



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2024 年春季学期 计算学部《软件工程》课程

Lab 1 实验报告

姓名	班级/学号	联系方式
徐柯炎	2103602/2021110683	15189789979
高浚豪	2103602/2021111700	13359705210

目 录

1	实验要求.....	1
1.1	结对编程.....	1
1.2	Git 实战.....	1
2	待求解问题描述.....	1
2.1	待求解问题 1: 读入文本并生成有向图.....	1
2.2	待求解问题 2: 查询桥接词 (bridge words)	1
2.3	待求解问题 3: 根据 bridge word 生成新文本.....	1
2.4	待求解问题 4: 计算两个单词之间的最短路径.....	2
2.5	待求解问题 5: 随机游走.....	2
3	算法与数据结构设计.....	2
3.1	设计思路与算法流程图.....	2
3.1.1	根据文本生成图.....	2
3.1.2	展示图.....	2
3.1.3	查询桥接词.....	3
3.1.4	根据桥接词生成新文本.....	3
3.1.5	计算最短路径.....	4
3.1.6	随机游走.....	5
3.2	数据结构设计.....	6
3.2.1	Node.....	6
3.2.2	DirectedGraph.....	7
3.3	算法时间复杂度分析.....	7
3.3.1	文本生成图.....	7
3.3.2	展示图.....	7
3.3.3	查询桥接词.....	7
3.3.4	根据桥接词生成新文本.....	7
3.3.5	计算最短路径.....	8
3.3.6	随机游走.....	8
4	实验与测试.....	8
4.1	读取文本文件并展示有向图.....	8
4.2	查询桥接词.....	10
4.3	根据桥接词生成新文本.....	11
4.4	计算最短路径.....	11
4.5	随机游走.....	14
5	编程语言与开发环境.....	15
6	结对编程.....	15
6.1	分组依据.....	15
6.2	角色切换与任务分工.....	16
6.3	工作照片.....	16
6.4	工作日志.....	17
7	Git 操作过程.....	17
7.1	实验场景(1): 仓库创建与提交.....	17
7.2	实验场景(2): 分支管理.....	20

8	在 IDE 中使用 Git Plugin	24
9	小结.....	26

1 实验要求

1. 结对编程

- 两人一组，自由组合；
- 使用一台计算机，共同编码，完成实验要求；
- 在工作期间，两人的角色至少切换 6 次；
- 选择一门支持面向对象的编程语言，推荐 Java。

2. Git 实战

- 熟练掌握 Git 的基本指令和分支管理指令；
- 掌握 Git 支持软件配置管理的核心机理；
- 在实践项目中使用 Git /Github 管理自己的项目源代码；
- 本部分实验由个人单独完成。

2 待求解问题描述

1. 待求解问题 1：读入文本并生成有向图

- **问题描述：**程序读入文本数据，将其转化为有向图。边 $A \rightarrow B$ 的定义如下：两个节点 A,B 之间存在一条边 $A \rightarrow B$ ，意味着在文本中至少有一处位置 A 和 B 相邻出现（即 A 和 B 之间有且仅有 1 或多个空格）。 $A \rightarrow B$ 的边权重为文本中 A 和 B 相邻出现的次数。
- **输入数据：**一个文本文件 (*.txt)。
- **输出数据：**将英文文本以图的数据结构存储；生成的有向图 (*.png)。
- **约束条件：**需让用户或以参数的形式选择文本文件的位置；输入文本为英文书写的文本数据；文本数据中的标点符号当作空格处理；文本中的非字母(A-Z, a-z)字符应被忽略。

2. 待求解问题 2：查询桥接词 (bridge words)

- **问题描述：**输入两个单词，并查询它们的桥接词。word1 和 word2 的桥接词 word3 定义如下：图中存在两条边 $word1 \rightarrow word3$ ， $word3 \rightarrow word2$ 。
- **输入数据：**两个英文单词 word1, word2。
- **输出数据：**如果存在桥接词，则输出“The bridge words from word1 to word2 are: xxx, xxx, and xxx.”；如果不存在桥接词，则输出“No bridge words from word1 to word2!”；如果 word1 或 word2 不存在，则输出“No word1 or word2 in the graph!”。
- **约束条件：**无。

