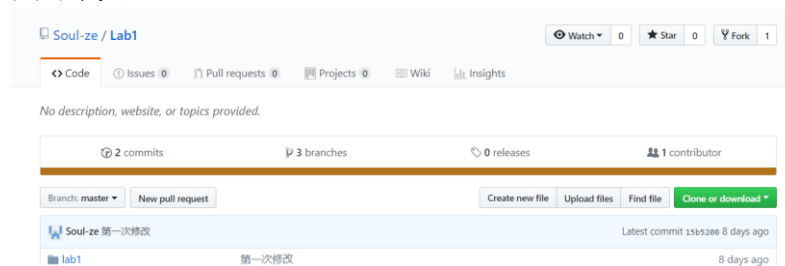
- 在 `Lab1.calcShortestPath()` 方法中的 `stack` 数组，可以按照点数的大小来初始化数组的大小，而不是每次都初始化为 100000，造成不必要的空间浪费。

# 9 利用 Git/GitHub 进行协作的过程

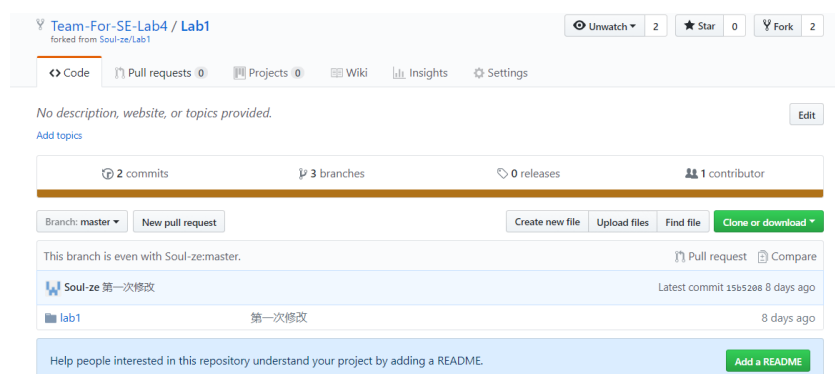
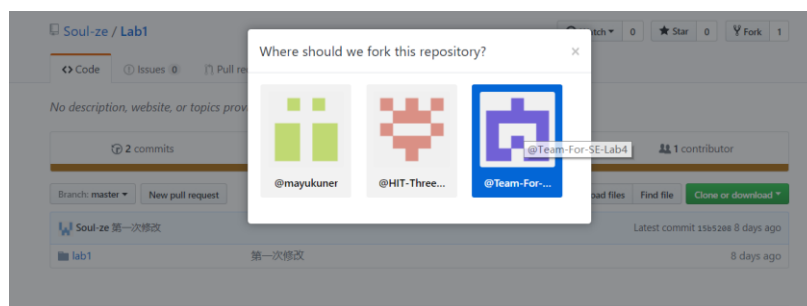
## Part 1:

