

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 6: 测试用例设计与 jUnit 单元测试

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 中配置测试环境.....	1
2.1	jUnit	1
2.2	EclEmma	3
2.3	Infinittest	5
3	针对 Lab1 的黑盒测试	5
3.1	所选的被测函数及其需求规约.....	5
3.2	等价类划分结果.....	6
3.3	测试用例设计.....	6
3.4	jUnit 测试代码	7
3.5	jUnit 单元测试结果.....	9
3.6	EclEmma 代码覆盖度分析	10
3.7	未通过测试的原因分析及代码修改.....	10
3.8	Git/GitHub 操作	11
4	针对 Lab1 的白盒测试	12
4.1	所选的被测函数.....	12
4.2	程序流程图.....	13
4.3	控制流图.....	14
4.4	圈复杂度计算与基本路径识别	14
4.5	测试用例设计.....	15
4.6	jUnit 测试代码	15
4.7	jUnit 单元测试结果.....	18
4.8	EclEmma 代码覆盖度分析	19
4.9	未通过测试的原因分析及代码修改.....	19
4.10	Git/GitHub 操作	19
5	Infinittest 持续测试	20
5.1	代码修改前.....	20
5.2	代码修改后.....	21
6	计划与实际进度.....	21
7	Travis CI 的配置.....	22
8	小结.....	24

1 实验要求

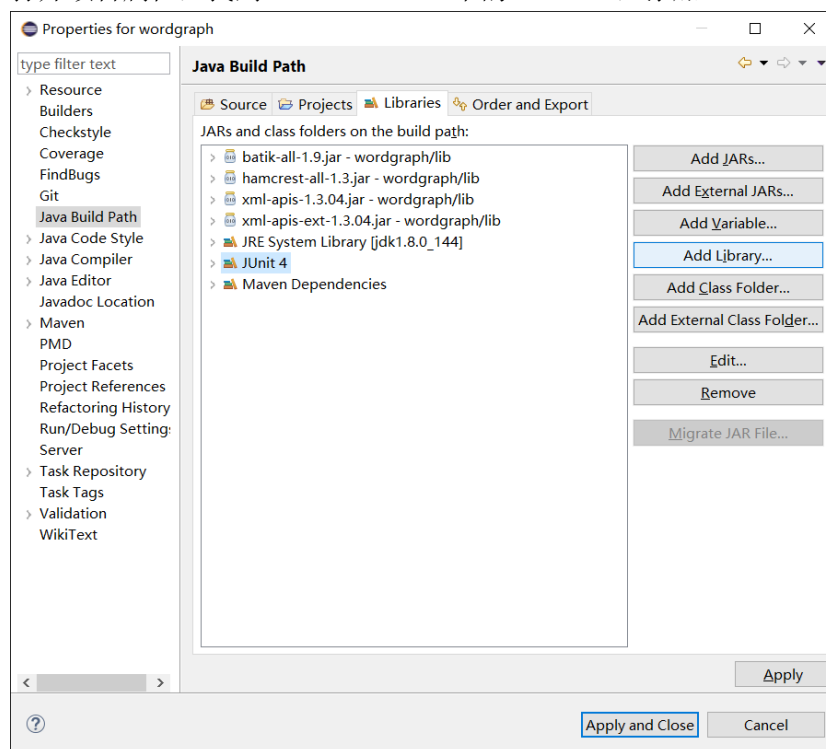
1. 按照在实验 1 中的分组，共同完成本次实验，针对本组在 Lab4 之后 git 仓库中的最新版本，展开本次实验；
2. 针对 Lab4 评审和优化过的程序，设计白盒测试用例；
3. 针对 Lab1 中包含的需求，设计黑盒测试用例；
4. 在 jUnit 环境下撰写测试代码并执行测试；
5. 使用 Infinitest 进行持续测试；
6. 使用 Eclemma 统计测试的覆盖度；
7. 如果让自己的 GitHub 项目具备持续集成的能力(Travis CI)还可以作为加分项。

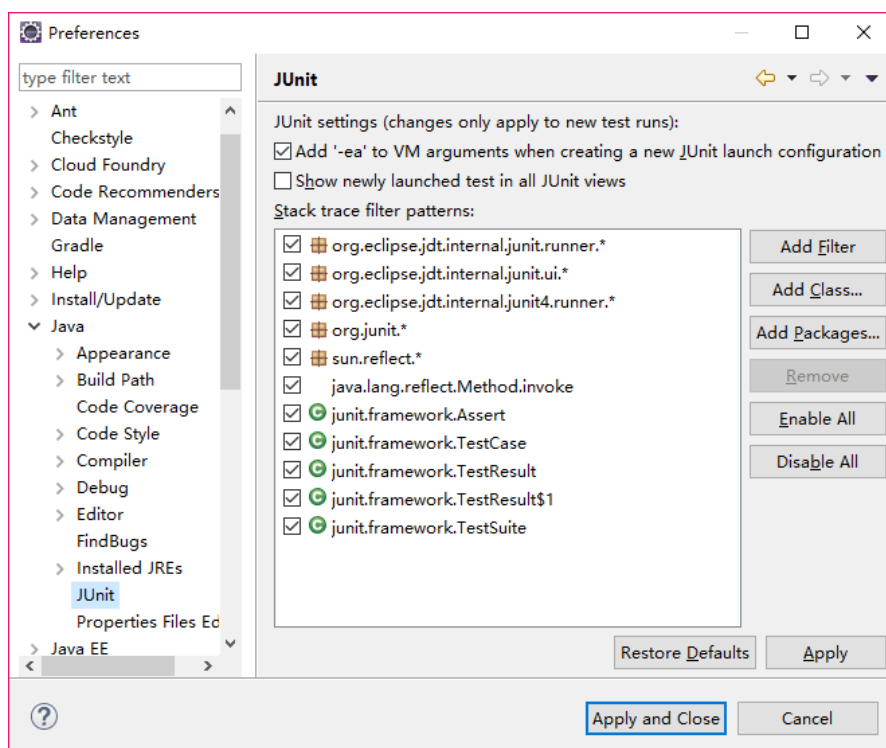
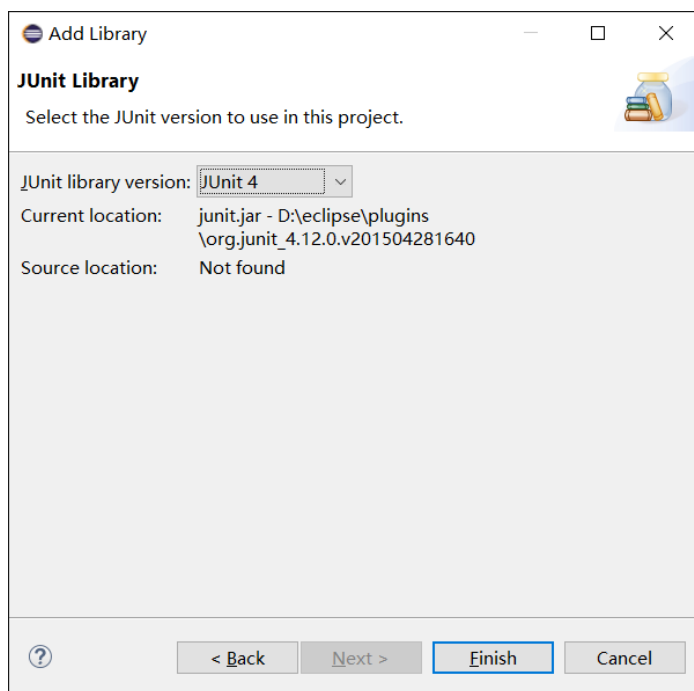
2 在 Eclipse 中配置测试环境