3. 待求解问题 3：根据 bridge word 生成新文本

- **问题描述：**输入一行文本，程序计算文本中两两相邻单词的桥接词，并插入到这两个单词中间，并生成新的文本。
- **输入数据：**一行英文文本。

- **输出数据:** 插入桥接词后的新的文本。
- **约束条件:** 如果两个单词之间没有桥接词, 则不插入任何单词; 如果两个单词之间存在多个桥接词, 则**随机选择**一个插入到文本中。

4. 待求解问题 4: 计算两个单词之间的最短路径

- **问题描述:** 输入两个单词, 计算这两个单词的最短路径。最短路径的定义: 路径上所有边权值之和最小。
- **输入数据:** 两个英文单词。
- **输出数据:** 如果两个单词之间可达, 将最短路径**标注在原图上**并输出路径的长度; 如果两个单词之间不可达, 则输出提示。
- **约束条件:** 若有多条最短路径, 展示一条即可; 若用户只输入一个单词, 则需要计算该单词到任意词的最短路径, 并逐一展示。

5. 待求解问题 5: 随机游走

- **问题描述:** 随机从图中选择一个节点, 并以此节点进行随机遍历。
- **输入数据:** 无。
- **输出数据:** 随机游走生成的文本, 并写入文件。
- **约束条件:** 停止遍历的条件: 出现第一条**重复**的边, 或者进入的某个节点不存在出边; 用户可以**随时**停止遍历。

3 算法与数据结构设计

3.1 设计思路与算法流程图

1. 根据文本生成图

- **算法设计思路:**
 - 1) 读取文本文件, 将其中的内容存储到字符串中;
 - 2) 对此字符串进行处理: 将所有字母转化为小写形式, 去除所有非英文字符, 然后分割字符串;
 - 3) 遍历字符串, 建立单词和节点的映射关系;
 - 4) 初始化邻接矩阵为零矩阵;
 - 5) 遍历字符串, 将两两相邻的 word 映射为边, 并更新邻接矩阵的权值。
- **流程图:**



2. 展示图

- **算法设计思路:**
 - 1) 利用生成的邻接矩阵, 将此图转化为 dot 语法, 并将其存储到 graph.dot 文件中;
 - 2) 使用 dot -Tpng -o graph.png graph.dot 命令生成 png 图并存储到磁盘;
 - 3) 展示生成的 png 图。

- 流程图:

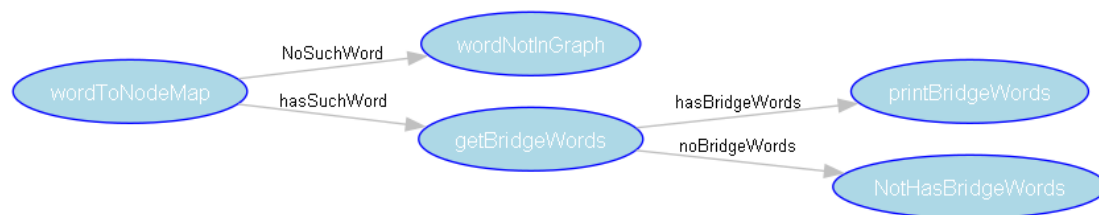


3. 查询桥接词

- 算法设计思路:

- 1) 将输入的两个单词映射到节点上;
- 2) 判断这两个节点是否都不为空, 若有一个为空, 输出提示信息;
- 3) 查询桥接词;
- 4) 如果桥接词列表的大小为 0, 输出提示信息; 如果不为 0, 输出桥接词。

- 流程图:

