2024 年春季学期

计算学部《软件工程》课程

实验报告

Lab3 代码评审与单元测试

姓名	学号	联系方式
徐柯炎	**2021110683**	**2259245769@qq.com/15189789979**
高浚豪	2021111700	2152246548@qq.com/13359705210

目 录

[文档全部完成之后，请更新上述区域]

1 实验要求

(1) 代码评审：

- 按照 Lab1 分组，两人共同完成实验；
- 针对 Lab1 所完成的代码，进行代码评审(走查) ，从代码规范性和正确性角度对代码进行评价；
- 使用以下两个工具完成实验：
  - Checkstyle
  - SpotBugs

(2) 单元测试：

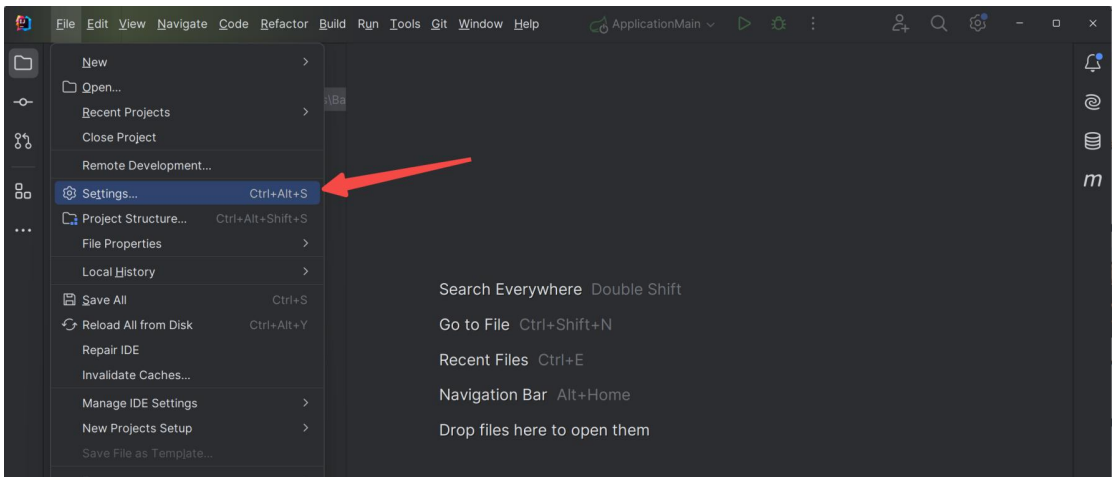
- 按 Lab1 分组，两人共同完成实验。
- 设计黑盒测试用例和白盒测试用例。
- 在 JUnit 环境下撰写测试代码并执行测试。
- 使用 EclEmma 或 IDE 自带工具统计测试的覆盖度。