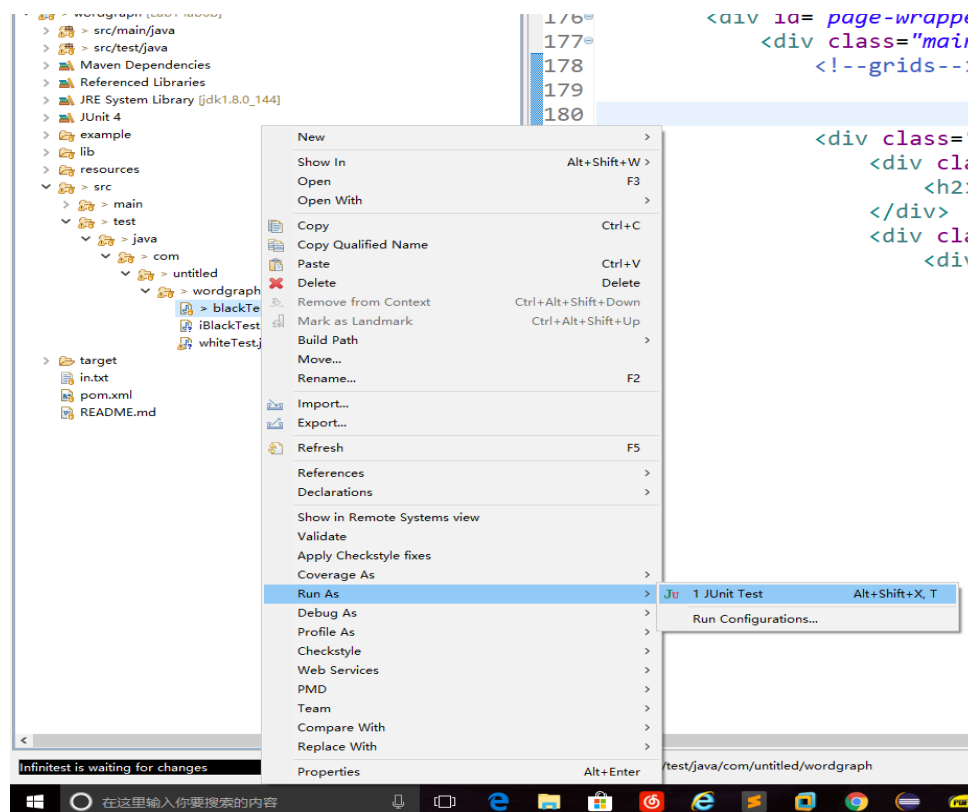
简要描述你在自己的 Eclipse 中配置 jUnit4、Eclemma、Infinitest 的过程。

2.1 jUnit

打开项目属性，找到 Java Build Path 中的 Libraries，添加 JUnit 4。

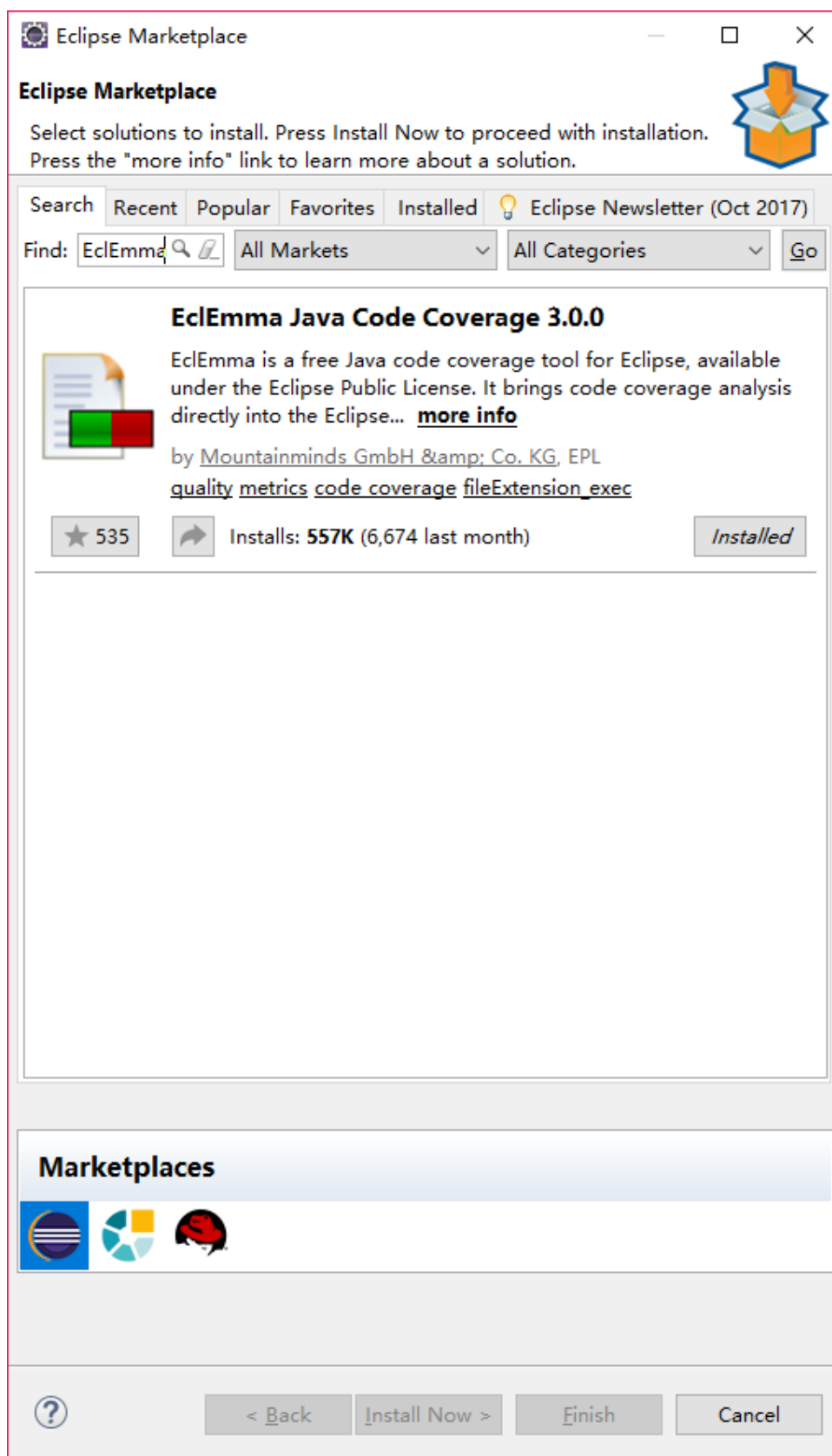






2.2 Eclemma

我从 eclipse 里面的 marketplace 里面下载了这个插件.