### 1. 在 GitHub 上找到待评审代码，fork 至本组 GitHub 仓库内并重命名为 Lab4;

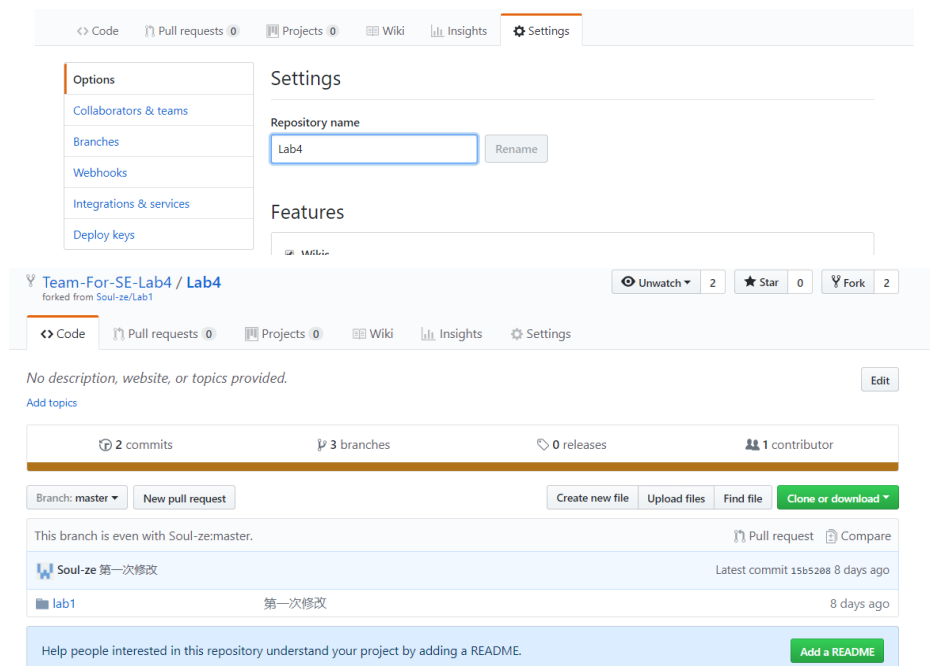
- 找到评审代码



- fork 至本组仓库内 (<https://github.com/orgs/Team-For-SE-Lab4/>)



### c) 重命名仓库为 Lab4



### 2. 将 fork 得到的仓库 Lab4 clone 至本组的本地仓库 Lab4;

```
mayuk@DESKTOP-3F16N34 MINGW64 /e/HITCS/Software Engineering/exp4 (master)
$ git clone git@github.com:Team-For-SE-Lab4/Lab4.git
Cloning into 'Lab4'...
Warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of known hosts.
remote: Counting objects: 146, done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 146 (delta 77), reused 146 (delta 77), pack-reused 0
Receiving objects: 100% (146/146), 5.19 MiB | 738.00 KiB/s, done.
Resolving deltas: 100% (77/77), done.
Checking connectivity... done.
```

### 3. 人工 review 和工具 review 之后将修改提交至本地仓库;



```

mayuk@DESKTOP-3F16N34 MINGW64 /e/HITCS/Software Engineering/exp4/Lab4 (Lab4)
$ git log
commit 2a7b88a3b285c578679554ada01bc93adbdb90fb
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 23:18:11 2017 +0800

    Fixed a potential problem

commit 1b6f960a2d75a1b119877bd95b85d2d831987647
Author: qioqio <32943497752qq.com>
Date:   Fri Oct 20 21:57:03 2017 +0800

    Fixed two potential bugs revealed by PMD

commit 805b0a563fd74931e2135df4721bbcd5daa016f3
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 21:28:05 2017 +0800

    Fixed two potential bugs revealed by FindBugs

commit 078b67fd952917c1bbdbcd17ffa2f339e09c7070
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 20:55:10 2017 +0800

    Reduced repainting times when calculating shortest paths from one single source

commit 2ec7c3c0e18834537433d05f3862299db4b194dc
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 20:26:21 2017 +0800

    Reduced repainting times when calculating shortest paths from one single source

commit 585b58877c635e1a76abbfa483eeb4193bfbfffa
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 20:21:53 2017 +0800

    added .gitignore

commit a709e3e5fcc9d1cccd694f61be734dae9fd5e9a
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 16:38:09 2017 +0800

    Fixed a bug in relative path

commit 7df0c240dd3f2aaf8abe14ce186d0db9f89712eb
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 16:17:17 2017 +0800

    Updated project

commit 3f58b874f4e58f611a1ceb484a2d66fab3637cc3
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 16:14:58 2017 +0800

    Updated .gitignore

commit c104553a328be67a0e18710806a760a58ebda78b
Author: Yukun Ma <gmayukun@gmail.com>
Date:   Fri Oct 20 14:53:23 2017 +0800

    fixed some problems using FindBugs, Checkstyle, PMD

```

4. 将所有修改历史 push 到 GitHub 上本组仓库 Lab4;

```

mayuk@DESKTOP-3F16N34 MINGW64 /e/HITCS/Software Engineering/exp4/Lab4 (Lab4)
$ git push origin Lab4
warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of known hosts.
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 472 bytes | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To git@github.com:Team-For-SE-Lab4/Lab4.git
   1b6f960..2a7b88a  Lab4 -> Lab4