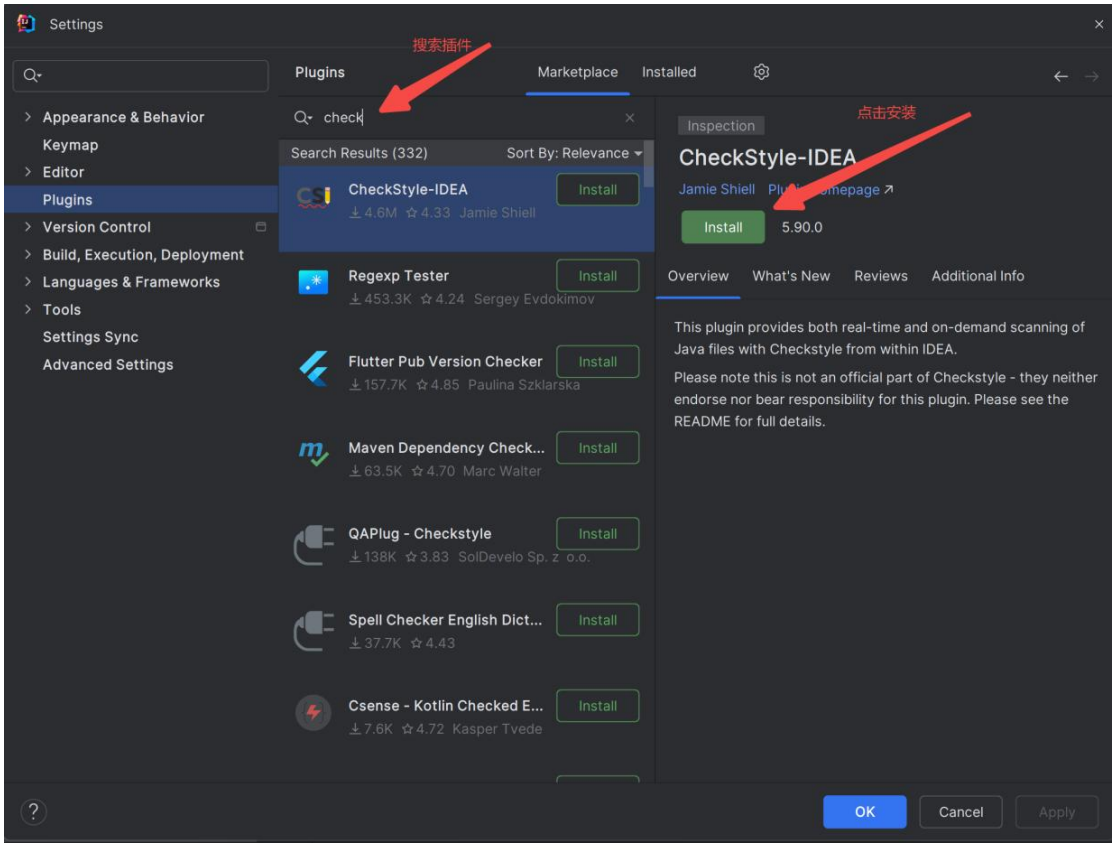
2 在 IDE 中配置代码审查与分析工具

简要描述在 IDE 中安装和配置 Checkstyle、SpotBugs、EclEmma、JUnit 等插件或 IDE 自带插件的过程。

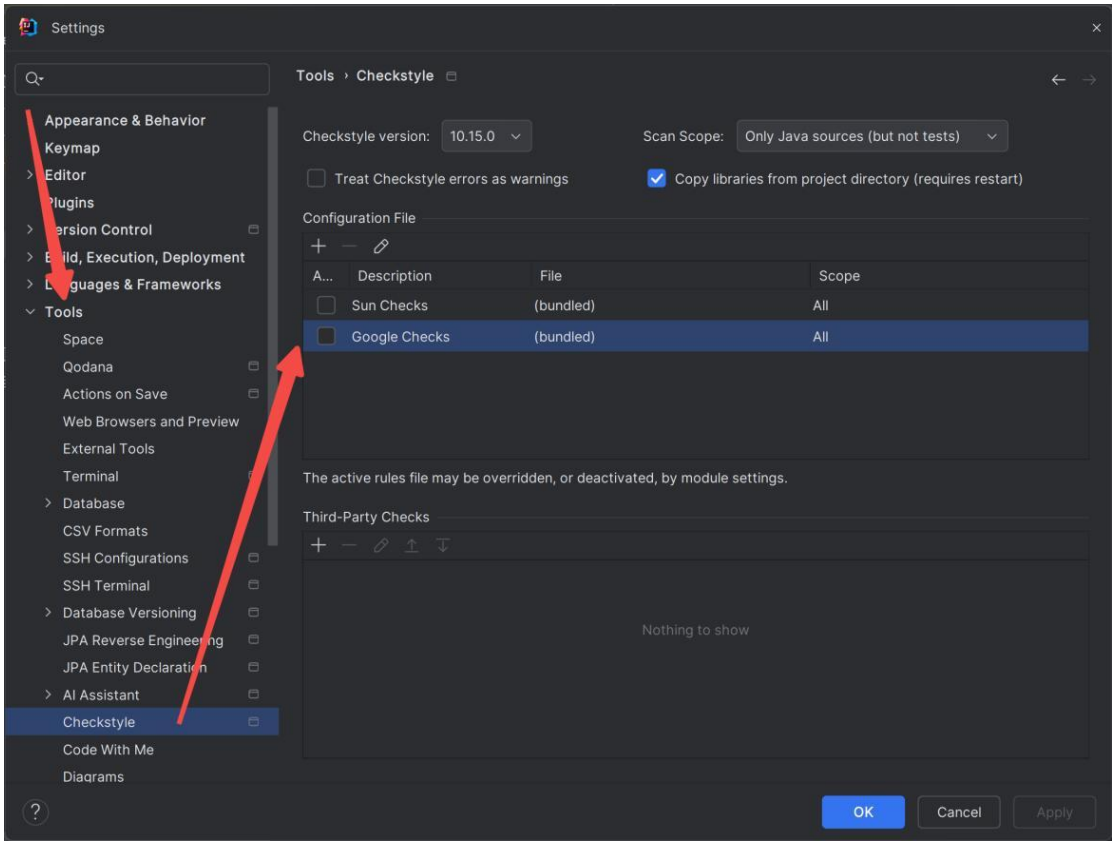
2.1 Checkstyle

首先打开 IDEA 菜单栏 Settings 中的 plugins，搜索 CheckStyle-IDEA 并安装；



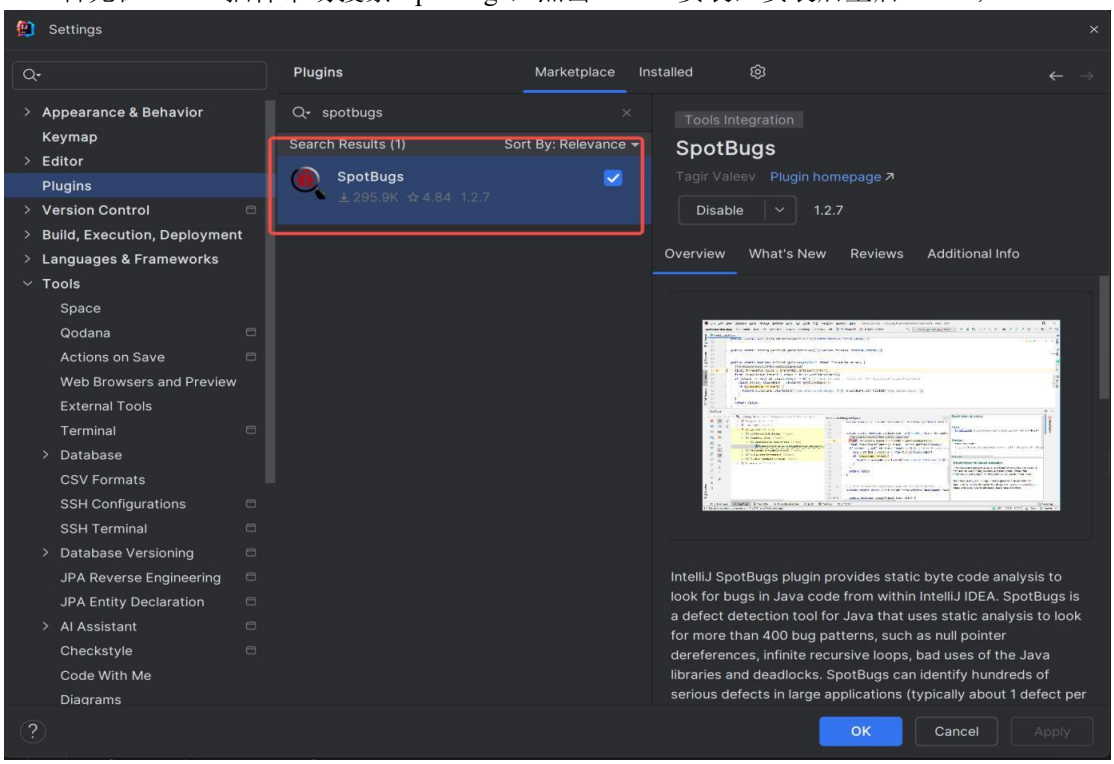


然后配置 checkstyle。这里我们使用 Google Checks。

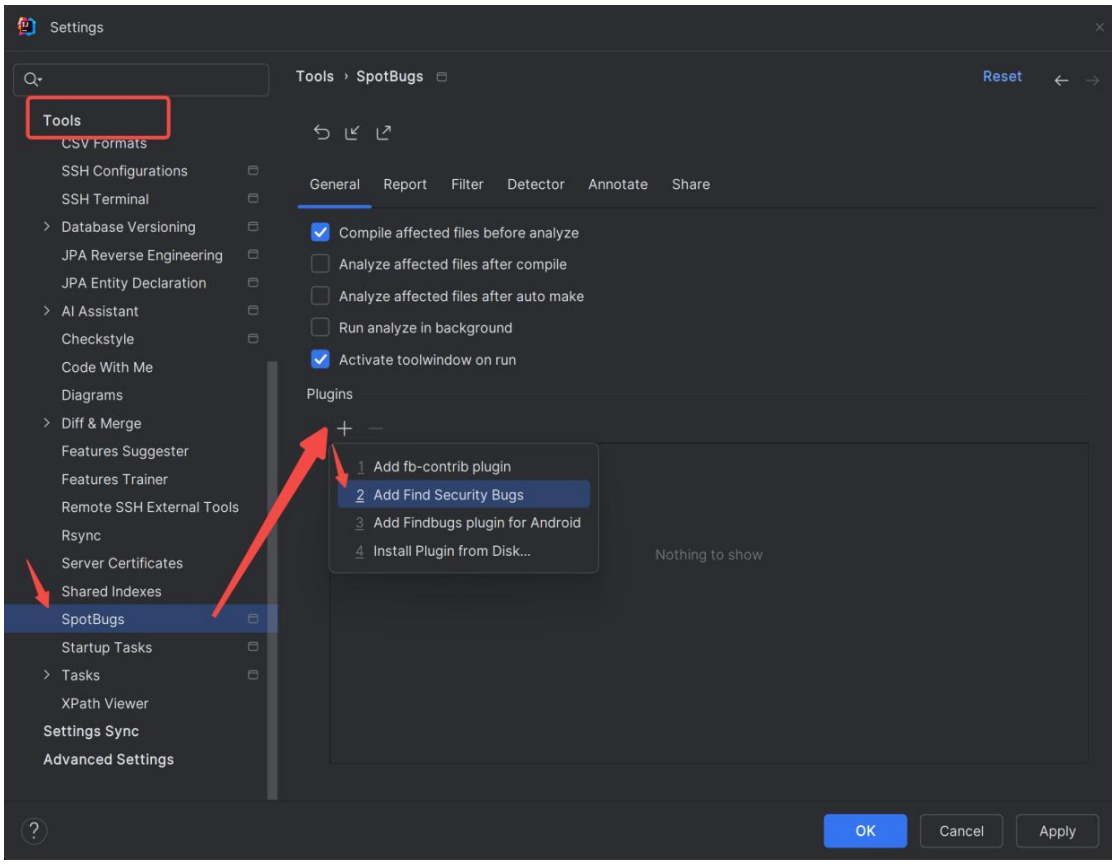


2.2 SpotBugs

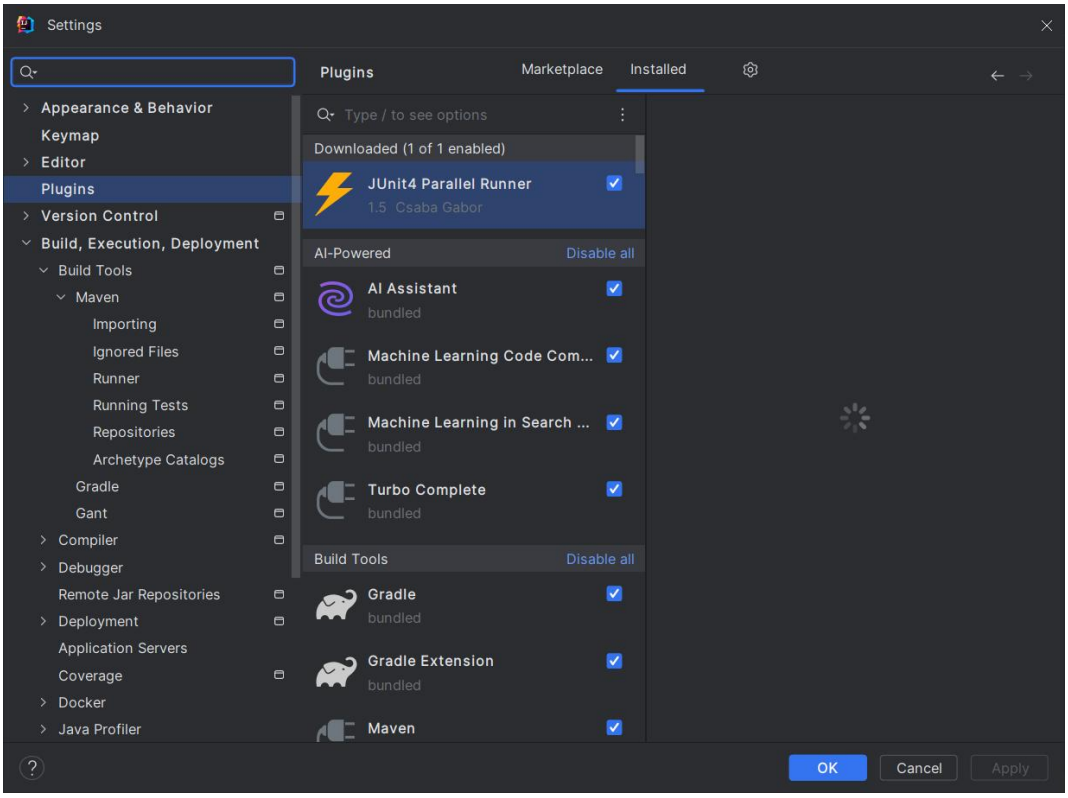
首先在 IDEA 插件市场搜索 SpotBugs，点击 Install 安装，安装后重启 IDEA；

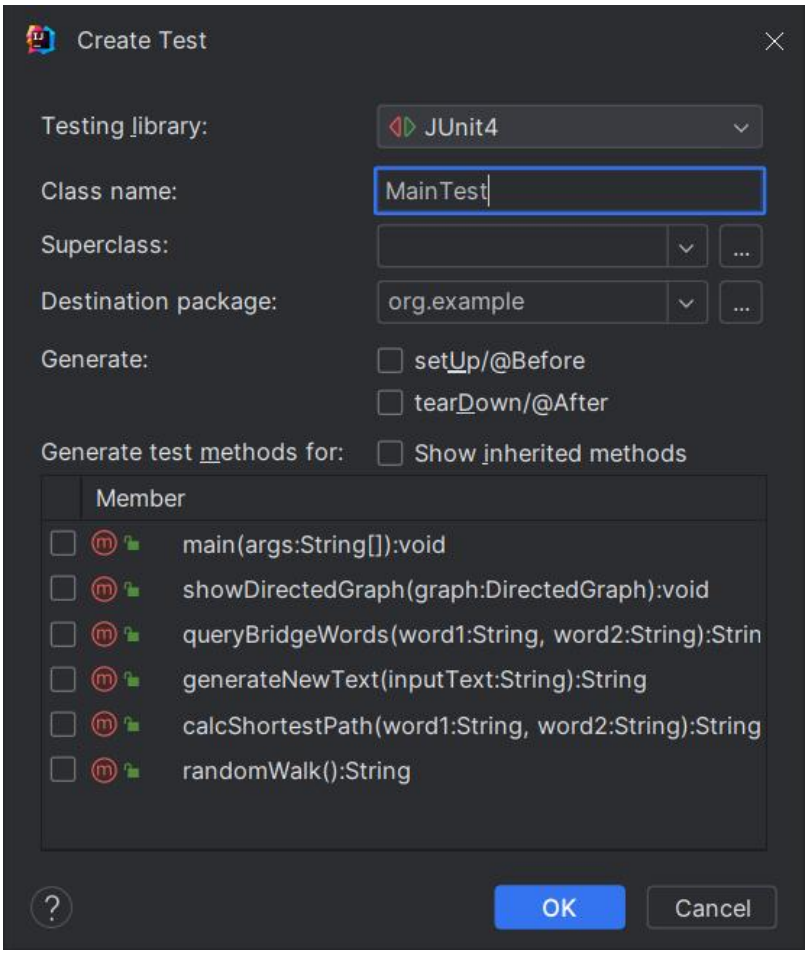


然后在设置的 Tools 下，点击 SpotBugs，按下图所示点击加号，添加规则。



2.3 Junit


Settings
Plugins
Marketplace
Installed
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Maven
Importing
Ignored Files
Runner
Running Tests
Repositories
Archetype Catalogs
Gradle
Gant
Compiler
Debugger
Remote Jar Repositories
Deployment
Application Servers
Coverage
Docker
Java Profiler
Type / to see options
Downloaded (1 of 1 enabled)
JUnit4 Parallel Runner
1.5 Csaba Gabor
AI-Powered
Disable all
AI Assistant
bundled
Machine Learning Code Com...
bundled
Machine Learning in Search ...
bundled
Turbo Complete
bundled
Build Tools
Disable all
Gradle
bundled
Gradle Extension
bundled
Maven
OK
Cancel
Apply


Create Test
Testing library:
JUnit4
Class name:
MainTest
Superclass:
Destination package:
org.example
Generate:
setUp/@Before
tearDown/@After
Generate test methods for:
Show inherited methods
Member
main(args:String[]):void
showDirectedGraph(graph:DirectedGraph):void
queryBridgeWords(word1:String, word2:String):Strin
generateNewText(inputText:String):String
calcShortestPath(word1:String, word2:String):String
randomWalk():String
OK
Cancel