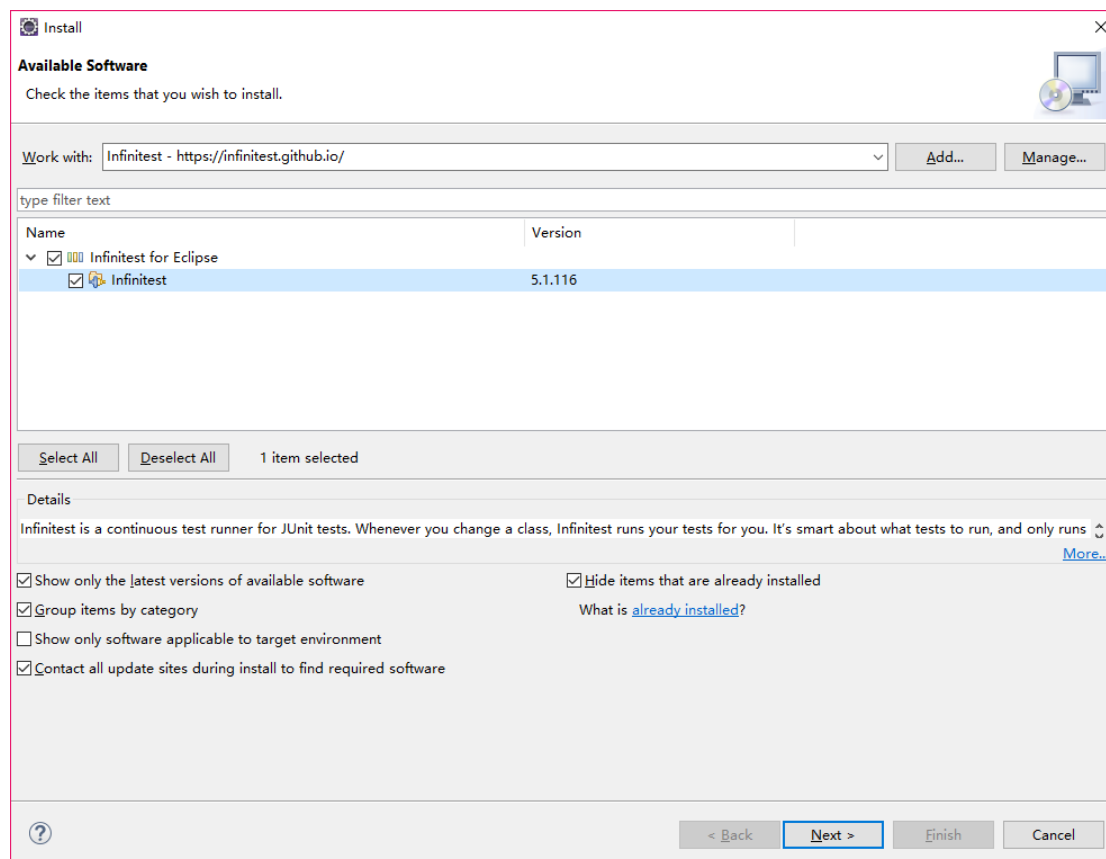
重启 Eclipse 后，EclEmma 即可使用。

2.3 Infinitest

使用 eclipse 的安装工具进行安装

Help->install new software

输入对应的下载网址, next->finish, 重启 eclipse。



3 针对 Lab1 的黑盒测试

3.1 所选的被测函数及其需求规约

只需要给出名字即可, 无需给出代码。

给出该函数所应满足的需求规约 (对应于你在 Lab1 里所覆盖的基本需求和扩展需求)。

注: 为了保证各组学生工作量的公平性, Lab1 中的扩展需求可以不需要在本次实验中进行覆盖。如果本次实验里覆盖了扩展需求, 将体现为本次实验的附加分数。

方法原型: `String calcShortestPath(type G, String word1, String word2)`

方法说明: 用户输入两个单词, 程序计算它们之间在图中的最短路径, 以某种突出的方式将路径标注在原图并展示在屏幕上, 同时展示路径的长度。例如输入 to 和 and, 则其最短路径为 to->explore->strange->new->life->and。如果有多条最短路径, 只需要展示一条即可。

如果输入的两个单词不可达，则给出提示。

3.2 等价类划分结果

约束条件说明	有效等价类及其编号	无效等价类及其编号
如果有多条最短路径，只需要展示一条即可	用户输入的两个单词之间有多条最短路径 (1)	
用户输入两个单词，程序计算它们之间在图中的最短路径，同时展示路径的长度	输入和 word1 和 word2 存在最短路径 (2)	
用户输入的两个单词不可达，则给出提示	输入的 word1 或 word2 如果不在图中出现 Word1 和 word2 均不出现 如果 word1 和 word2 不存在路径 (3)(4)(5)(6)	

3.3 测试用例设计

测试用例编号	输入字符串	期望输出字符串	所覆盖的等价类编号
1.	I music	"i" 到 "music" 有 2 条最短路径，其长度为 2。其中字典序第 1 小的最短路径为:i->and->music	(1)
2.	Make become	"make" 到 "become" 有 2 条最短路径，其长度为 2。其中字典序第 1 小的最短路径为:make->heat->become	(2)
3.	I haat	No \"haat\" in the graph!	(3)
4.	It haat	No \"it\" and \"haat\" in the graph!	(4)
5.	tyfbuhdsnj. i	No paths from \"tyfbuhdsnj\" to \"i\" in the graph!	(5)
6.	Jefiwjoif i	No \"Jefiwjoif\" in the graph!	(6)

3.4 jUnit 测试代码

针对 3.3 中的每一个测试用例，把其测试代码粘贴如下，代码必须是完整的。

测试用例编号	jUnit 测试代码
1.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelShortestPath PanelShortestPath = new PanelShortestPath(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("\i\" 到 \"music\" 有 2 条最短路，其长度为2。其中字典序第1小的最短路为:i->and->music",PanelShortestPath.calcShortestPath(G,"i","music")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
2.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelShortestPath PanelShortestPath = new PanelShortestPath(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("\make\" 到 \"become\" 有 2 条最短路，其长度为2。其中字典序第1小的最短路为:make->heat->become",PanelShortestPath.calcShortestPath(G,"make","become")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>