```

The screenshot shows the GitHub repository page for 'Team-For-SE-Lab4 / Lab4', which is forked from 'Soul-ze/Lab1'. The repository has 2 watches, 0 stars, and 2 forks. The 'Code' tab is selected, showing a list of commits on Oct 20, 2017. The commits are as follows:

Commit Message	Author	Time Ago	Commit Hash
Fixed a potential problem	mayukuner	3 minutes ago	2a7b88a
Fixed two potential bugs revealed by PMD	qioqio	an hour ago	1b6f96b
Fixed two potential bugs revealed by FindBugs	mayukuner	2 hours ago	805b8a5
Reduced repainting times when calculating shortest paths from one sin...	mayukuner	2 hours ago	078b67f
Reduced repainting times when calculating shortest paths from one sin...	mayukuner	3 hours ago	2ec7c3c
added .gitignore	mayukuner	3 hours ago	585b588
Fixed a bug in relative path	mayukuner	7 hours ago	a709e3e
Updated project	mayukuner	7 hours ago	7df0c24
Updated .gitignore	mayukuner	7 hours ago	3f58b87
fixed some problems using FindBugs, Checkstyle, PMD	mayukuner	8 hours ago	c104553


5. 在 Github 上将最新的提交 Pull Request 到原作者的 Lab1 仓库;

The screenshot shows the GitHub Pull Request page for 'Soul-ze / Lab1'. The page is titled 'Comparing changes' and shows a comparison between the 'base fork: Soul-ze/Lab1' and the 'head fork: Team-For-SE-Lab4/Lab4'. The pull request is titled 'Create pull request' and has 9 commits, 44 files changed, 0 commit comments, and 2 contributors. The commits are as follows:


Commit Message	Author	Commit Hash
fixed some problems using FindBugs, Checkstyle, PMD	mayukuner	c104553
Updated .gitignore	mayukuner	3f58b87
Updated project	mayukuner	7df0c24
Fixed a bug in relative path	mayukuner	a709e3e
added .gitignore	mayukuner	585b588
Reduced repainting times when calculating shortest paths from one sin...	mayukuner	2ec7c3c
Reduced repainting times when calculating shortest paths from one sin...	mayukuner	078b67f
Fixed two potential bugs revealed by FindBugs	mayukuner	805b8a5
Fixed two potential bugs revealed by PMD	qioqio	1b6f96b

Showing 44 changed files with 1,071 additions and 1,197 deletions.


## Code Review by Zhang Ning and Ma Yukun #2











 mayukuner wants to merge 10 commits into `Soul-ze:Lab4` from `Team-For-SE-Lab4:Lab4`

Conversation 0 Commits 10 Files changed 44

 mayukuner commented a minute ago

1. 调用dot的命令中的绝对路径修改为项目相对路径
2. 修正代码风格为Sun Java标准
3. 修正了若干可能会导致程序在不同环境下执行结果产生差异的代码，例如依赖于默认区域设置的toLowerCase方法
4. 删除了若干对程序运行结果没有影响的语句，例如对同一变量的连续两条赋值语句
5. 提高了程序的时间和空间效率，在求从一点到其他各点最短路的过程中尤为明显，此外还包括使用StringBuffer来替换String执行若干字符串的连续拼接操作

 mayukuner and others added some commits 9 hours ago

-  fixed some problems using FindBugs, Checkstyle, PMD c104553
-  Updated .gitignore 3f58b87
-  Updated project 7df0c24
-  Fixed a bug in relative path a709e3e
-  added .gitignore 585b588
-  Reduced repainting times when calculating shortest paths from one sin... 2ec7c3c
-  Reduced repainting times when calculating shortest paths from one sin... 078b67f
-  Fixed two potential bugs revealed by FindBugs 805b0a5
-  Fixed two potential bugs revealed by PMD 1b6f960
-  Fixed a potential problem 2a7b88a

## Part 2:

1. 原作者在自己的本地 Lab1 仓库建立新分支 Lab4;

```
PS E:\HITCS\Software Engineering\exp3\wordgraph> git branch Lab4
PS E:\HITCS\Software Engineering\exp3\wordgraph> git push origin Lab4
Warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of known hosts.
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:mayukuner/Lab1.git
 * [new branch]      Lab4 -> Lab4
PS E:\HITCS\Software Engineering\exp3\wordgraph> a.
```