3 Checkstyle 所发现的代码问题清单及原因分析

针对同种类型的问题，只需要列出一个典型代表即可。

编号	问题描述	类型	所在代码行号	修改策略
1	xx 元素缩进了 x 个缩进符，应为 y 个。	Indentation	几乎所有行	修改 idea 的缩进符策略，从四个修改为两个
2	注释应与第 x 行代码同样缩进 y 个缩进符，而不是 0 个。	CommentsIndentation	27、30、33、284、316、324、341、354	修改 idea 的注释风格，使得注释符号紧跟在注释前面
3	导入组之前的额外空行 'java.io.IOException'。	CustomImportOrder	9	删除空行
4	导入语句'java.io.IOException' 字典顺序错误。应在 'org.jetbrains.annotations.NotNull' 之前。	CustomImportOrder	10、11、12、13、14	手动调整顺序
5	不应使用 '.*' 形式的导入 - java.util.* 。	AvoidStarImport	4	更改导入形式，分别导入
6	缺少 Javadoc 。	MissingJavadocType	12	添加注释
7	缺少 Javadoc 。	MissingJavadocMethod	17	添加注释
8	Parameter name 'G' must match pattern '^[a-z]([a-z0-9][a-zA-Z0-9]*)?$'.	ParameterName	89	更改变量命名方式
9	本行字符数 109 个，最多：100 个。	LineLength	114	换行
10	Method name 'ShortestPath' must match pattern '^[a-z][a-z0-9]\w*$'	MethodName	174	更改方法命名方式

4 SpotBugs 所发现的代码问题清单及原因分析

针对同种类型的问题，只需要列出一个典型代表即可。

优先级	问题描述	违反的规则集	所在代码行号	修改策略
3	This random generator	Security	171	修改随机方法，在

	(java.lang.Math.random()) is predictable			这里我们采用 SecureRandom()
2	Random object created and used only once	Bad practice	171	将 SecureRandom() 声明为一个静态的类成员，从而复用这个成员
1	Found reliance on default encoding: new	Internationalization	28、40、76	明确指定字符编码，这里采用 UTF-8
1	getNodes() may expose internal representation by returning DirectedGraph.nodes	Malicious code vulnerability	173	返回一个不可变的视图来保护内部表示。在这里我们使用 Collections.unmodifiableList 来实现
3	Unused field: Node.neighbors	Performance	9	删除此变量即可
2	Exceptional return value of java.io.File.delete() ignored	Bad practice	45	记录 delete 的返回值，并增加删除失败时的操作

5 针对 Lab1 的黑盒测试

5.1 所选的被测函数及其需求规约

```
public static String queryBridgeWords(String word1, String word2)
```

函数功能：

queryBridgeWords 函数用于在一个有向图中查找从 word1 到 word2 的桥接词（bridge words）。桥接词是那些在两个指定词之间存在的词。

输入描述：

word1（类型：String）：查询桥接词的起始单词。

word2（类型：String）：查询桥接词的目标单词。

输出描述：

返回一个 String 类型的结果，该结果可能是以下几种情况：

如果 word1 和 word2 中的任何一个单词不在图中，返回描述缺失单词的错误信息。例如："No "word1" and "word2" in the graph!" 或 "No "word1" in the graph!"。

如果在 word1 和 word2 之间没有桥接词，返回："No bridge words from "word1" to "word2"!"。

如果存在桥接词，返回："The bridge words from "word1" to "word2" is: bridgeWord1 bridgeWord2 ..."，列出所有桥接词。

5.2 等价类划分结果

请根据自己的情况扩展该表格，给各个等价类唯一的编号
根据需要，增加下表的行数

约束条件说明	有效等价类及其编号		无效等价类及其编号	
word1 和 word2 都在图中	word1 和 word2 都在图中，且存在桥接词	(1)	word1 和 word2 都不在图中	(6)
	word1 和 word2 都在图中，但没有桥接词存在	(2)	word1 和 word2 都不在图中	(6)
	word1 和 word2 相同	(3)	word1 和 word2 都不在图中	(6)
	word1 和 word2 直接相连且之间没有桥接词	(4)	word1 和 word2 都不在图中	(6)
word1 或 word2 在图中不存在	word1 或 word2 不在图中	(5)	word1 和 word2 都不在图中	(6)
			输入 word1 或 word2 为 null 或空字符串	(7)

5.3 测试用例设计

测试用例编号	输入	期望输出	所覆盖的等价类编号
1.	"to", "strange"	"The bridge words from "to" to "strange" is: explore"	有效等价类 1
2.	"boldly", "before"	"No bridge words from "boldly" to "before"!"	有效等价类 2
3.	"out", "out"	No bridge words from "out" to "out"!	有效等价类 3
4.	"to", "boldly"	"No bridge words from "to" to "boldly"!"	有效等价类 4
5.	"to", "gjh"	No "gjh" in the graph!	有效等价类 5

6.	"hello", "world"	"No "hello" and "world" in the graph!"	无效等价类 6
7.	"world", ""	"No "world" and "" in the graph!"	无效等价类 7

5.4 JUnit 测试代码

针对 5.3 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.	package org.example;
	
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	
	import org.example.util.Func;
	
	import java.util.List;
	
	public class MainTest {
	
	@Test
	public void testValidBridgeWordsExist() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = main.queryBridgeWords("to", "strange");
	assertEquals("The bridge words from \"to\" to \"strange\" is: explore ", result);
	}
	}

2.	package org.example; import org.junit.Test; import static org.junit.Assert.assertEquals; import org.example.util.Func;

	import java.util.List;
	
	public class MainTest {
	
	public void testNoBridgeWordsExist() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = org.example.Main.queryBridgeWords("boldly", "before");
	assertEquals("No bridge words from \"boldly\" to \"before\"!", result);
	}
	
	}

3.	package org.example;
	
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	
	import org.example.util.Func;
	
	import java.util.List;
	
	public class MainTest {
	
	@Test
	public void testWordsEqual() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = org.example.Main.queryBridgeWords("out", "out");
	assertEquals("No bridge words from \"out\" to \"out\"!", result);
	}
	
	}

4.	package org.example; import org.junit.Test; import static org.junit.Assert.assertEquals; import org.example.util.Func;

	import java.util.List;
	
	public class MainTest {
	
	@Test
	public void testWordsConnect() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = org.example.Main.queryBridgeWords("to", "boldly");
	assertEquals("No bridge words from \"to\" to \"boldly\"!", result);
	}
	
	}

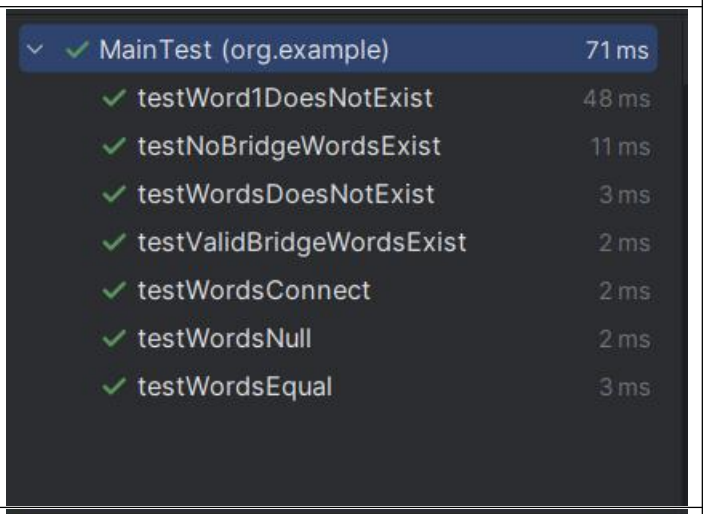
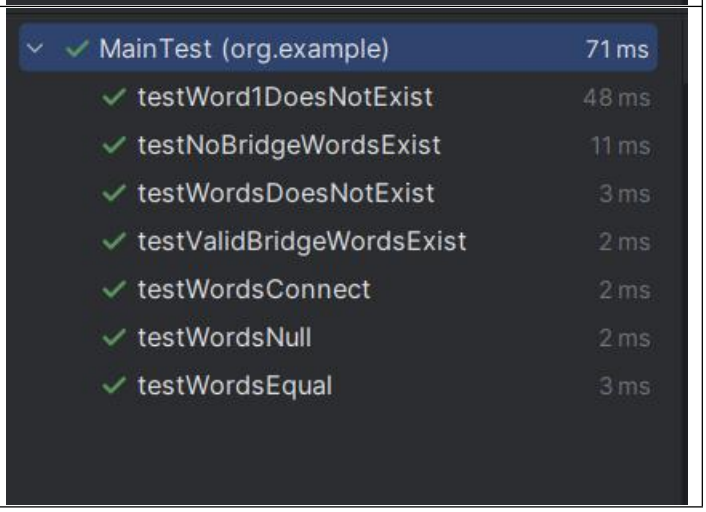
5.	package org.example;
	
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	
	import org.example.util.Func;
	
	import java.util.List;
	
	public class MainTest {
	
	@Test
	public void testWord1DoesNotExist() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = org.example.Main.queryBridgeWords("gjh", "to");
	assertEquals("No \"gjh\" in the graph!", result);
	}
	
	}

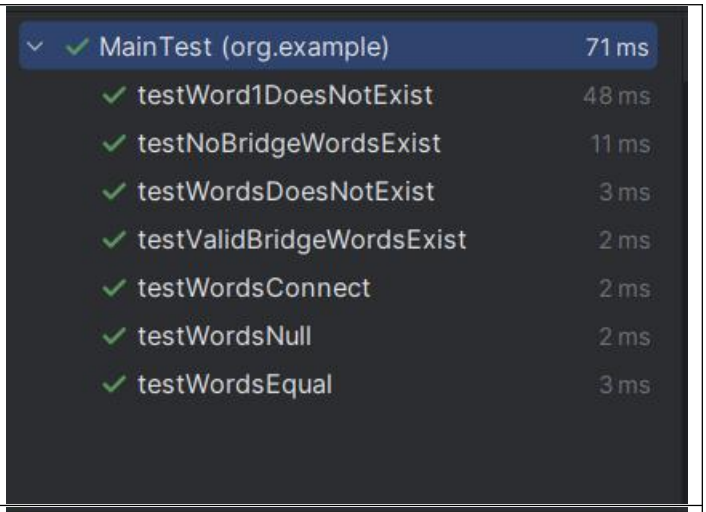
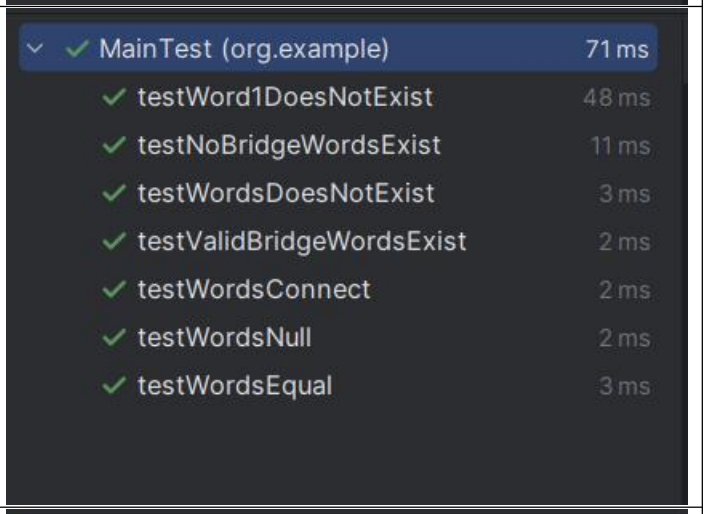
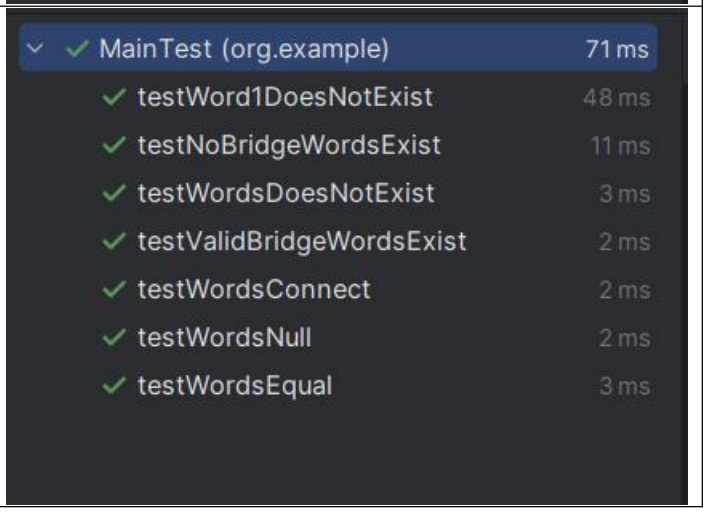
6.	package org.example; import org.junit.Test; import static org.junit.Assert.assertEquals;