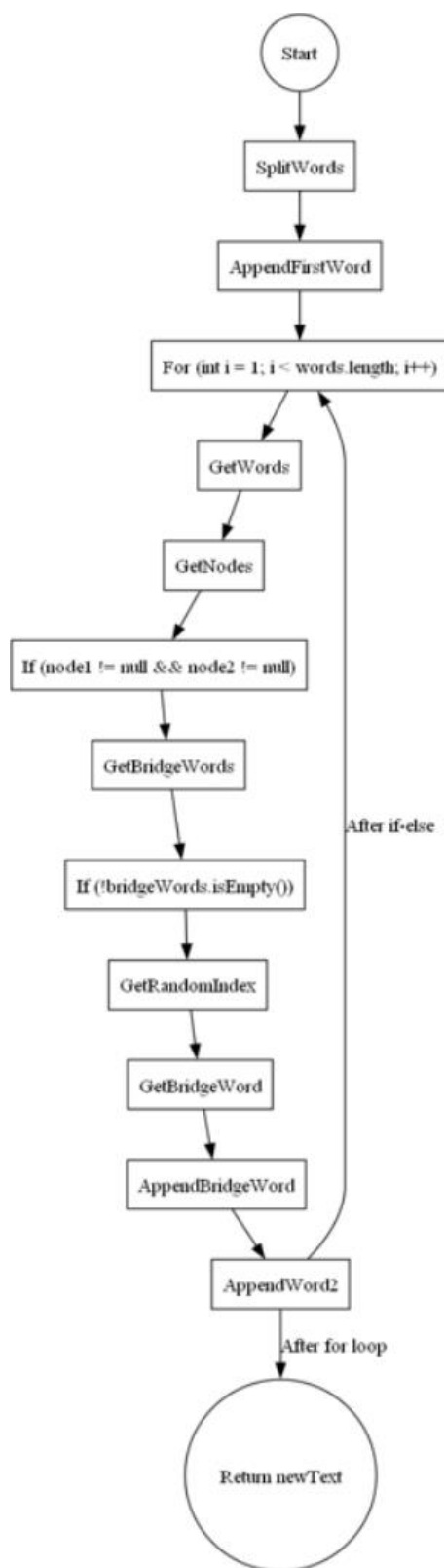


4. 根据桥接词生成新文本

- 算法设计思路:

- 1) 拆分并构建文本
- 2) 遍历并逐个处理每个单词对
- 3) 检查两个单词对应的节点是否存在, 若均存在, 则获取两个节点的桥接词列表
- 4) 如果桥接词列表不为空, 随机选择一个桥接词加入到新文本中并返回新文本

- 流程图:



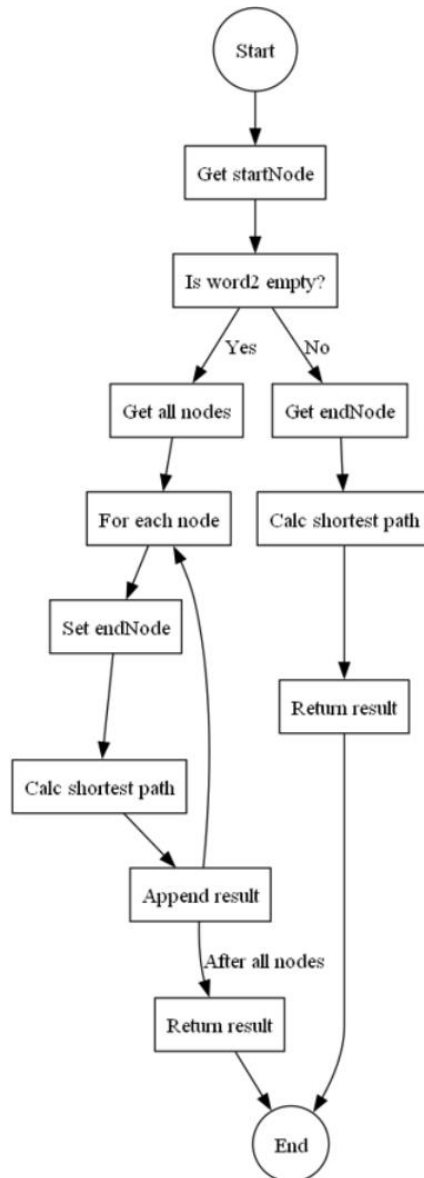
5. 计算最短路径

- 算法设计思路:

- 1) 将输入的两个单词映射到图中的起始节点和目标节点;

- 2) 判断起始节点和目标节点是否存在，若有一个不存在，输出提示信息；
- 3) 使用优先队列（Dijkstra 算法）遍历图，找到从起始节点到目标节点的最短路径；
- 4) 如果没有找到路径，输出提示信息；如果找到路径，重构最短路径并进行显示；
- 5) 返回最短路径的字符串表示。