2. 原作者在 GitHub 上查看评审组的 pull request 并合并至 Lab4, 将评审组 pull request 过来的代码合并进去, 将 Lab4 分支 fetch 到本地 Lab1 仓库

- a) 查看 pull request

mayukuner / Lab1


Unwatch 2 Star 0 Fork 1

Code Issues Pull requests 2 Projects 0 Wiki Insights Settings

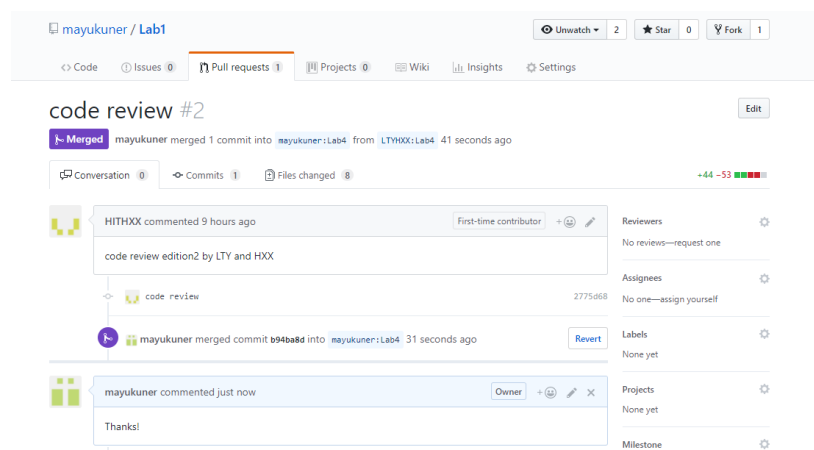
Label issues and pull requests for new contributors  
Now, GitHub will help potential first-time contributors discover issues labeled with `help wanted` or `good first issue`

Filters  Labels Milestones [New pull request](#)

2 Open 0 Closed Author Labels Projects Milestones Reviews Assignee Sort

 code review  
#2 opened 9 hours ago by HITXXX

- b) merge 评审组的 pull request



c) 将远程仓库的 Lab4 分支 fetch 到本地仓库

```
PS E:\eclipse-workspace\wordgraph> git fetch origin Lab4
Warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of known hosts.
From github.com:mayukuner/Lab1
 * branch          Lab4          -> FETCH_HEAD
PS E:\eclipse-workspace\wordgraph> git log
commit b94ba8d8835clffa995604a459ca23b52239a6ac
Merge: f475e30 2775d68
Author: Yukun Ma <gmayukun@gmail.com>
Date: Thu Oct 19 21:33:14 2017 +0800

Merge pull request #2 from LTYHXX/Lab4

code review
```

3. 原作者列出评审组所做的所有修改:

修改位置	修改原因	修改结果
PanelOpenFile.java:56	自赋值	取消自赋值
App.java:102	变量名大写开头	更改变量名
App.java:93	异常获取之后未处理	打印异常
App.java:64	子类未调用 super() 方法	构造函数前调用 super 方法
PanelApp.java	无用的导入包	删除无用导入语句
App.java	使用.*形式的导入	删除.*形式的导入
App.java:44	变量 G 应定义为 final 类型	将 G 定义为 final 类型
App.java:52-60	单行代码字符数过多	加入换行, 拆行
App.java:12,13,14,19, 20, 27	导入未使用的库	删除导入语句
App.java:49	Javadoc 标识词拼写错误	修正拼写
App.java:77	异常捕获之后无操作	加入打印异常信息语句