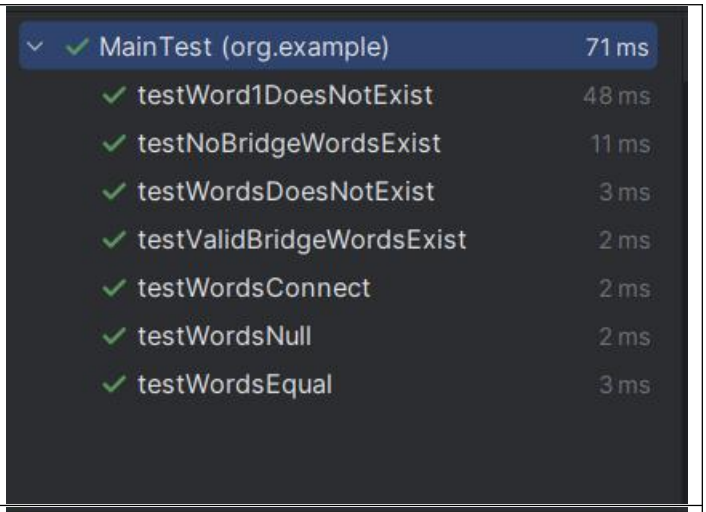
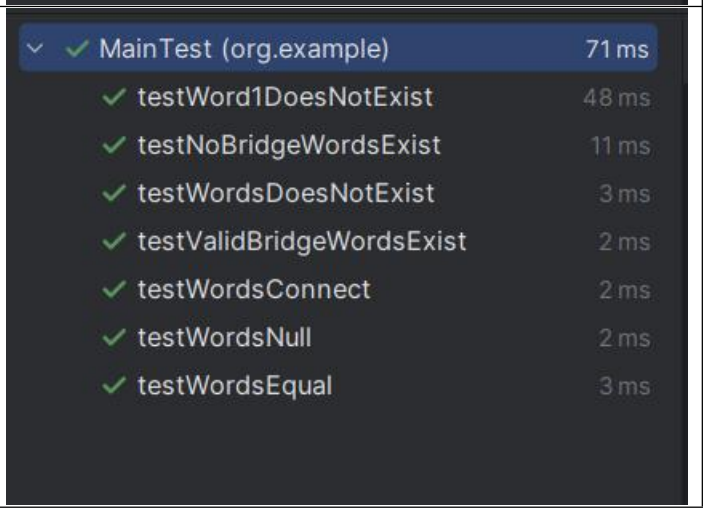
	import org.example.util.Func;
	
	import java.util.List;
	
	public class MainTest {
	
	@Test
	public void testWordsDoesNotExist() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = org.example.Main.queryBridgeWords("hello", "world");
	assertEquals("No \"hello\" and \"world\" in the graph!", result);
	}
	
	}

7.	package org.example;
	
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	
	import org.example.util.Func;
	
	import java.util.List;
	
	public class MainTest {
	
	@Test
	public void testWordsNull() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String result = org.example.Main.queryBridgeWords("world", " ");
	assertEquals("No \"world\" and \" \" in the graph!", result);
	}
	
	}

5.5 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.	The bridge words from "to" to "strange" is: explore	The bridge words from "to" to "strange" is: explore	
2.	"No bridge words from "boldly" to "before"!	"No bridge words from "boldly" to "before"!	

3.	No bridge words from "out" to "out"!	No bridge words from "out" to "out"!	
4.	No bridge words from "to" to "boldly"!	No bridge words from "to" to "boldly"!	
5.	No "gjh" in the graph!	No "gjh" in the graph	

6.	No "hello" and "world" in the graph!	No "hello" and "world" in the graph!	
7.	"No "world" and " " in the graph!	"No "world" and " " in the graph!	

5.6 未通过测试的原因分析及代码修改

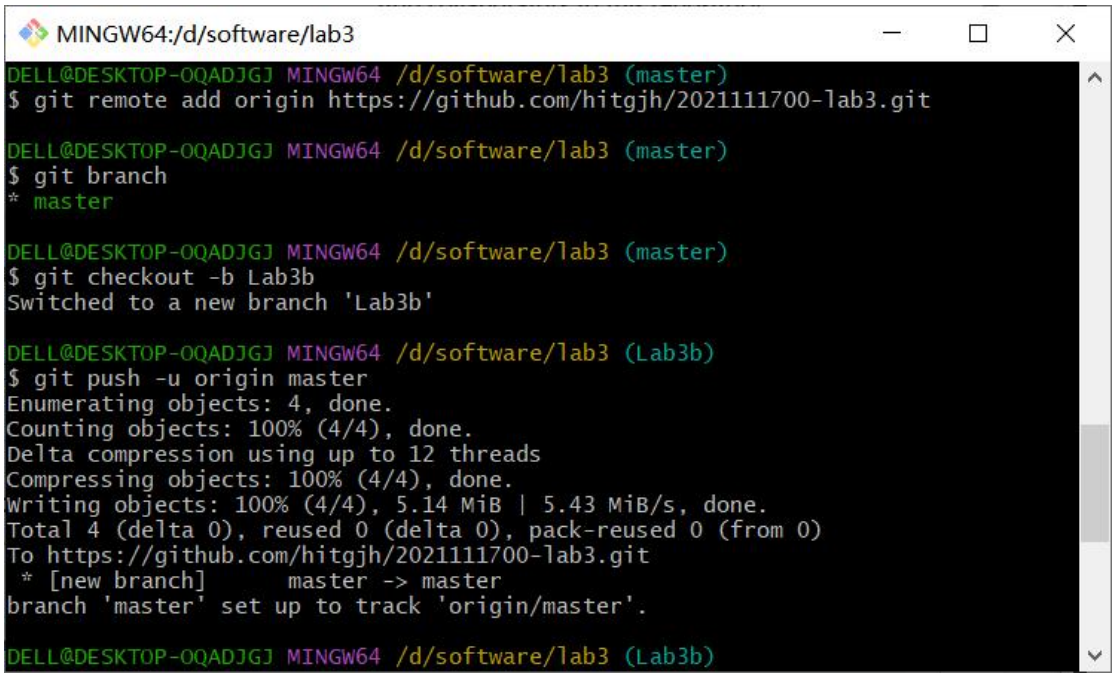
测试全部通过

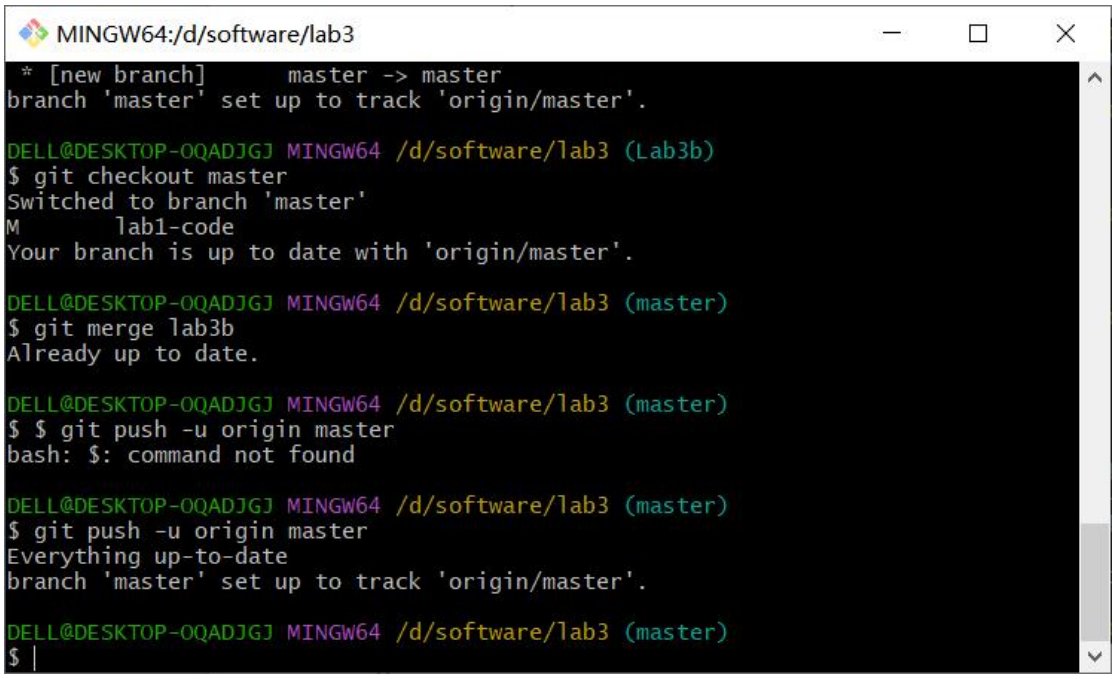
修改代码之后，请重新填写下表，尽可能保证所有测试用例都能通过测试。

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.			
2.			

3.			
4.			