3.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelShortestPath PanelShortestPath = new PanelShortestPath(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No \"haat\" in the graph!", PanelShortestPath.calcShortestPath(G, "i", "haat")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
4.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelShortestPath PanelShortestPath = new PanelShortestPath(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No \"it\" and \"haat\" in the graph!", PanelShortestPath.calcShortestPath(G, "it", "haat")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
5.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelShortestPath PanelShortestPath = new PanelShortestPath(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No paths from \"tyfbuhdsnj\" to \"i\" in the graph!", PanelShortestPath.calcShortestPath(G, "tyfbuhdsnj", "i")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>

6.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelShortestPath PanelShortestPath = new PanelShortestPath(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No \"Jefiwjoif\" in the graph!\",PanelShortestPath.calcShortestPath(G,\"Jefiwjoif\",\"i\"); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
----	--

3.5 JUnit 单元测试结果

测试用例编号	期望输出字符串	实际输出字符串	是否通过测试, 请给出屏幕截图
1.	"i" 到 "music" 有 2 条最短路, 其长度为 2。其中字典序第 1 小的最短路为:i->and->music	"i" 到 "music" 有 2 条最短路, 其长度为 2。其中字典序第 1 小的最短路为:i->and->music	
2.	"make" 到 "become" 有 2 条最短路, 其长度为 2。其中字典序第 1 小的最短路为:make->heat->become	"make" 到 "become" 有 2 条最短路, 其长度为 2。其中字典序第 1 小的最短路为:make->heat->become	
3.	No \"haat\" in the graph!	No \"haat\" in the graph!	
4.	No "it" and "haat" in the graph!	No "it" and "haat" in the graph!	
5.	No paths from \"tyfbuhdsnj\" to \"i\" in the graph!	No paths from \"tyfbuhdsnj\" to \"i\" in the graph!	
6.	No \"Jefiwjoif\" in the graph!	No \"Jefiwjoif\" in the graph!	

3.6 Eclemma 代码覆盖度分析

给出 Eclemma 的代码覆盖度分析报告的截图

PanelShortestPath

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
printKthShortestPath(WordGraph, String, String, int)		0%		0%	9 9	27 27	1 1
PanelShortestPath()		100%		n/a	0 1	0 42	0 1
dijkstra(WordGraph, String, String)		100%		100%	0 12	0 37	0 1
calcShortestPath(WordGraph, String, String)		100%		100%	0 6	0 15	0 1
getKthShortestPath(WordGraph, String, String, int)		100%		75%	2 5	0 12	0 1
countShortestPaths(WordNode, WordNode)		100%		100%	0 4	0 10	0 1
addComponent(Component, int, int, int, int, int, int, int, int)		100%		n/a	0 1	0 12	0 1
Total	222 of 998	77%	18 of 62	70%	11 38	27 155	1 7

```

public String calcShortestPath(WordGraph G, String word1, String word2) {
    if (!G.nodes.containsKey(word1) || !G.nodes.containsKey(word2)) {
        return "No \"" + word1 + "\" and \"" + word2 + "\" in the graph!";
    } else if (!G.nodes.containsKey(word1)) {
        return "No \"" + word1 + "\" in the graph!";
    } else if (!G.nodes.containsKey(word2)) {
        return "No \"" + word2 + "\" in the graph!";
    } else {
        int numberPaths = dijkstra(G, word1, word2);
        if (G.nodes.get(word2).distance == Integer.MAX_VALUE) {
            return "No paths from \"" + word1 + "\" to \"" + word2 + "\" in the graph!";
        }
        srcWord = word1;
        destWord = word2;
        currentKth = 1;
        ArrayList<String> pathWords = getKthShortestPath(G, word1, word2, 1);
        return String.format("\"%s\" 到 \"%s\" 有 %d 条最短路，其长度为%d。其中字典序第%d小的最短路为:%s", word1, word2, numberPaths,
            G.nodes.get(word2).distance, currentKth, String.join("-", pathWords));
        //return String.format("%s", String.join("-", pathWords));
    }
}

```

3.7 未通过测试的原因分析及代码修改

请简要分析自己的 Lab1 代码为何未通过 3.5 节表格中某些测试用例的原因，并通过修改代码消除此类不符合需求的 bug。必要时给出修改后的代码。

修改代码之后，请重新填写下表，尽可能保证所有测试用例都能通过测试。

测试用例编号	期望输出字符串	实际输出字符串	是否通过测试，请给出屏幕截图
1.			

3.8 Git/GitHub 操作

给出命令行界面截图：

- 在本组 Lab1 的 git 仓库里，建立新的 git 分支，命名为 lab6b (b 代表 black-box testing)；
- 将 lab6b 合并到 master 分支，并推送至 GitHub。

```
qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git checkout -b lab6b
Switched to a new branch 'lab6b'

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6b)
$ git add -A

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6b)
$ git commit -m "black test"
[lab6b d2b8251] black test
 3 files changed, 47 insertions(+), 8 deletions(-)
 create mode 100644 src/test/java/com/untitled/wordgraph/whiteTest1.java

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6b)
$ git add -A

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6b)
$ git commit -m "black test"
[lab6b 7c2aa15] black test
 1 file changed, 38 insertions(+)
 create mode 100644 src/test/java/com/untitled/wordgraph/blackTest1.java

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6b)
$ |
```

```
qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6b)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git merge lab6b
Updating 9188350..7c2aa15
Fast-forward
 .../java/com/untitled/wordgraph/blackTest.java      | 5 +--
 .../java/com/untitled/wordgraph/blackTest1.java     | 38 +++++
 .../java/com/untitled/wordgraph/whiteTest.java      | 5 +--
 .../java/com/untitled/wordgraph/whiteTest1.java     | 45 +++++
 4 files changed, 85 insertions(+), 8 deletions(-)
 create mode 100644 src/test/java/com/untitled/wordgraph/blackTest1.java
 create mode 100644 src/test/java/com/untitled/wordgraph/whiteTest1.java

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git push origin master
Counting objects: 20, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (20/20), 1.21 KiB | 311.00 KiB/s, done.
Total 20 (delta 7), reused 0 (delta 0)
remote: Resolving deltas: 100% (7/7), completed with 3 local objects.
|
```