4. 原作者在 Lab4 分支上对不认可的修改进行调整, 并在 Lab4 分支上加以提交:

```

PS E:\eclipse-workspace\wordgraph> git add src/main/java
PS E:\eclipse-workspace\wordgraph> git status
On branch Lab4
Your branch is up-to-date with 'origin/Lab4'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   src/main/java/com/untitled/wordgraph/App.java
        modified:   src/main/java/com/untitled/wordgraph/PanelApp.java
        modified:   src/main/java/com/untitled/wordgraph/PanelNewText.java
        modified:   src/main/java/com/untitled/wordgraph/PanelOpenFile.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .eclipse-pmd
        .settings/org.eclipse.jdt.ui.preferences
        test.txt

PS E:\eclipse-workspace\wordgraph> git commit -m "Updated code review"
[Lab4 c38f8e6] Updated code review
 4 files changed, 5 deletions(-)

```

5. 原作者将 Lab4 分支与原来的主分支 merge 起来, 推送至本组 GitHub 的 Lab1 仓库。

```

PS E:\eclipse-workspace\wordgraph> git merge Lab4
Merge made by the 'recursive' strategy.
 src/main/java/com/untitled/wordgraph/App.java      45 ++++++
 src/main/java/com/untitled/wordgraph/PanelApp.java  25 +-----
 .../java/com/untitled/wordgraph/PanelOpenFile.java 14 +-----
 .../com/untitled/wordgraph/PanelQueryBridge.java   2 +-
 .../com/untitled/wordgraph/PanelRandomAccess.java  2 +-
 .../com/untitled/wordgraph/PanelSaveImage.java     2 +-
 .../com/untitled/wordgraph/PanelShortestPath.java  2 +-
 7 files changed, 39 insertions(+), 53 deletions(-)
PS E:\eclipse-workspace\wordgraph> git push origin master
Warning: Permanently added 'github.com,192.30.255.113' (RSA) to the list of known hosts.
Counting objects: 15, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.27 KiB | 0 bytes/s, done.
Total 15 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), completed with 5 local objects.
To git@github.com:mayukuner/Lab1.git
 f475e30..98df11f master -> master

```

mayukuner / Lab1

Unwatch 2 Star 0 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master

Commits on Oct 23, 2017

- Merge branch 'Lab4'
  - mayukuner committed a minute ago
  - 98df11f
- Updated code review
  - mayukuner committed 3 minutes ago
  - c38f8e6

Commits on Oct 20, 2017

- Merge branch 'master' of github.com:mayukuner/Lab1
  - mayukuner committed 3 days ago
  - e45f3e9

Commits on Oct 19, 2017

- Merge pull request #2 from LTYHXX/Lab4
  - mayukuner committed 4 days ago
  - b94ba8d
- code review
  - HITHXX committed 4 days ago
  - 2775d68

## 10 评述

### 10.1 对代码规范方面的评述

对你们组所拿到的其他同学的程序在遵循代码风格规范方面的水平做简要评论。

若代码风格较好，给你们本次实验的工作带来哪些方面的好处？

若代码风格较差，给你们本次实验的工作带来哪些方面的坏处？

1. 被评审组的同学的代码整体风格比较好,代码的缩进比较规范,
2. 但是使用了很多的 tab 键,这个在 sun 的规范里面是不允许的.
3. 此外代码的**变量命名**不够规范,有大量过短的变量名,难以理解大小写不区分的标识符
4. **注释**写的并不详细,有些地方本组成员难以理解.
5. GUI 构建的代码在同一方法内,稍过于冗长,对阅读和理解代码造成了些许的障碍。
6. 最后整个程序的诸多方法封装在了一个类内，封装性稍差。

### 10.2 对代码性能方面的评述

对你们组所拿到的其他同学的程序在代码性能方面的水平做简要评论。

1. CPU 占用
  - a) 评审程序在大部分工作时，速度十分令人满意
  - b) 在执行某些工作特别是执行最短路查询时，评审程序的性能稍差，在性能不太好的电脑上运行，程序会有比较明显的卡顿
  - c) 输入数据的规模较大时，初始化与绘图时会有明显卡顿
  - d) 总体来说，评审程序 CPU 性能良好，表现稳定
2. 内存占用
  - a) 评审程序内存占用不大，空间复杂度与输入数据的规模同阶
  - b) 评审程序的内存占用表现优秀，即使在输入规模较大的情况下，也没有消耗过多的内存