● 流程图：

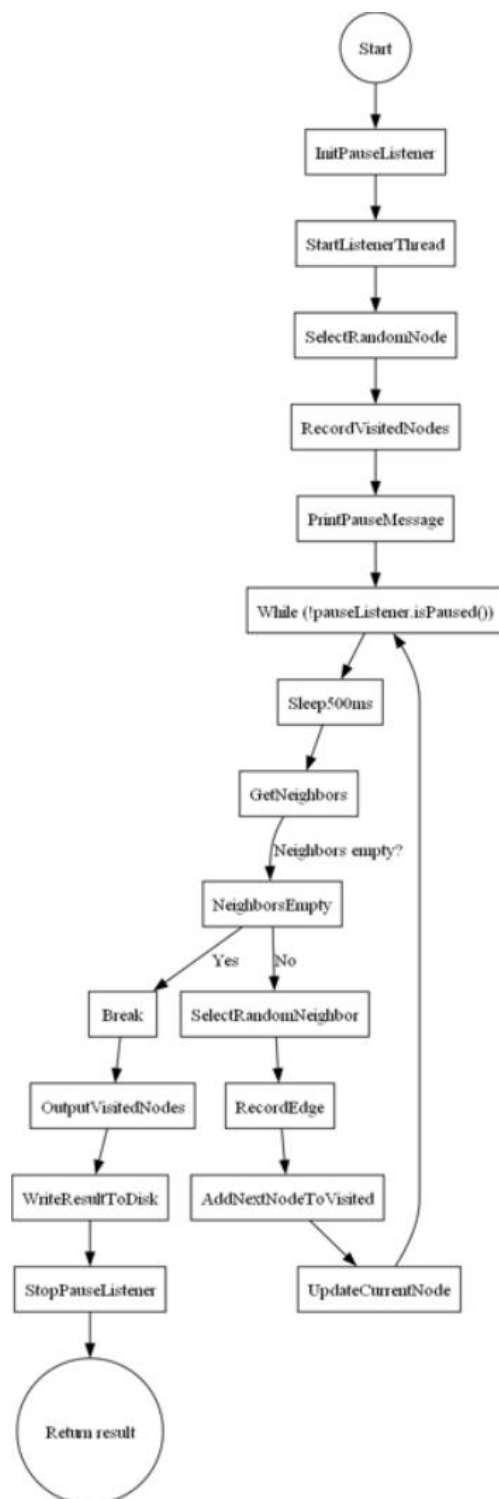


6. 随机游走

● 算法设计思路：

- 1) 随机选择一个起始节点；
- 2) 记录遍历的节点和边；
- 3) 添加起始节点到已访问节点列表；
- 4) 进行随机遍历，直到没有出边或出现重复节点；
- 5) 将遍历的节点和边输出为文本；
- 6) 将结果写入磁盘文件并返回结果。

● 流程图:



3.2 数据结构设计

本实验我们定义了两个数据结构，定义如下：

1. Node

该数据结构表示有向图的节点，包含以下两个部分：

- 1) 该节点的 id;
- 2) 该节点代表的 word。

2. DirectedGraph

该数据结构表示有向图，包含以下五个部分：

- 1) 节点列表 nodes;
- 2) Word 到节点的映射 wordToNodeMap;
- 3) Id 到 word 的映射 idToWordMap;
- 4) 邻接矩阵 adjacencyMatrix;
- 5) 下一个节点的标号 nextId。

3.3 算法时间复杂度分析

1. 文本生成图

假设文本文件有 n 个 word:

- 1) 读取文本文件，此步时间复杂度为 $O(n)$;
- 2) 对此字符串进行处理，此步时间复杂度为 $O(n)$;
- 3) 遍历字符串，建立单词和节点的映射关系，此步时间复杂度为 $O(n)$;
- 4) 初始化邻接矩阵为零矩阵，java 默认初始化为 0，此步时间复杂度为 $O(0)$;
- 5) 遍历字符串，将两两相邻的 word 映射为边，并更新邻接矩阵的权值。此步时间复杂度为 $O(n)$;

综上所述，时间复杂度为 $O(n)$;

2. 展示图

假设有向图有 n 条边:

- 1) 利用生成的邻接矩阵，将此图转化为 dot 语法，此步时间复杂度为 $O(n)$;
- 2) 使用 `dot -Tpng -o graph.png graph.dot` 命令生成 png 图并存储到磁盘; 此步时间复杂度为 $O(n)$;
- 3) 展示生成的 png 图。

综上所述，时间复杂度为 $O(n)$;