4 针对 Lab1 的白盒测试

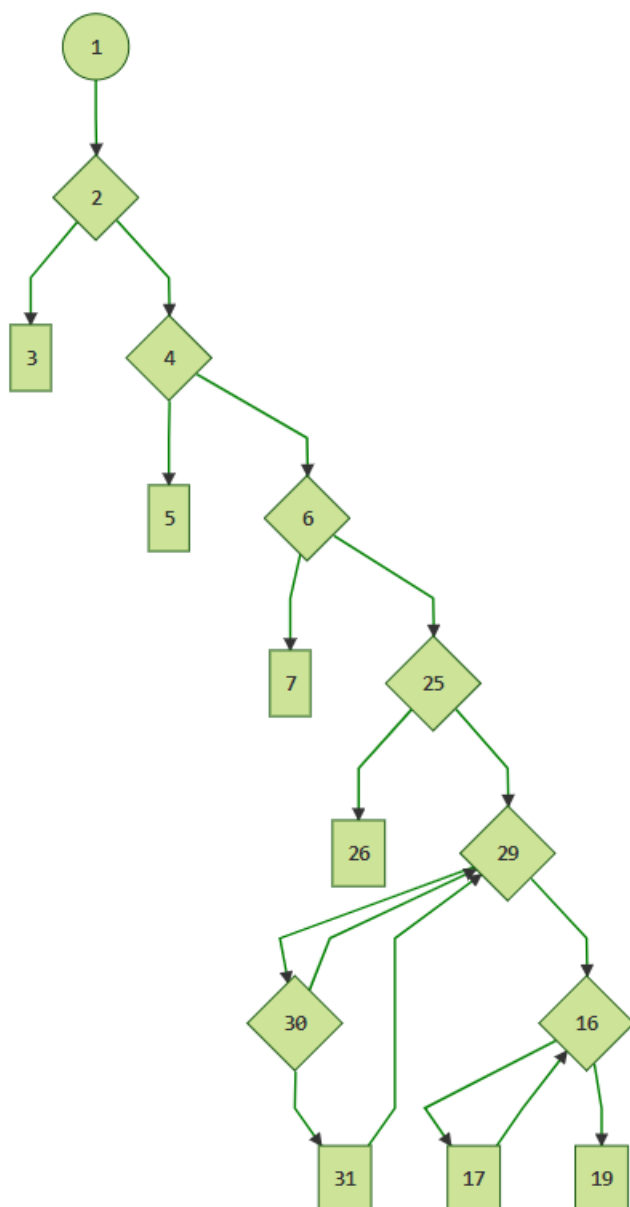
4.1 所选的被测函数

注意：不能与 3.1 节所选函数重复。

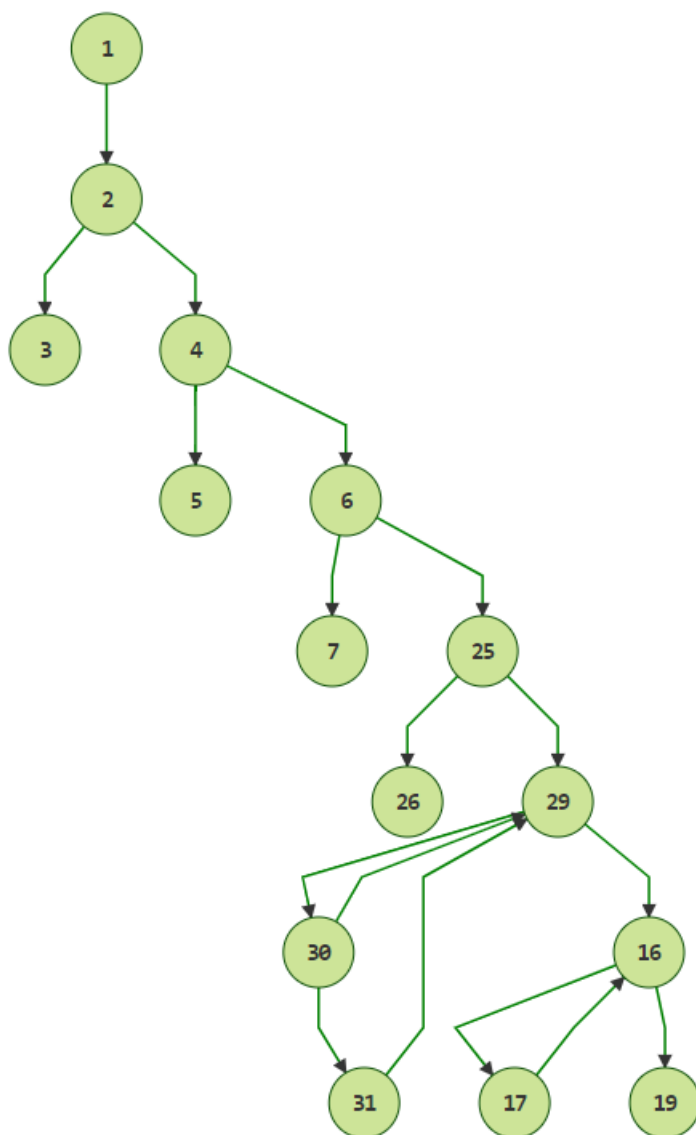
被测函数的名称	String queryBridgeWords(type G, String word1, String word2)		
功能描述	<p>在生成有向图之后，用户输入任意两个英文单词 word1 、 word2 ，程序从图中查询它们的“桥接词”。</p> <ul style="list-style-type: none"> word1 、 word2 的桥接词 word3：图中存在两条边 word1->word3, word3->word2。 输入的 word1 或 word2 如果不在图中出现，则输出 “No word1 or word2 in the graph!” 如果不存在桥接词，则输出 “No bridge words from word1 to word2!” 如果存在一个或多个桥接词，则输出 “The bridge words from word1 to word2 are: xxx, xxx, and xxx.” 		
被测函数的代码 （以 Eclipse 环境下的截图方式给出，确保能看清楚，并保留 Eclipse 为每行代码分配的行号，后续各部分均以此行号为准。如果一屏截取不下，可以分多屏截取，均插入右侧格子）	<pre> 1 public static String queryBridgeWords(WordGraph G, String word1, String word2) { 2 if (!G.nodes.containsKey(word1) && !G.nodes.containsKey(word2)) { 3 return "No \"" + word1 + "\" and \"" + word2 + "\" in the graph!"; 4 } else if (!G.nodes.containsKey(word1)) { 5 return "No \"" + word1 + "\" in the graph!"; 6 } else if (!G.nodes.containsKey(word2)) { 7 return "No \"" + word2 + "\" in the graph!"; 8 } else { 9 String[] words = queryBridgeWordsList(G, word1, word2); 10 if (words.length == 0) { 11 return "No bridge words from \"" + word1 + "\" to \"" + word2 + "\"!"; 12 } else if (words.length == 1) { 13 return "The bridge word from \"" + word1 + "\" to \"" + word2 + "\" is: " + words[0] + "."; 14 } else { 15 String hint = "The bridge words from \"" + word1 + "\" to \"" + word2 + "\" are: " + words[0]; 16 for (int i = 1; i + 1 < words.length; i++) { 17 hint += ", " + words[i]; 18 } 19 hint += " and " + words[words.length - 1]; 20 return hint + "."; 21 } 22 } 23 } 24 protected static String[] queryBridgeWordsList(WordGraph G, String word1, String word2) { 25 if (G.nodes.containsKey(word1) == false) 26 return new String[0]; 27 WordNode wordNode = G.nodes.get(word1); 28 ArrayList<String> bridgeWords = new ArrayList<String>(); 29 for (WordEdge wordEdge : wordNode.edges.values()) { 30 if (wordEdge.to.edges.containsKey(word2)) { 31 bridgeWords.add(wordEdge.to.text); 32 System.out.println("bridgeWords of " + word1 + " and " + word2 + ": " + wordEdge.to.text); 33 } 34 } 35 return (String[]) bridgeWords.toArray(new String[0]); 36 } </pre>		
输入参数列表	参数名	含义	数据类型
	G	图的所有信息	WordGraph
	Word1	第一个单词	String