5.7 Git 操作记录





6 针对 Lab1 的白盒测试

6.1 所选的被测函数

注意：不能与 5.1 节所选函数重复。

被测函数的名称	calcShortestPath
功能描述	输入两个节点，函数计算出两个节点间的最短路径，输出路径及其长度，如果只输入了一个节点，则输出该节点到所有节点的路径，如输入节点在图中不存在，则报告错误信息
被测函数的代码（以 IDE 环境下的截图方式给出，确保能看清楚，并保留 IDE 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，均插入右侧格中）

```java
public static String calcShortestPath(String word1, String word2) {
    Node startNode = graph.getNode(word1);
    if (startNode == null) {
        return "The word \"" + word1 + "\" is not in the graph.";
    }

    if (Objects.equals(word2, b: "")) {
        StringBuilder result = new StringBuilder();
        List<Node> nodes = graph.getNodes();
        for (Node endNode : nodes) {
            result.append(shortestPath(startNode, endNode, showGraph: false)).append("\n");
        }
        return result.toString();
    }

    Node endNode = graph.getNode(word2);
    if (endNode == null) {
        return "The word \"" + word2 + "\" is not in the graph.";
    }

    return shortestPath(startNode, endNode, showGraph: true);
}

2 usages  hitgjh *
private static String shortestPath(Node startNode, Node endNode, boolean showGraph) {
    String startWord = startNode.getWord();
    String endWord = endNode.getWord();

    PriorityQueue<Node> queue = new PriorityQueue<>(Comparator.comparingInt(node -> node.dis
    Map<Node, Integer> distances = new HashMap<>();
```

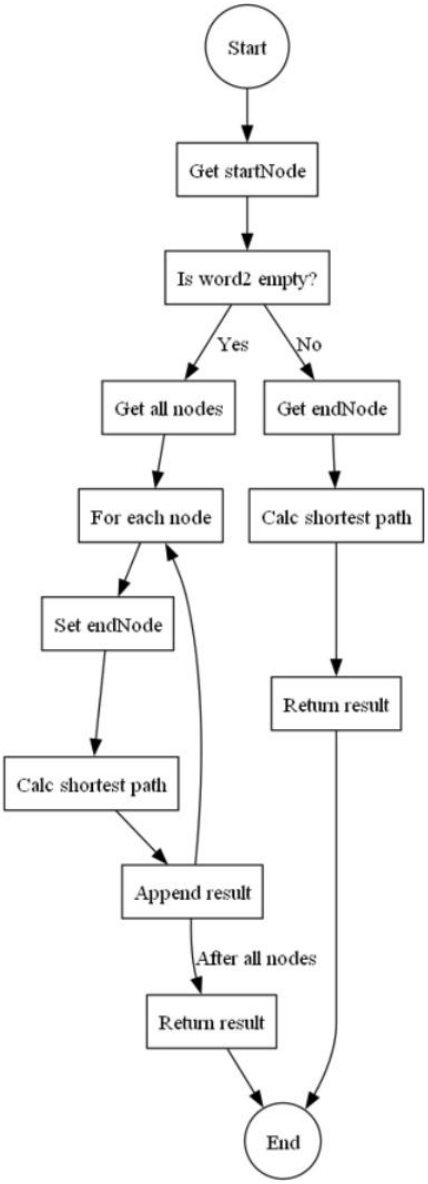
```java
    Map<Node, Node> previousNodes = new HashMap<>();

    for (Node node : graph.getNodes()) {
        distances.put(node, Integer.MAX_VALUE);
        previousNodes.put(node, null);
    }

    distances.put(startNode, 0);
    queue.add(startNode);

    while (!queue.isEmpty()) {
        Node currentNode = queue.poll();

        if (currentNode.equals(endNode)) {
            break;
        }

        for (Node neighbor : graph.getNeighbors(currentNode)) {
            int weight = graph.getEdgeWeight(currentNode, neighbor);
            int distanceThroughCurrent = distances.get(currentNode) + weight;

            if (distanceThroughCurrent < distances.get(neighbor)) {
                distances.put(neighbor, distanceThroughCurrent);
                previousNodes.put(neighbor, currentNode);
                queue.add(neighbor);
            }
        }
    }

    if (previousNodes.get(endNode) == null) {
```

```java
        return "There is no path from \"" + startWord + "\" to \"" + endWord + "\".";
    }

    List<Node> shortestPath = new ArrayList<>();
    Node currentNode = endNode;

    while (currentNode != null) {
      shortestPath.add(currentNode);
      currentNode = previousNodes.get(currentNode);
    }

    Collections.reverse(shortestPath);

    int totalWeight = 0;
    StringBuilder pathBuilder = new StringBuilder("Shortest path from \""
            + startWord + "\" to \"" + endWord + "\":");

    for (int i = 0; i < shortestPath.size(); i++) {
      pathBuilder.append(shortestPath.get(i).getWord());

      if (i < shortestPath.size() - 1) {
        int weight = graph.getEdgeWeight(shortestPath.get(i), shortestPath.get(i + 1));
        totalWeight += weight;
        pathBuilder.append(" → ").append(weight);
      }
    }

    pathBuilder.append("\nTotal weight of the path: ").append(totalWeight);

    if (showGraph) {
      String dotContent = graph.toDigraphStringPath(shortestPath);
```

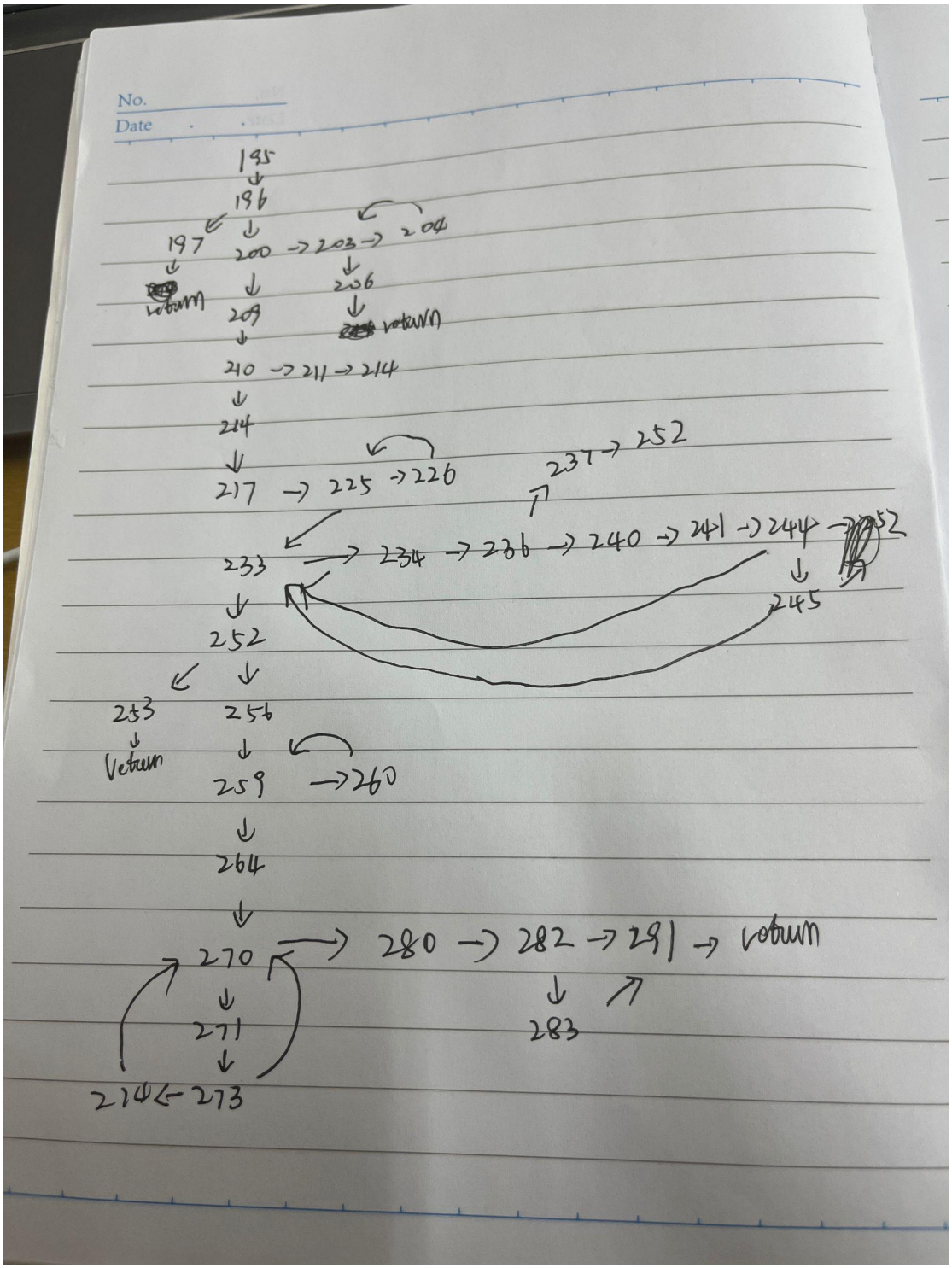
```java
        try {
          Func.graphvizShow(dotContent);
        } catch (IOException | InterruptedException e) {
          throw new RuntimeException(e);
        }
      }

      return pathBuilder.toString();
    }
```

输入参数列表	参数名	含义	数据类型
	word1	起始节点单词	字符串
	word2	目标节点单词	字符串
输出参数	含义		数据类型
	最短路径描述字符串		字符串
代码总行数	99		
包含的循环数	6		
包含的判定数	8		

6.2 程序流程图


Start
Get startNode
Is word2 empty?
Yes
No
Get all nodes
Get endNode
For each node
Calc shortest path
Set endNode
Return result
Calc shortest path
Append result
After all nodes
Return result
End

6.3 控制流图



6.4 圈复杂度计算与基本路径识别

圈复杂度为：（请给出计算过程）

节点数量 36

边数量 47

圈复杂度 13

基本路径 1：195->196->197(起始节点不存在)

基本路径 2：195->196->200->203->204->252->203->206(目标词为空）

基本路径 3：195->196->200->210->211（目标词不存在）

基本路径 4：(目标节点前驱节点为空）

195->196->200->214->217->225->226->225->230->233->234->236->240->241->244->245->233->234->236->237->252->253

基本路径 5:(一般情况）

195->196->200->214->217->225->226->225->230->233->234->236->240->241->244->245->233->234->236->237->252->256->259->260->259->264->270->271->273->274->270->280->282->283->291

注意：各基本路径要使用 6.1 节表格里给出的行号。

6.5 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.	gjh Explore	"The word "gjh" is not in the graph.	1
2.	to	There is no path from "to" to "gjhgjh". There is no path from "to" to "to". Shortest path from "to" to "explore":to → 1explore Total weight of the path: 1 Shortest path from "to" to "strange":to → 1explore → 1strange Total weight of the path: 2 Shortest path from "to" to "new":to → 1explore → 1strange → 1new Total weight of the path: 3 Shortest path from "to" to "worlds":to → 1explore → 1strange → 1new → 1worlds Total weight of the path: 4 Shortest path from "to" to "seek":to → 1seek Total weight of the path: 1 Shortest path from "to" to "out":to → 1seek → 1out Total weight of the path: 2 Shortest path from "to" to "life":to → 1explore → 1strange → 1new → 1life	2