3. 查询桥接词

假设有向图有 n 条边，则查询桥接词的过程至多循环 n 次，所以时间复杂度为 $O(n)$;

4. 根据桥接词生成新文本

- 1) 分割输入文本: $O(n)$;
- 2) 遍历单词数组: $O(m)$;
- 3) 每次迭代中:
 - a) 获取节点: $O(1)$
 - b) 获取桥接词: $O(V)$
 - c) 添加桥接词和第二个单词: $O(1)$
 - d) 由于 m 是 words 数组的长度，它与 n 成正比（因为 words 数组的长度取决于输入文本的长度），所以可以认为 $m=O(n)$ 。

因此，循环内每次迭代的时间复杂度为 $O(V)$ ，整个循环的时间复杂度为 $O(n \cdot V)$ 。

综上所述，`generateNewText` 方法的总时间复杂度为 $O(n + n \cdot V)$ ，即 $O(n \cdot V)$ ，其中 n 是输入文本的长度， V 是图中节点的数量。

5. 计算最短路径

- 1) `ShortestPath` 方法中 `dijkstra` 算法：
 - a) 主循环运行次数最多为 $O(V)$ 次；
 - b) 在每次循环中，`poll` 操作的时间复杂度为 $O(\log V)$ ；
 - c) 遍历每个节点的邻居，假设邻居数为 E (边数)，总的时间复杂度为 $O(E)O(E)$ ；
 - d) 更新距离、前驱节点和优先队列，时间复杂度为 $(\log V)$ 。总时间复杂度 $O(V \log V + E)$ ；
 - 2) `ShortestPath` 方法总时间复杂度 $O(V \log V + E)$ ；
 - 3) 输入两个单词，当 `word2` 不为空时，即正常测试两个单词间的最短路径，调用 `ShortestPath` 方法 1 次。总时间复杂度为 $O(V \log V + E)$ ；
 - 4) 当 `word2` 为空时，即测试一个单词到所有词的最短路径，调用 `ShortestPath` 方法 V 次，每次的复杂度为 $O(V \log V + E)$ 。
- 综上所述，总时间复杂度为 $O(V (V \log V + E))$ 。

6. 随机游走

- 1) 获取所有节点： $O(V)$ ；
 - 2) 主循环每次迭代的时间复杂度为 $O(E)$ ，最坏情况下迭代 V 次，时间复杂度为 $O(VE)$ ；
 - 3) 构建结果字符串： $O(V)$ ；
 - 4) 写入文件： $O(V + E)$ 。
- 综上所述，总时间复杂度 $O(VE)$ 。