	Word2	第二个单词	String
输出参数	含义		数据类型
	Word1 和 Word2 之间桥接词的情况		字符串
代码总行数	36		
包含的循环数	2		
包含的判定数	5		

4.2 程序流程图



4.3 控制流图



4.4 圈复杂度计算与基本路径识别

圈复杂度为：8（流图中判定结点的数目为 7, $7+1=8$ ）

基本路径 1: 1,2,3

基本路径 2: 1,2,4,5

基本路径 3: 1,2,4,6,7

基本路径 4: 1,2,4,6,25,26 (由于使用了函数且判断语句不相容，不可能遍历)

基本路径 5: 1,2,4,6,25,29,16,19

基本路径 6: 1,2,4,6,25,29,30,29,16,19

基本路径 7: 1,2,4,6,25,29,30,31,29,16,19

基本路径 8: 1,2,4,6,25,29,30,31,29,16,17,16,19

注意：各基本路径要使用 4.1 节表格里给出的行号。

4.5 测试用例设计

测试用例编号	输入数据	期望的输出	所覆盖的基本路径编号
1.	It haat	No \"it\" and \"haat\" in the graph!	1,2,3
2.	BridgeWord i	No \"BridgeWord\" in the graph!	1,2,4,5
3.	I haat	No \"haat\" in the graph!	1,2,4,6,7
4.	I play	No bridge words from \"i\" to \"play\"!	1,2,4,6,25,29,16,19
5.	Make become	The bridge word from \"make\" to \"become\" is: heat.	1,2,4,6,25,29,30,29,16,19
6.	I music	The bridge words from \"i\" to \"music\" are: heat, like and and.	1,2,4,6,25,29,30,31,29,16,19
7.	Tyfbuhdsnj i	No bridge words from \"tyfbuhdsnj\" to \"i\"!	1,2,4,6,25,29,30,31,29,16,17,16,19

4.6 jUnit 测试代码

针对 4.5 中的每一个用例，把其测试代码粘贴如下，代码必须是完整的。

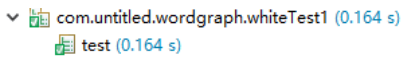
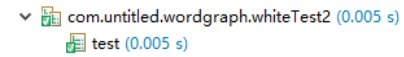
测试用例编号	jUnit 测试代码
1.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new </pre>

	<pre> PanelQueryBridge(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No \"it\" and \"haat\" in the graph!",PanelQueryBridge.queryBridgeWords(G,"it","haat")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
2.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new PanelQueryBridge(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No \"BridgeWord\" in the graph!",PanelQueryBridge.queryBridgeWords(G,"BridgeWord","i")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
3.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new PanelQueryBridge(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No \"haat\" in the graph!",PanelQueryBridge.queryBridgeWords(G,"i","haat")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>

4.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new PanelQueryBridge(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No bridge words from \"i\" to \"play\"!", PanelQueryBridge.queryBridgeWords(G, "i", "play")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
5.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new PanelQueryBridge(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("The bridge word from \"make\" to \"become\" is: heat.", PanelQueryBridge.queryBridgeWords(G, "make", "become")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
6.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new PanelQueryBridge(); WordGraph G = </pre>

	<pre> PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("The bridge words from \"i\" to \"music\" are: heat, like and and.", PanelQueryBridge.queryBridgeWords(G, "i", "music")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>
7.	<pre> @Before public void setUp() { System.out.println("---begin test---"); } @Test public void test() throws IOException { PanelQueryBridge PanelQueryBridge = new PanelQueryBridge(); WordGraph G = PanelOpenFile.createDirectedGraph("in.txt"); assertEquals("No bridge words from \"tyfbuhdsnj\" to \"i\"!", PanelQueryBridge.queryBridgeWords(G, "tyfbuhdsnj", "i")); } @After public void tearDown() { System.out.println("---end test---"); } </pre>

4.7 JUnit 单元测试结果

测试用例编号	期望输出	实际输出	是否通过测试，请给出屏幕截图
1.	No \"it\" and \"haat\" in the graph!	No \"it\" and \"haat\" in the graph!	 com.untitled.wordgraph.whiteTest1 (0.164 s)
2.	No \"BridgeWord\" in the graph!	No \"BridgeWord\" in the graph!	 com.untitled.wordgraph.whiteTest2 (0.005 s)

3.	No \"haat\" in the graph!	No \"haat\" in the graph!	com.untitled.wordgraph.whiteTest3 (0.005 s) test (0.005 s)
4.	No bridge words from \"i\" to \"play\"!	No bridge words from \"i\" to \"play\"!	com.untitled.wordgraph.whiteTest4 (0.003 s) test (0.003 s)
5	The bridge word from \"make\" to \"become\" is: heat.	The bridge word from \"make\" to \"become\" is: heat.	com.untitled.wordgraph.whiteTest5 (0.004 s) test (0.004 s)
6	The bridge words from \"i\" to \"music\" are: heat, like and and.	The bridge words from \"i\" to \"music\" are: heat, like and and.	com.untitled.wordgraph.whiteTest6 (0.007 s) test (0.007 s)
7	No bridge words from \"tyfbuhdsnj\" to \"i\"!	No bridge words from \"tyfbuhdsnj\" to \"i\"!	com.untitled.wordgraph.whiteTest7 (0.005 s) test (0.005 s)

4.8 Eclemma 代码覆盖度分析

给出 Eclemma 的代码覆盖度分析报告的截图。

PanelQueryBridge

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
PanelQueryBridge()		100%		n/a	0 1	0 32	0 1
queryBridgeWords(WordGraph, String, String)		100%		100%	0 8	0 16	0 1
addComponent(Component, int, int, int, int, int, int, int)		100%		n/a	0 1	0 12	0 1
Total	0 of 404	100%	0 of 14	100%	0 10	0 60	0 3

4.9 未通过测试的原因分析及代码修改

分析自己的 Lab1 代码为何未通过 4.6 节表格中某些测试用例的原因，并通过修改代码消除此类 BUG。必要时给出修改后的代码。

若 4.7 节表格中没有未通过的测试用例，本节可空。

修改代码之后，请重新填写下表，保证所有测试用例都能通过测试。

测试用例编号	期望输出	实际输出	是否通过测试，请给出屏幕截图
1.			