## 11 计划与实际进度

任务名称	计划时间长度（分钟）	实际耗费时间（分钟）	提前或延期的原因分析
配置四个工具	60	20	配置过程非常顺利,没有出现什么错误.
代 码 review	30	60	注释写的不够完整,代码不易理解
静态评审	120	400	静态评审需要熟悉工具的使用,掌握工具花费了不少的时间

Visualvm 评审	60	120	测试代码过程需要手动的完成,然后观看代码的执行时间和内存状态,所以很慢.
----------------	----	-----	--------------------------------------

## 12 小结

- FindBugs、PMD、CheckStyles 三者都是代码规范静态检查工具，它们之间有何异同？从分析结果看，它们有什么优劣？

- **CheckStyle** 检查的是代码是否和代码规范相符,主要是指注释,变量命名,import 语句,重复代码等问题,关注的并不是程序中的错误,关注的主要是代码是不是好看,漂亮,是不是容易让别人理解.
- 和 checkstyle 形成鲜明对比的是 **findbugs**, 虽然它检查出的错误会有一些重合的部分,但是和另外的两个工具相比, 它注重检测真正的 bug 及潜在的性能问题,它检查的是 class 文件而不是源代码,所以需要先进行编译,生成 class 文件,它可以帮我检查代码的性能,有没有合理的利用内存资源,有没有释放内存等,所以 findbugs 的错误是需要引起开发者重点注意的.
- 还有 **pmd**, 它即可以查出一些格式上的错误,同时还会给出一些潜在的 bug,是比较全面的检查工具,下面是举的一些例子:
  - ◆ 潜在的 bug: 空的 try/catch/finally/switch 语句
  - ◆ 未使用的代码: 未使用的局部变量、参数、私有方法等
  - ◆ 可选的代码: String/StringBuffer 的滥用
  - ◆ 复杂的表达式: 不必须的 if 语句、可以使用 while 循环完成的 for 循环
  - ◆ 重复的代码: 拷贝/粘贴代码意味着拷贝/粘贴 bugs

某种意义上,它找出来的并不是错误,而是一些可能会引致错误的地方,这就导致它会查出很多的警告来,但是它会把这些警告编写一个危险等级来提醒开发者哪些需要额外的注意.

- 我觉得三者没有什么优劣之分,有的只是能不能满足程序员的需求,findbugs 注重会导致程序出现错误的真正 bugs,checkstyle 帮助用户更好的检查自己的代码风格是不是符合规范,但是有的时候检查的有些求全责备了.pmd 则表现的全面一些,给出一些格式上的错误,不如 checkstyle 给的全面,和一些潜在的 bugs,pmd 还将错误给分出等级来,这样的话,用户可以有针对性快速检查自己的代码.
- 从配置/使用的简便性、代码规范的可扩展性/可自定义性等角度对 FindBugs、PMD、CheckStyle 进行对比;
 

首先三者的配置和使用都不复杂.