4 实验与测试

4.1 读取文本文件并展示有向图

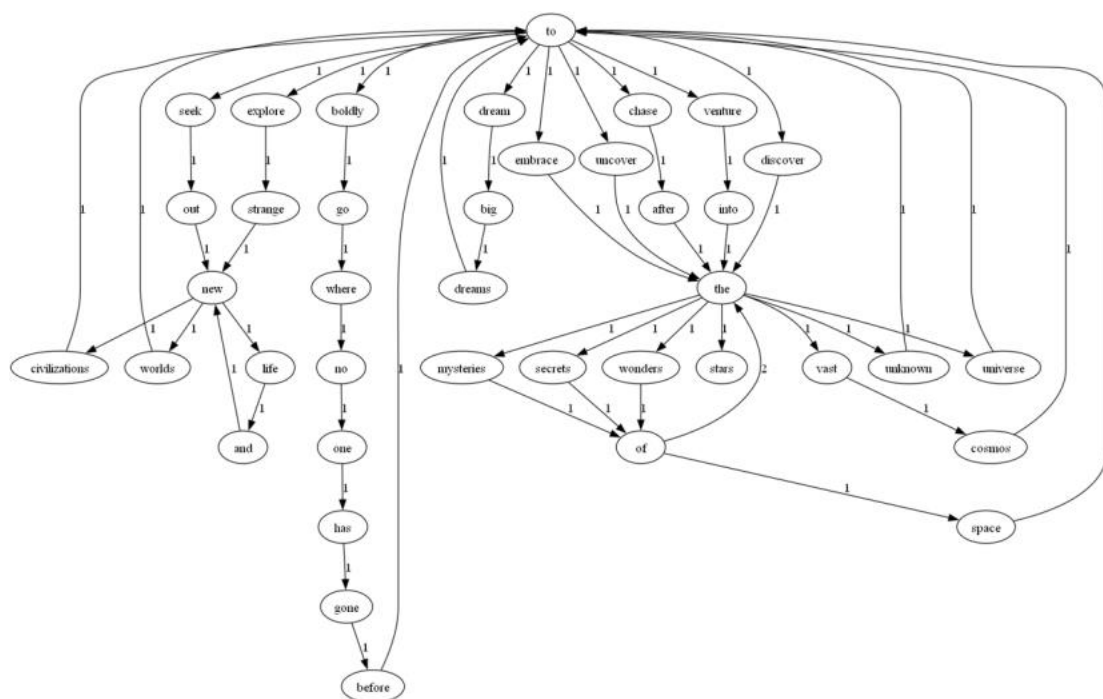
文本文件中包含的内容：

@ To explore strange new worlds, to seek out new life and new * civilizations, to boldly go where no one has gone before.

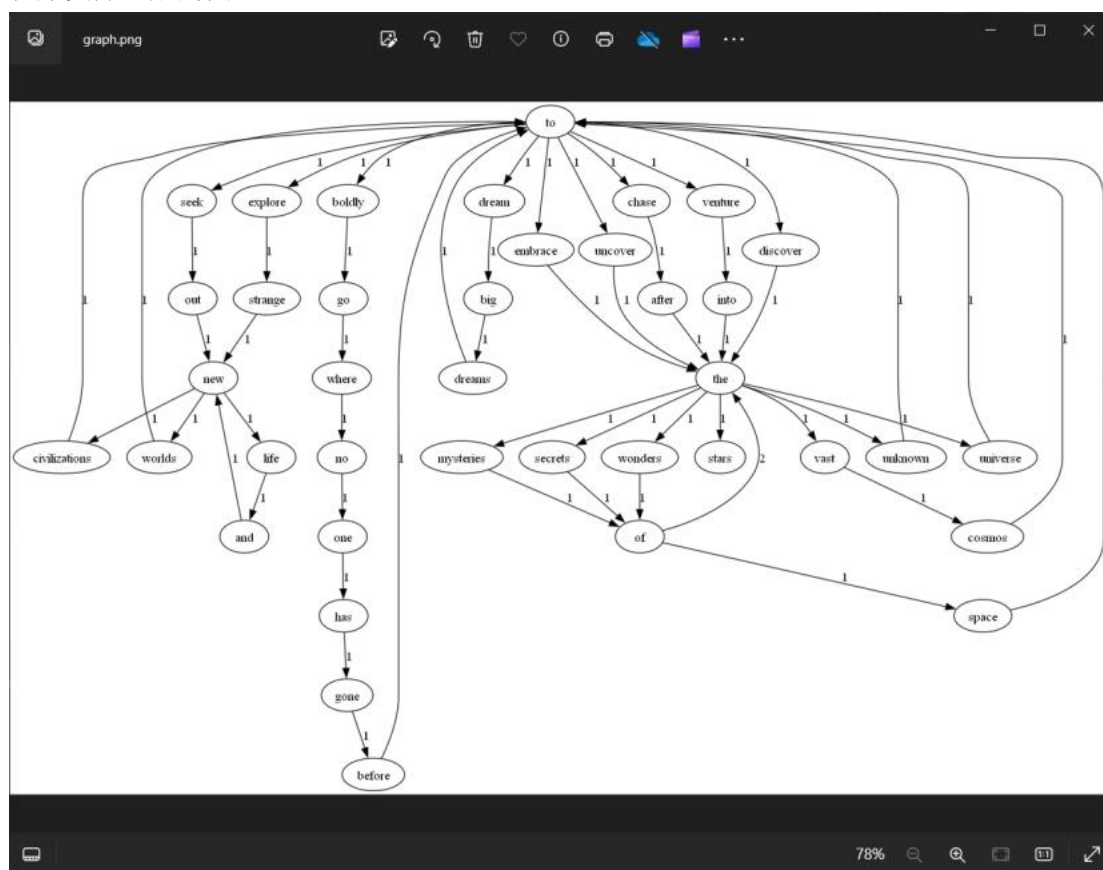
To dream big dreams, # to chase after the unknown, & to discover the mysteries of the universe.