4.10 Git/GitHub 操作

给出命令行界面截图：

- 在本组 Lab1 的 git 仓库里，建立新的 git 分支，命名为 lab6w (w 代表 white-box)

testing);

```
qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git checkout -b lab6w
Switched to a new branch 'lab6w'

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6w)
$
```

- 将 lab6w 合并到 master 分支，并推送至 GitHub。

```
qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6w)
$ git add src

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6w)
$ git commit -m "change white test"
[lab6w dce33d3] change white test
5 files changed, 8 insertions(+), 39 deletions(-)

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (lab6w)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git merge lab6w
Auto-merging src/test/java/com/untitled/wordgraph/whiteTest1.java
CONFLICT (content): Merge conflict in src/test/java/com/untitled/wordgraph/whiteTest1.java
Auto-merging src/test/java/com/untitled/wordgraph/whiteTest.java
CONFLICT (content): Merge conflict in src/test/java/com/untitled/wordgraph/whiteTest.java
Automatic merge failed; fix conflicts and then commit the result.

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master|MERGING)
$ git add src

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master|MERGING)
$ git commit -m "merge lab6w"
[master 4fbc07] merge lab6w

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$ git push origin master
Counting objects: 48, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (32/32), done.
Writing objects: 100% (48/48), 3.29 KiB | 481.00 KiB/s, done.
Total 48 (delta 23), reused 0 (delta 0)
remote: Resolving deltas: 100% (23/23), completed with 15 local objects.
To https://github.com/mayukuner/Lab1.git
   c88f18d..4fbc07  master -> master

qio@DESKTOP-LQ1KJIM MINGW64 ~/Desktop/Lab1 (master)
$
```

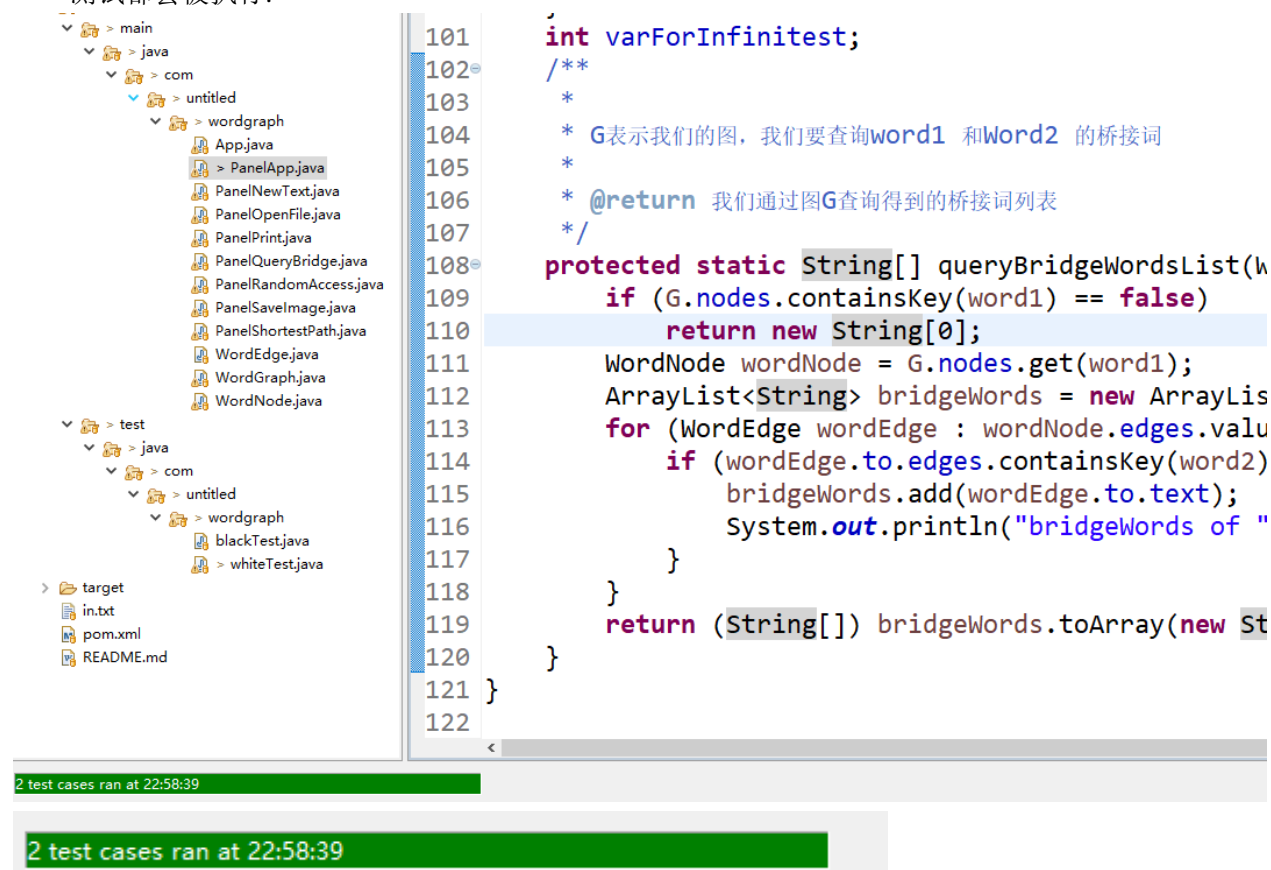
5 Infinitest 持续测试

5.1 代码修改前

```
101
102  /**
103     *
104     * G表示我们的图，我们要查询word1 和Word2 的桥接词
105     *
106     * @return 我们通过图G查询得到的桥接词列表
107     */
```

5.2 代码修改后

由于 PanelApp 所涉及的类较多,因此黑盒(blackTest.java)和白盒(whiteTest.java)两个测试都会被执行.