		Total weight of the path: 4 Shortest path from "to" to "and":to → 1explore → 1strange → 1new → 1life → 1and Total weight of the path: 5 Shortest path from "to" to "civilizations":to → 1explore → 1strange → 1new → 1civilizations Total weight of the path: 4 Shortest path from "to" to "boldly":to → 2boldly Total weight of the path: 2 Shortest path from "to" to "go":to → 2boldly → 2go Total weight of the path: 4 Shortest path from "to" to "where":to → 2boldly → 2go → 2where Total weight of the path: 6 Shortest path from "to" to "no":to → 2boldly → 2go → 2where → 2no Total weight of the path: 8 Shortest path from "to" to "one":to → 2boldly → 2go → 2where → 2no → 2one Total weight of the path: 10 Shortest path from "to" to "has":to → 2boldly → 2go → 2where → 2no → 2one → 2has Total weight of the path: 12 Shortest path from "to" to "gone":to → 2boldly → 2go → 2where → 2no → 2one → 2has → 2gone Total weight of the path: 14 Shortest path from "to" to "before":to → 2boldly → 2go → 2where → 2no → 2one → 2has → 2gone → 2before Total weight of the path: 16 Shortest path from "to" to "dream":to → 1dream Total weight of the path: 1 Shortest path from "to" to "big":to → 1dream → 1big Total weight of the path: 2 Shortest path from "to" to "dreams":to → 1dream → 1big → 1dreams Total weight of the path: 3 Shortest path from "to" to "chase":to → 1chase Total weight of the path: 1 Shortest path from "to" to "after":to → 1chase → 1after Total weight of the path: 2 Shortest path from "to" to "the":to → 1exercise → 1the	

		Total weight of the path: 2 Shortest path from "to" to "unknown":to → 1exercise → 1the → 2unknown Total weight of the path: 4 Shortest path from "to" to "discover":to → 1discover Total weight of the path: 1 Shortest path from "to" to "mysteries":to → 1exercise → 1the → 1mysteries Total weight of the path: 3 Shortest path from "to" to "of":to → 1exercise → 1the → 1mysteries → 1of Total weight of the path: 4 Shortest path from "to" to "universe":to → 1exercise → 1the → 1universe Total weight of the path: 3 Shortest path from "to" to "exercise":to → 1exercise Total weight of the path: 1 Shortest path from "to" to "imagination":to → 1exercise → 1the → 1imagination Total weight of the path: 3 Shortest path from "to" to "expand":to → 1exercise → 1the → 1imagination → 1expand Total weight of the path: 4 Shortest path from "to" to "consciousness":to → 1exercise → 1the → 1imagination → 1expand → 1consciousness Total weight of the path: 5 Shortest path from "to" to "embrace":to → 1explore → 1strange → 1new → 1life → 1and → 1embrace Total weight of the path: 6	
3.	to gjh	The word "gjh" is not in the graph.	3
4.	to gjhgjh	There is no path from "to" to "gjhgjh".	4
5.	to strange	Shortest path from "to" to "strange":to → 1explore → 1strange Total weight of the path: 2	5

6.6 JUnit 测试代码

针对 6.5 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.	package org.example;
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	import org.example.util.Func;
	import java.util.List;
	
	public class MainTest2 {
	
	@Test
	public void test1() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String word1 = "gjh";
	String word2 = "explore";
	String expected = "The word \"gjh\" is not in the graph.";
	String result = org.example.Main.calcShortestPath(word1, word2);
	assertEquals(expected, result); }
	}

2.	package org.example;
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	import org.example.util.Func;
	import java.util.List;
	
	public class MainTest2 {
	
	@Test
	public void test2() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String word1 = "to";
	String word2 = "";
	String expected = "There is no path from \"to\" to \"gjhgjh\".\n" +
	"There is no path from \"to\" to \"to\".\n" +
	"Shortest path from \"to\" to \"explore\":to → 1explore\n" +
	"Total weight of the path: 1\n" +

```
"Shortest path from \"to\" to \"strange\":to → 1explore → 1strange\n" +
"Total weight of the path: 2\n" +
"Shortest path from \"to\" to \"new\":to → 1explore → 1strange → 1new\n" +
"Total weight of the path: 3\n" +
"Shortest path from \"to\" to \"worlds\":to → 1explore → 1strange → 1new → 1worlds\n" +
"Total weight of the path: 4\n" +
"Shortest path from \"to\" to \"seek\":to → 1seek\n" +
"Total weight of the path: 1\n" +
"Shortest path from \"to\" to \"out\":to → 1seek → 1out\n" +
"Total weight of the path: 2\n" +
"Shortest path from \"to\" to \"life\":to → 1explore → 1strange → 1new → 1life\n" +
"Total weight of the path: 4\n" +
"Shortest path from \"to\" to \"and\":to → 1explore → 1strange → 1new → 1life → 1and\n" +
"Total weight of the path: 5\n" +
"Shortest path from \"to\" to \"civilizations\":to → 1explore → 1strange → 1new → 1civilizations\n" +
"Total weight of the path: 4\n" +
"Shortest path from \"to\" to \"boldly\":to → 2boldly\n" +
"Total weight of the path: 2\n" +
"Shortest path from \"to\" to \"go\":to → 2boldly → 2go\n" +
"Total weight of the path: 4\n" +
"Shortest path from \"to\" to \"where\":to → 2boldly → 2go → 2where\n" +
"Total weight of the path: 6\n" +
"Shortest path from \"to\" to \"no\":to → 2boldly → 2go → 2where → 2no\n" +
"Total weight of the path: 8\n" +
"Shortest path from \"to\" to \"one\":to → 2boldly → 2go → 2where → 2no → 2one\n" +
"Total weight of the path: 10\n" +
"Shortest path from \"to\" to \"has\":to → 2boldly → 2go → 2where → 2no → 2one → 2has\n" +
"Total weight of the path: 12\n" +
"Shortest path from \"to\" to \"gone\":to → 2boldly → 2go → 2where → 2no → 2one → 2has → 2gone\n" +
"Total weight of the path: 14\n" +
"Shortest path from \"to\" to \"before\":to → 2boldly → 2go → 2where → 2no → 2one → 2has → 2gone → 2before\n" +
"Total weight of the path: 16\n" +
```

```
                "Shortest path from \"to\" to \"dream\":to → 1dream\n" +
                "Total weight of the path: 1\n" +
                "Shortest path from \"to\" to \"big\":to → 1dream → 1big\n" +
                "Total weight of the path: 2\n" +
                "Shortest path from \"to\" to \"dreams\":to → 1dream → 1big → 1dreams\n" +
                "Total weight of the path: 3\n" +
                "Shortest path from \"to\" to \"chase\":to → 1chase\n" +
                "Total weight of the path: 1\n" +
                "Shortest path from \"to\" to \"after\":to → 1chase → 1after\n" +
                "Total weight of the path: 2\n" +
                "Shortest path from \"to\" to \"the\":to → 1exercise → 1the\n" +
                "Total weight of the path: 2\n" +
                "Shortest path from \"to\" to \"unknown\":to → 1exercise → 1the → 2unknown\n" +
                "Total weight of the path: 4\n" +
                "Shortest path from \"to\" to \"discover\":to → 1discover\n" +
                "Total weight of the path: 1\n" +
                "Shortest path from \"to\" to \"mysteries\":to → 1exercise → 1the → 1mysteries\n" +
                "Total weight of the path: 3\n" +
                "Shortest path from \"to\" to \"of\":to → 1exercise → 1the → 1mysteries → 1of\n" +
                "Total weight of the path: 4\n" +
                "Shortest path from \"to\" to \"universe\":to → 1exercise → 1the → 1universe\n" +
                "Total weight of the path: 3\n" +
                "Shortest path from \"to\" to \"exercise\":to → 1exercise\n" +
                "Total weight of the path: 1\n" +
                "Shortest path from \"to\" to \"imagination\":to → 1exercise → 1the → 1imagination\n" +
                "Total weight of the path: 3\n" +
                "Shortest path from \"to\" to \"expand\":to → 1exercise → 1the → 1imagination → 1expand\n" +
                "Total weight of the path: 4\n" +
                "Shortest path from \"to\" to \"consciousness\":to → 1exercise → 1the → 1imagination → 1expand → 1consciousness\n" +
                "Total weight of the path: 5\n" +
                "Shortest path from \"to\" to \"embrace\":to → 1explore → 1strange → 1new → 1life → 1and → 1embrace\n" +
                "Total weight of the path: 6\n";
        String result =    org.example.Main.calcShortestPath(word1, word2);
        assertEquals(expected, result);        }
```

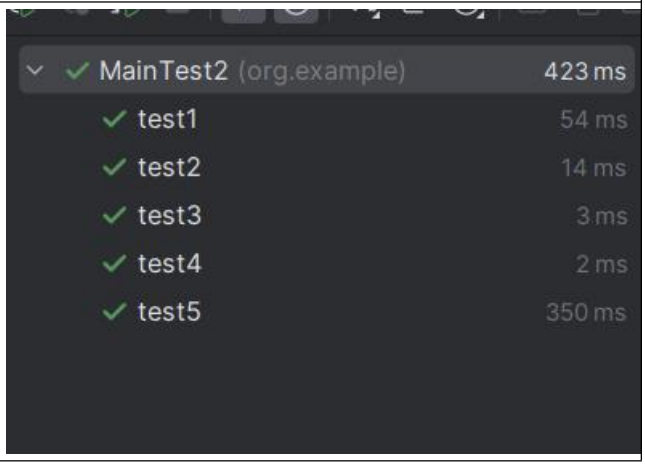
	}
	package org.example;
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	import org.example.util.Func;
	import java.util.List;
	
	public class MainTest2 {
	@Test
	public void test3() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String word1 = "to";
	String word2 = "gjh";
	String expected = "The word \"gjh\" is not in the graph.";
	String result = org.example.Main.calcShortestPath(word1, word2);
	assertEquals(expected, result); }
	
	}

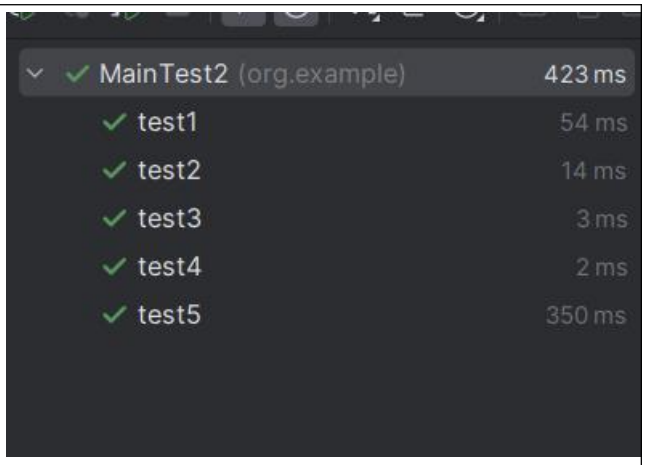
4.	package org.example;
	import org.junit.Test;
	import static org.junit.Assert.assertEquals;
	import org.example.util.Func;
	import java.util.List;
	
	public class MainTest2 {
	@Test
	public void test4() {
	Main main = new Main();
	String str = Func.getStr("1.txt");
	List<String> wordList = Func.getWords(str);
	main.graph = Func.buildDirectedGraph(wordList);
	String word1 = "to";
	String word2 = "gjhgjh";
	String expected = "There is no path from \"to\" to \"gjhgjh\".";
	String result = org.example.Main.calcShortestPath(word1, word2);
	assertEquals(expected, result);
	}
	}

5.	package org.example; import org.junit.Test;

```
import static org.junit.Assert.assertEquals;
import org.example.util.Func;
import java.util.List;

public class MainTest2 {
@Test
public void test5() {
    Main main = new Main();
    String str = Func.getStr("1.txt");
    List<String> wordList = Func.getWords(str);
    main.graph = Func.buildDirectedGraph(wordList);
    String word1 = "to";
    String word2 = "strange";
    String expected = "Shortest path from \"to\" to \"strange\":to → 1explore → 1strange\nTotal weight of the path: 2";

    String result =  org.example.Main.calcShortestPath(word1, word2);
    assertEquals(expected, result);
    }
}
```

6.7 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.	"The word "gjh" is not in the graph.	"The word "gjh" is not in the graph.	