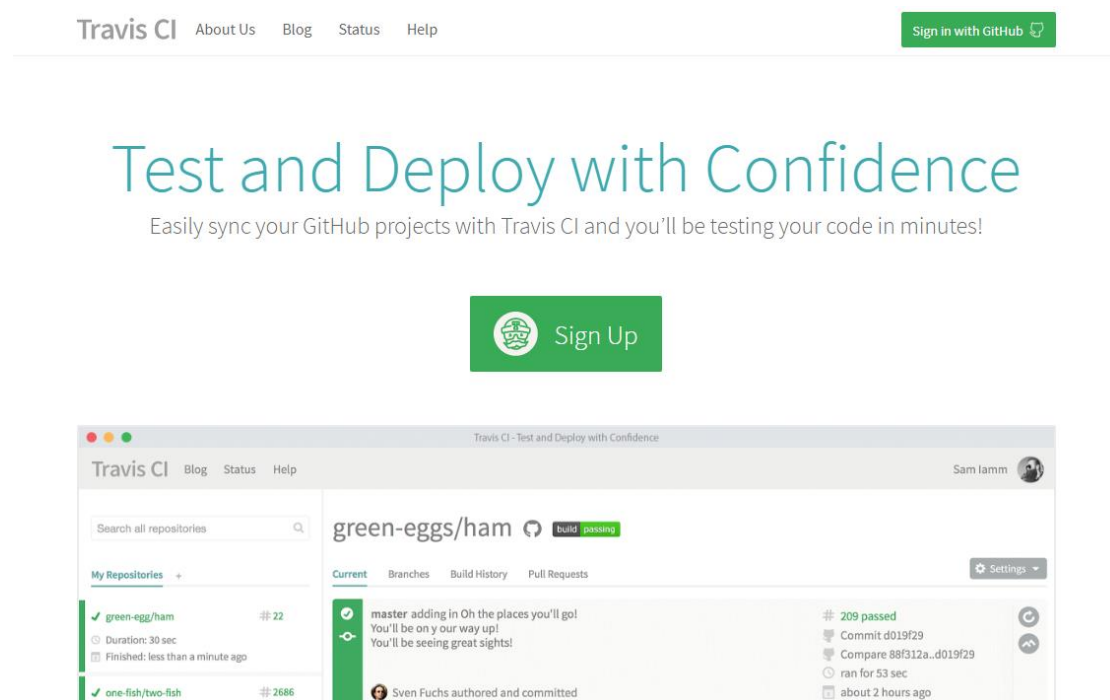
    - Checkstyle 提供了高可配置性, 以便适用于各种代码规范, 所以除了使用它提供的几种常见标准之外, 你也可以定制自己的标准。
    - PMD 附带了许多可以直接使用的规则, 利用这些规则可以找出 Java 源程序的许多问题,用户还可以自己定义规则, 检查 Java 代码是否符合某些特定的编码规范。用户可以通过编写 Java 代码并重新编译 PDM, 或者更简单些, 编写 XPath 表达式, 来添加新的规则.

- Findbugs 自带了很多的检查器 60 余种 Bad practice, 80 余种 Correntness, 1 种 Internationalization, 12 种 Malicious code vulnerability, 27 种 Multithreaded correntness, 23 种 Performance, 43 种 Dodgy,于此同时用户也可以自己来编写自定义检查器,网上有不少介绍如何自定义检查器的文章.
3. VisualVM 如何帮助提升代码的运行时性能?
- VisualVM 可以在程序运行的同时, **动态的监听** CPU 使用情况、内存占用情况等程序性能指标, 然后通过**可视化界面**将其展现出来。
  - 通过对这些结果的分析,我们可以更加**准确**的知晓代码运行时的性能究竟如何,从而有效地分析代码运行时**性能的短板**,并作出有**针对性的修改**,从而提高代码整体的**效率**。
4. “代码是否符合编码规范”与“代码执行的时空复杂性”是否有直接的联系? 为什么?
- 我觉得没有直接的联系,就算我的变量命名并不够规范,我们的程序也一样可以顺利的运行,而且时空复杂性和这个没有半毛钱关系.
  - 相反,就算我们的程序代码很符合规范,我们程序的时空复杂度也不会因此而降低.
  - 综上所述,所以我认为二者没有直接的联系。但是对于一个好的程序,这两个东西必须同时兼备,绝对不能抛弃任何一个,只有符合代码规范,并且时空复杂度比较低的代码才是好的代码,如果二者存在冲突的时候,我们就要进行一番权衡,然后做出对程序的开发最有益的方案来。
5. 对软件代码优化方面的其他体会。
- 总体来说,测试工具设计的挺合理的,确实大大降低了程序员的工作难度,将代码评审这个工作的难度大大降低了。
  - Checkstyle 能够将代码形式上的错误找出来,但是它并没有将代码给修改过来,为什么就不能再进一步,将代码给修改过来呢?在这个方面 checkstyle 做的并不好。
  - 个人认为程序代码优化总是亡羊补牢,其实在程序代码优化的过程之中浪费了很多的时间和精力,我们真正应该做的是在写代码的阶段,就编写符合代码规范的代码,写完代码之后对代码进行充分的测试,要把 bug 消灭在萌芽阶段,敏捷开发就可以比较好的满足这些需求,敏捷开发的测试数据在编写程序之前就已经完成了,所以程序写完,立即测试,而且由于结对编程,使得两个人可以更好的在编程过程之中,发现 bug,将 bug 导致的损耗降到最低,同时两人同时工作还可以迫使程序员把代码变得工整,漂亮,易读。
  - 虽然咱们现在是在学习代码的评审与性能优化,但是对于代码的优化和改良最有效的工具不是咱们使用的那四个,反而是之前就已经学习过的敏捷开发,尤其是结对编程。