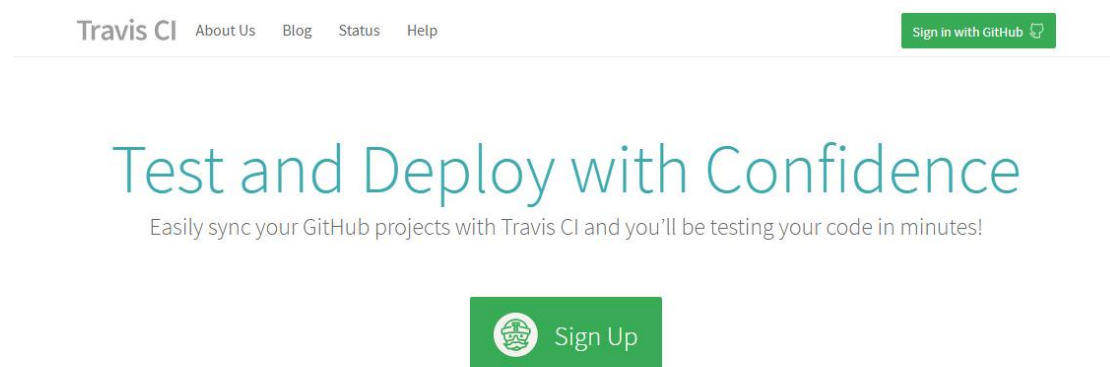


6 计划与实际进度

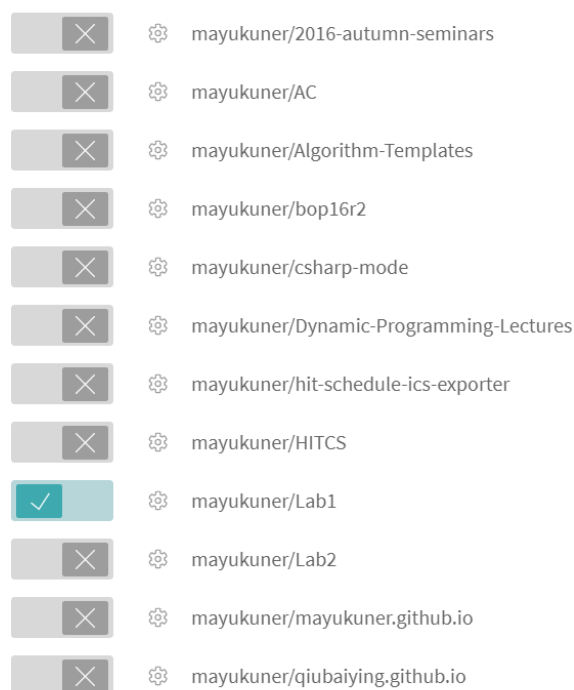
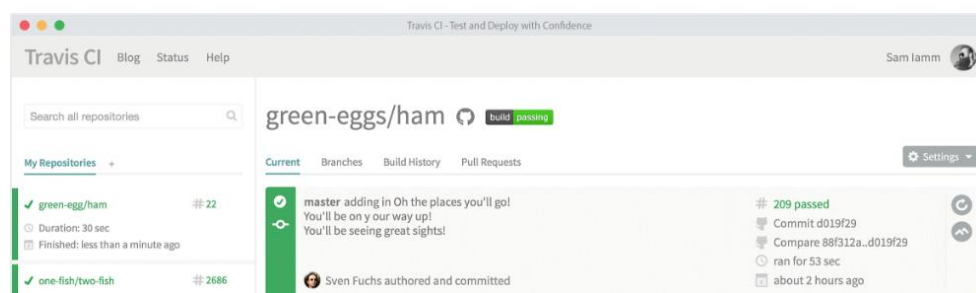
任务名称	计划时间长度（分钟）	实际耗费时间（分钟）	提前或延期的原因分析
黑盒测试	50	240	由于黑盒测试用例写完了之后,我在选取白盒测试用例的时候,发现出了一点问题,所以又重新选择了黑盒测试用例,所以浪费了很长的时间.
白盒测试	100	150	白盒测试用例画程序流程图还是比较麻烦的.
写报告	100	60	程序没啥错误,所以基本不需要怎么修改,只进行了简单的测试就可以了

7 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_PROJECT_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

8 小结

(1) “编写代码”和“编写测试代码”是什么关系？

在传统的瀑布模型里面是先编写代码,然后再编写测试代码,二者存在一定的先后顺序,但是在我们所提倡的敏捷开发之中,提出了以测试为驱动的开发,我们往往根据用户故事直接编写测试数据,然后再编写代码,debug,直到通过测试.

(2) “设计测试用例”和“编写测试代码”是什么关系？

“设计测试用例”是“编写测试代码”的前一步骤。在“编写测试代码”前,测试人员应设计好测试用例。测试用例的设计应该考虑到测试代码编写的复杂度以及代码的覆盖度。

(3) 使用黑盒方法设计的测试用例的“测试覆盖度”和白盒方法的“测试覆盖度”,二者可能会有什么差异?为什么?

白盒的测试覆盖度往往比黑盒的测试覆盖度要高上不少,因为我们在编写白盒测试用例的时候,使用基本路径法,这样的话,代码中每一条语句至少执行一次,语句的覆盖度就会很高,

但是我们在进行黑盒测试的时候,考虑的是用户故事,我们需要检测代码是不是正确的实现了应该实现的功能,但是由于程序员开发的功能 并不一定都是黑盒测试需要测试的部分,所以测试覆盖度就会低上不少.

(4) “程序员应该为自己编写的每个函数设计测试用例、写出测试代码,并让自己的程序通过测试”,你是否愿意养成这个习惯?你觉得“测试”该是独立的测试人员所主导的任务吗?

我十分愿意养成这样的习惯,程序员应该为自己的代码负责,这不仅是对客户负责还是对自己负责,而且测试自己代码的能力是一个程序员必须具备的能力.

最好的情况是自己对自己的代码进行白盒测试部分,因为白盒测试要求对代码的了解,对于测试人员来说,就算看懂了你的代码,也很难对你的代码理解到位,所以白盒测试不应该由独立的测试人员进行.

但是由于黑盒测试不需要对代码了解,所以我觉得这部分的工作由独立的测试人员来做也无可厚非.

(5) 从实验过程中你体会到“黑盒测试”和“白盒测试”谁的难度更高?

我个人觉得是白盒测试的难度高一些,因为你需要理解代码,分析代码的控制结构,在此基础上设计出测试用例,但是黑盒测试用例,我们只需要根据边界值分析法和等价类划分法,就可以找到我们需要的测试用例,并不需要我们付出很多的脑力.

(6) 其他体会。

我个人觉得在白盒测试里面,判定覆盖就已经可以满足我们的需求了,因为我们编写的程序往往判断条件很简单,出现错误往往不是因为判断条件错误,所以我觉得如果我们能够做到每一个分支至少执行一次的话,一般就可以找到所有的错误.不过这仅仅是我的个人想法,仅仅依托我的经验,并不是很有说服力.

由于随机游走代码层面的不确定性,我们无法根据输入来判断程序会如何执行,所以无法进行白盒测试,这和老师上课讲的内容并不是很兼容,我通过自己的认真思索,觉得随机函数要进行白盒测试的时候可以通过将随机函数变化一下,写一个伪随机函数,就像白盒测试里面写桩函数一样,我们自己写一个函数去替换随机函数,我

们写的函数可以根据输入产生确定的输出,这样的话,就可以顺利完成包含随机函数代码的白盒测试.