2.	There is no path from "to" to "gjhgjh". There is no path from "to" to "to". Shortest path from "to" to "explore":to → 1explore Total weight of the path: 1 Shortest path from "to" to "strange":to → 1explore → 1strange Total weight of the path: 2 Shortest path from "to" to "new":to → 1explore → 1strange → 1new Total weight of the path: 3 Shortest path from "to" to "worlds":to → 1explore → 1strange → 1new → 1worlds Total weight of the path: 4 Shortest path from "to" to "seek":to → 1seek Total weight of the path: 1 Shortest path from "to" to	There is no path from "to" to "gjhgjh". There is no path from "to" to "to". Shortest path from "to" to "explore":to → 1explore Total weight of the path: 1 Shortest path from "to" to "strange":to → 1explore → 1strange Total weight of the path: 2 Shortest path from "to" to "new":to → 1explore → 1strange → 1new Total weight of the path: 3 Shortest path from "to" to "worlds":to → 1explore → 1strange → 1new → 1worlds Total weight of the path: 4 Shortest path from "to" to "seek":to → 1seek Total weight of the path: 1 Shortest path from "to" to	

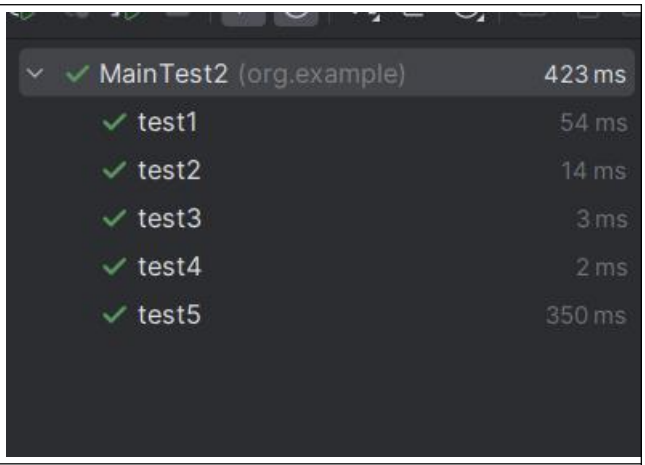
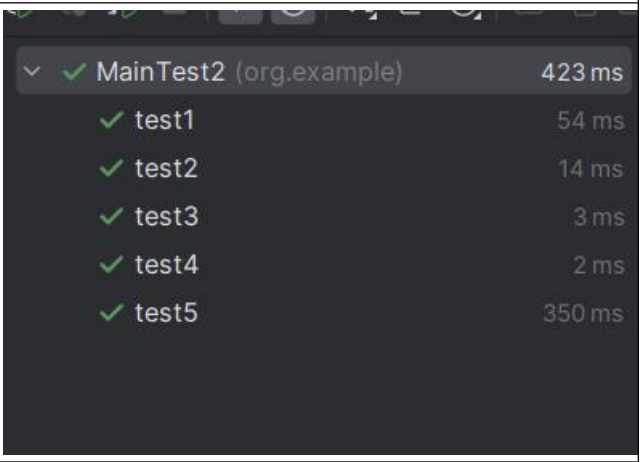
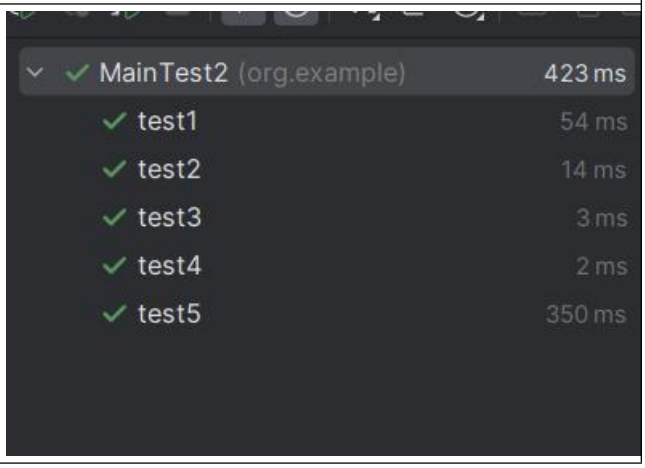
	"out":to → 1seek → 1out Total weight of the path: 2 Shortest path from "to" to "life":to → 1explore → 1strange → 1new → 1life Total weight of the path: 4 Shortest path from "to" to "and":to → 1explore → 1strange → 1new → 1life → 1and Total weight of the path: 5 Shortest path from "to" to "civilizations":to → 1explore → 1strange → 1new → 1civilizations Total weight of the path: 4 Shortest path from "to" to "boldly":to → 2boldly Total weight of the path: 2 Shortest path from "to" to "go":to → 2boldly → 2go Total weight of the path: 4 Shortest path from "to" to	"out":to → 1seek → 1out Total weight of the path: 2 Shortest path from "to" to "life":to → 1explore → 1strange → 1new → 1life Total weight of the path: 4 Shortest path from "to" to "and":to → 1explore → 1strange → 1new → 1life → 1and Total weight of the path: 5 Shortest path from "to" to "civilizations":to → 1explore → 1strange → 1new → 1civilizations Total weight of the path: 4 Shortest path from "to" to "boldly":to → 2boldly Total weight of the path: 2 Shortest path from "to" to "go":to → 2boldly → 2go Total weight of the path: 4 Shortest path from "to" to	

	"where":to → 2boldly → 2go → 2where Total weight of the path: 6 Shortest path from "to" to "no":to → 2boldly → 2go → 2where → 2no Total weight of the path: 8 Shortest path from "to" to "one":to → 2boldly → 2go → 2where → 2no → 2one Total weight of the path: 10 Shortest path from "to" to "has":to → 2boldly → 2go → 2where → 2no → 2one → 2has Total weight of the path: 12 Shortest path from "to" to "gone":to → 2boldly → 2go → 2where → 2no → 2one → 2has → 2gone Total weight of the path: 14 Shortest path from "to" to "before":to → 2boldly → 2go	"where":to → 2boldly → 2go → 2where Total weight of the path: 6 Shortest path from "to" to "no":to → 2boldly → 2go → 2where → 2no Total weight of the path: 8 Shortest path from "to" to "one":to → 2boldly → 2go → 2where → 2no → 2one Total weight of the path: 10 Shortest path from "to" to "has":to → 2boldly → 2go → 2where → 2no → 2one → 2has Total weight of the path: 12 Shortest path from "to" to "gone":to → 2boldly → 2go → 2where → 2no → 2one → 2has → 2gone Total weight of the path: 14 Shortest path from "to" to "before":to → 2boldly → 2go	

	→ 2where → 2no → 2one → 2has → 2gone → 2before Total weight of the path: 16 Shortest path from "to" to "dream":to → 1dream Total weight of the path: 1 Shortest path from "to" to "big":to → 1dream → 1big Total weight of the path: 2 Shortest path from "to" to "dreams":to → 1dream → 1big → 1dreams Total weight of the path: 3 Shortest path from "to" to "chase":to → 1chase Total weight of the path: 1 Shortest path from "to" to "after":to → 1chase → 1after Total weight of the path: 2 Shortest path from "to" to "the":to →	→ 2where → 2no → 2one → 2has → 2gone → 2before Total weight of the path: 16 Shortest path from "to" to "dream":to → 1dream Total weight of the path: 1 Shortest path from "to" to "big":to → 1dream → 1big Total weight of the path: 2 Shortest path from "to" to "dreams":to → 1dream → 1big → 1dreams Total weight of the path: 3 Shortest path from "to" to "chase":to → 1chase Total weight of the path: 1 Shortest path from "to" to "after":to → 1chase → 1after Total weight of the path: 2 Shortest path from "to" to "the":to →	

	1exercise → 1the Total weight of the path: 2 Shortest path from "to" to "unknown":to → 1exercise → 1the → 2unknown Total weight of the path: 4 Shortest path from "to" to "discover":to → 1discover Total weight of the path: 1 Shortest path from "to" to "mysteries":to → 1exercise → 1the → 1mysteries Total weight of the path: 3 Shortest path from "to" to "of":to → 1exercise → 1the → 1mysteries → 1of Total weight of the path: 4 Shortest path from "to" to "universe":to → 1exercise → 1the → 1universe Total weight of the path: 3 Shortest path	1exercise → 1the Total weight of the path: 2 Shortest path from "to" to "unknown":to → 1exercise → 1the → 2unknown Total weight of the path: 4 Shortest path from "to" to "discover":to → 1discover Total weight of the path: 1 Shortest path from "to" to "mysteries":to → 1exercise → 1the → 1mysteries Total weight of the path: 3 Shortest path from "to" to "of":to → 1exercise → 1the → 1mysteries → 1of Total weight of the path: 4 Shortest path from "to" to "universe":to → 1exercise → 1the → 1universe Total weight of the path: 3 Shortest path	

	from "to" to "exercise":to → 1exercise Total weight of the path: 1 Shortest path from "to" to "imagination":to → 1exercise → 1the → 1imagination Total weight of the path: 3 Shortest path from "to" to "expand":to → 1exercise → 1the → 1imagination → 1expand Total weight of the path: 4 Shortest path from "to" to "consciousness":to → 1exercise → 1the → 1imagination → 1expand → 1consciousness Total weight of the path: 5 Shortest path from "to" to "embrace":to → 1explore → 1strange → 1new → 1life → 1and → 1embrace Total weight of the path: 6	from "to" to "exercise":to → 1exercise Total weight of the path: 1 Shortest path from "to" to "imagination":to → 1exercise → 1the → 1imagination Total weight of the path: 3 Shortest path from "to" to "expand":to → 1exercise → 1the → 1imagination → 1expand Total weight of the path: 4 Shortest path from "to" to "consciousness":to → 1exercise → 1the → 1imagination → 1expand → 1consciousness Total weight of the path: 5 Shortest path from "to" to "embrace":to → 1explore → 1strange → 1new → 1life → 1and → 1embrace Total weight of the path: 6	

3.	The word "gjh" is not in the graph.	The word "gjh" is not in the graph.	
4.	There is no path from "to" to "gjhgjh".	There is no path from "to" to "gjhgjh".	
5.	Shortest path from "to" to "strange":to → 1explore → 1strange Total weight of the path: 2	Shortest path from "to" to "strange":to → 1explore → 1strange Total weight of the path: 2	