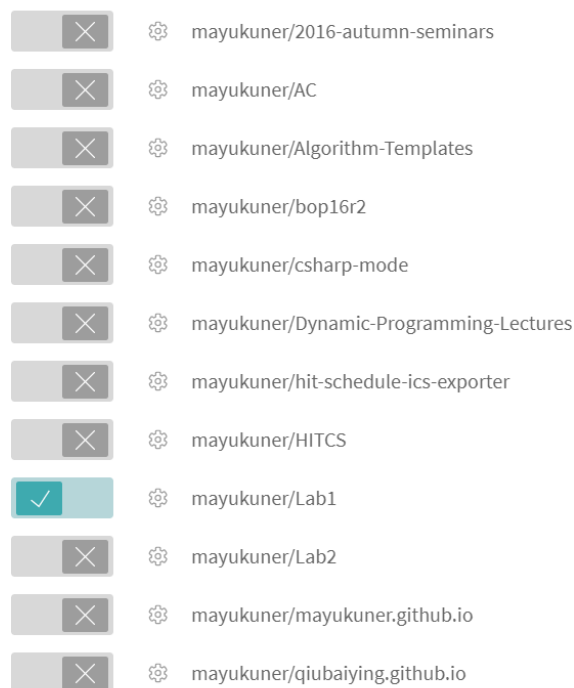
## 13 Travis CI 的配置

1. 首先登陆网站 <https://travis-ci.org/>, 点击右上角的 Sign in with GitHub 按钮, 输入自己的 github 账号和密码, 并允许 Travis CI 的认证。





2. 然后访问 Travis CI 的 profile，选择相应的 repository 打开 Service Hook 开关。



3. 给 repository 配置 .travis.yml 文件。该文件需要放置在 repository 的根目录下。 .travis.yml 的具体内容如下:

```
language: java
jdk:
  - openjdk8
```

4. 在 repository 的 README.md 文件中加入 build 状态图标。只需在 README.md 文件中加入以下一行 markdown 代码:

```
[![BuildStatus](https://travis-ci.org/[YOUR_GITHUB_USERNAME]/[YOUR_PROJEC
```

T\_NAME].png)](https://travis-ci.org/[YOUR\_GITHUB\_USERNAME]/[YOUR\_PROJECT\_NAME])

5. 登录 Travis 官网查看 log，看到[INFO] BUILD SUCCESS，说明构建成功。

```

1187 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
1188 Running com.untitled.wordgraph.AppTest
1189 Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.011 sec
1190
1191 Results :
1192
1193 Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
1194
1195 [INFO] -----
1196 [INFO] BUILD SUCCESS
1197 [INFO] -----
1198 [INFO] Total time: 2.569 s
1199 [INFO] Finished at: 2017-10-23T14:32:42Z
1200 [INFO] Final Memory: 16M/619M
1201 [INFO] -----
1202
1203
1204 The command "mvn test -B" exited with 0.
1205
1206 Done. Your build exited with 0.

```

6. 登录 repo 的 github 网址，可以看到下方的绿色图标

<https://github.com/mayukuner/Lab1>

9 commits   4 branches   0 releases   2 contributors

Branch: master   New pull request   Create new file   Upload files   Find file   Clone or download

mayukuner Configured Travis CI for this Repo   Latest commit bd21fe3 2 minutes ago

File	Commit Message	Time
.settings	initialize	8 days ago
example	initialize	8 days ago
lib	initialize	8 days ago
resources	initialize	8 days ago
src	Updated code review	16 minutes ago
.classpath	initialize	8 days ago
.gitignore	initialize	8 days ago
.project	initialize	8 days ago
.travis.yml	Configured Travis CI for this Repo	2 minutes ago
README.md	Configured Travis CI for this Repo	2 minutes ago
pom.xml	initialize	8 days ago

README.md

## Lab1

build passing