6.8 代码覆盖度分析

由于是对整个 main 类进行测试，但只测试了算最短路径函数，所以覆盖率比较低，但可以看到对函数的覆盖率很高

```
public static String calcShortestPath(String
    Node startNode = graph.getNode(word1); // 获
    if (startNode == null) { // 如果起始节点不存在
        return "The word \"" + word1 + "\" is not
    }

    if (Objects.equals(word2, b: "")) { // 如果目
        StringBuilder result = new StringBuilder(
        List<Node> nodes = graph.getNodes(); // 获
        for (Node endNode : nodes) { // 遍历所有节点
            result.append(shortestPath(startNode, e
        }
        return result.toString(); // 返回结果字符串
    }

    Node endNode = graph.getNode(word2); // 获取
    if (endNode == null) { // 如果目标节点不存在
        return "The word \"" + word2 + "\" is not
    }

    return shortestPath(startNode, endNode, sho
}

2 usages  hitgjh *
private static String shortestPath(Node start
    String startWord = startNode.getWord(); //
    String endWord = endNode.getWord(); // 获取目

    PriorityQueue<Node> queue = new PriorityQue
    Map<Node, Integer> distances = new HashMap<
```

```
232                                                                    ⚠ 32
233         while (!queue.isEmpty()) { // 当优先队列不为空时循环
234             Node currentNode = queue.poll(); // 取出优先队列中的当前节点
235
236             if (currentNode.equals(endNode)) { // 如果当前节点是目标节点，则
237                 break;
238             }
239
240             for (Node neighbor : graph.getNeighbors(currentNode)) { //
241                 int weight = graph.getEdgeWeight(currentNode, neighbor);
242                 int distanceThroughCurrent = distances.get(currentNode) +
243
244                 if (distanceThroughCurrent < distances.get(neighbor)) {
245                     distances.put(neighbor, distanceThroughCurrent); // 更新
246                     previousNodes.put(neighbor, currentNode); // 更新前驱节点
247                     queue.add(neighbor); // 将邻居节点加入队列
248                 }
249             }
250         }
251
252         if (previousNodes.get(endNode) == null) { // 如果目标节点的前驱节
253             return "There is no path from \"" + startWord + "\" to \""
254         }
255
256         List<Node> shortestPath = new ArrayList<>(); // 创建最短路径列表
257         Node currentNode = endNode; // 初始化当前节点为目标节点
258
259         while (currentNode != null) { // 循环直到当前节点为空
260             shortestPath.add(currentNode); // 将当前节点加入最短路径列表
261             currentNode = previousNodes.get(currentNode); // 获取当前节点
```

```java
        Collections.reverse(shortestPath); // 反转最短路径列表，使

        int totalWeight = 0; // 初始化总权重为0
        StringBuilder pathBuilder = new StringBuilder("Shortest path from
                + startWord + "\" to \"" + endWord + "\":"); // 创建路径字符

        for (int i = 0; i < shortestPath.size(); i++) { // 遍历最短路径列表
            pathBuilder.append(shortestPath.get(i).getWord()); // 将节点的词加

            if (i < shortestPath.size() - 1) { // 如果不是最后一个节点
                int weight = graph.getEdgeWeight(shortestPath.get(i), shortest
                totalWeight += weight; // 更新总权重
                pathBuilder.append(" → ").append(weight); // 将边的权重加入路径字符
            }
        }

        pathBuilder.append("\nTotal weight of the path: ").append(totalWei

        if (showGraph) { // 如果需要展示图形
            String dotContent = graph.toDigraphStringPath(shortestPath); //
            try {
                Func.graphvizShow(dotContent); // 使用图形库展示图形
            } catch (IOException | InterruptedException e) {
                throw new RuntimeException(e); // 捕获可能的异常并抛出运行时异常
            }
        }

        return pathBuilder.toString(); // 返回最短路径的字符串表示
    }
    /**
     * @Author xukeyan
```

Element	Class, %	Method, %	Line, %
org.example	100% (1/1)	37% (3/8)	32% (61/190)
Main	100% (1/1)	37% (3/8)	32% (61/190)

6.9 未通过测试的原因分析及代码修改

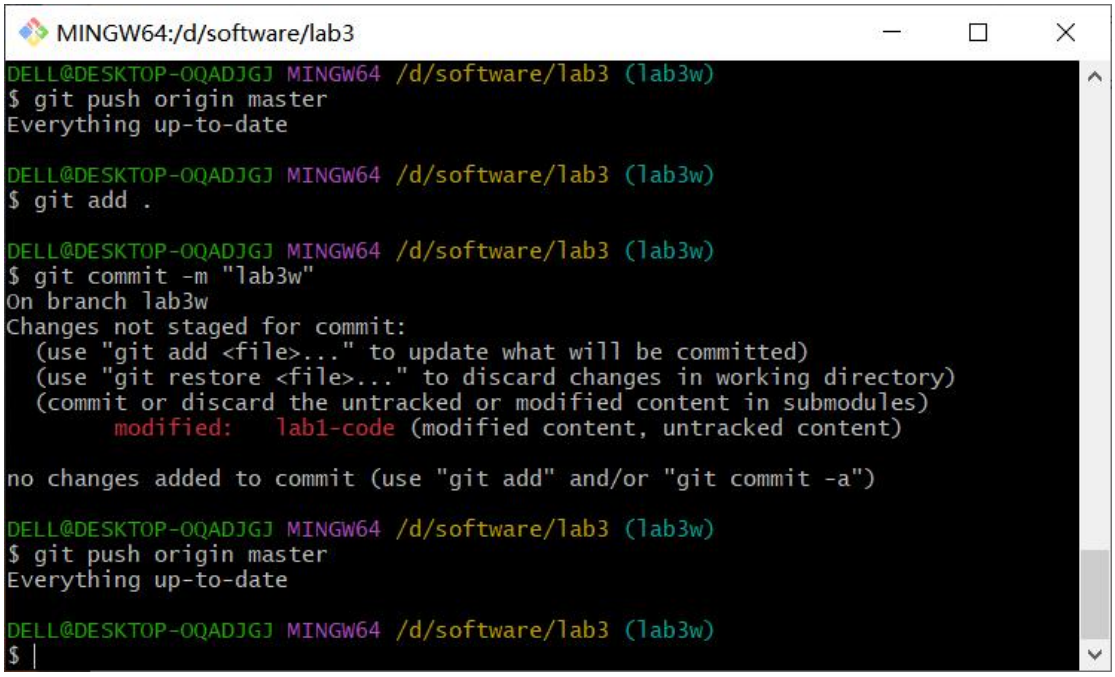
分析自己的 Lab1 代码为何未通过 6.7 节表格中某些测试用例的原因，并通过修改代码消除此类 BUG。必要时给出修改后的代码。

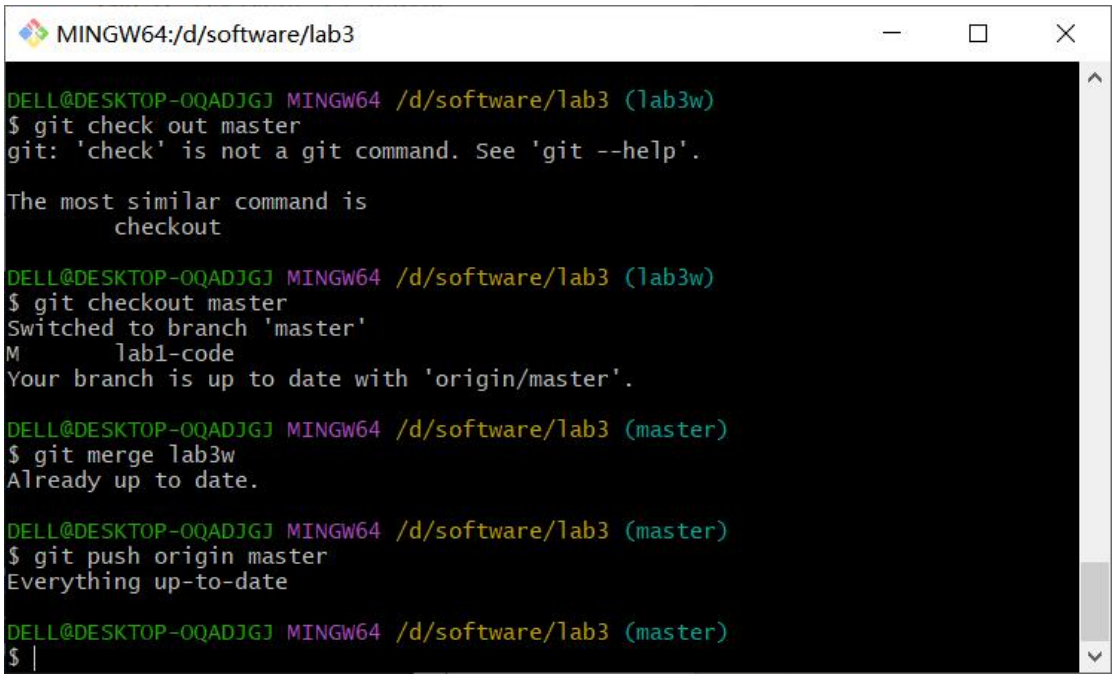
若 6.7 节表格中没有未通过的测试用例，本节可空。

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.			
2.			
3.			
4.			

6.10 Git 操作记录





7 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
进行代码评审	180	200	代码出现较多不规范，进行了修改
完成黑盒测试	150	150	按时完成
完成白盒测试	200	150	由于有黑盒测试的基础，编写测试代码耗时较少

8 小结

(1) 代码评审

在本次实验中，我们对 Lab1 所完成的代码进行了全面的代码评审（走查），重点从以下几个方面进行了评价：

代码规范性：总体上，代码命名规范，结构清晰，但部分注释较少，某些异常处理可以进一步完善。

代码正确性：算法逻辑基本正确，能够正确处理各种输入，但在极端边界条件和特殊输入的处理上存在改进空间。

通过这次评审，我们确定了代码的优点和需要改进的地方，并为后续优化提供了方向。

(2) 测试设计与执行

在测试设计与执行部分，我们完成了以下任务：

黑盒测试：设计了若干测试用例，涵盖了正常路径、无路径和边界条件等多种情况，确保了功能的正确性。

白盒测试：根据代码结构设计测试用例，保证了代码的高覆盖率，包括语句覆盖和分支覆盖。

具体步骤如下：

编写 JUnit 测试代码：在 JUnit 环境下撰写测试代码，验证各种输入情况下的输出是否符合预期。

执行测试：通过 JUnit 运行测试用例，验证代码功能是否正确。

统计覆盖率：使用 IDE 自带工具统计测试覆盖率，确保测试覆盖了代码的主要部分。

通过上述步骤，我们不仅验证了代码的功能正确性，还提高了代码的可靠性和稳定性。覆盖率统计结果表明，测试覆盖了绝大部分代码分支和语句，达到了预期的测试效果。

(3) 总结

本次实验不仅对现有代码进行了全面评审，找出了其中的优点和不足之处，还通过系统的测试设计与执行，提高了代码的可靠性和健壮性。通过这次实验，我们深刻理解了代码规范性的重要性，以及全面测试对保证代码质量的关键作用。