To venture into the vast cosmos, to embrace ^ the wonders of space, to uncover the secrets of the stars.

期望生成的图（手工计算得到）：

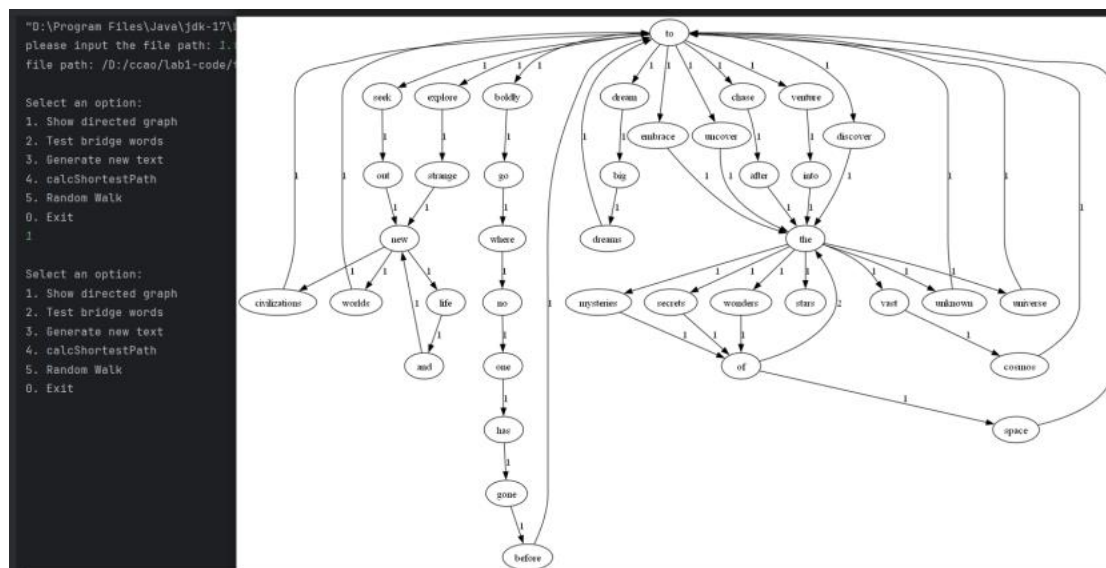


程序实际生成的图：



二者是否一致：一致

给出实际运行得到结果的界面截图。



4.2 查询桥接词

三个测试分别覆盖手册三个功能：

- 1) 输入的 word1 或 word2 如果不在图中出现，则输出 “No word1 or word2 in the graph!”
- 2) 如果不存在桥接词，则输出 “No bridge words from word1 to word2!”
- 3) 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.”

序号	输入（2个单词）	期望输出	实际输出	运行是否正确
1	to,the	The bridge words from "to" to "the" is: discover embrace uncover	The bridge words from "to" to "the" is: discover embrace uncover	是
2	to,dream	No bridge words from "to" to "dream"!	No bridge words from "to" to "dream"!	是
3	a,b	No "a" and "b" in the graph!	No "a" and "b" in the graph!	是

给出实际运行得到结果的界面截图。

```
Enter word 1: to
Enter word 2: the
The bridge words from "to" to "the" is: discover embrace uncover
```

```
Enter word 1: to
Enter word 2: dream
No bridge words from "to" to "dream"!
```

```
Enter word 1: a
Enter word 2: b
No "a" and "b" in the graph!
```

4.3 根据桥接词生成新文本

三个测试覆盖手册的两个功能:

- 1) 如果两个单词无 bridge word, 则保持不变, 不插入任何单词;
- 2) 如果两个单词之间存在多个 bridge words, 则随机从中选择一个插入进去形成新文本。

序号	输入 (一行文本)	期望输出	实际输出	运行是否正确
1	to big dreams	to dream big dreams	to dream big dreams	是
2	to the cosmos	to embrace the vast cosmos	to embrace the vast cosmos	是
3	a b c d	a b c d	a b c d	是

给出实际运行得到结果的界面截图。

```
Enter the new text: to big dreams
Generated new text: to dream big dreams
```

```
Enter the new text: to the cosmos
Generated new text: to embrace the vast cosmos
```

```
Enter the new text: a b c d
Generated new text: a b c d
```

4.4 计算最短路径

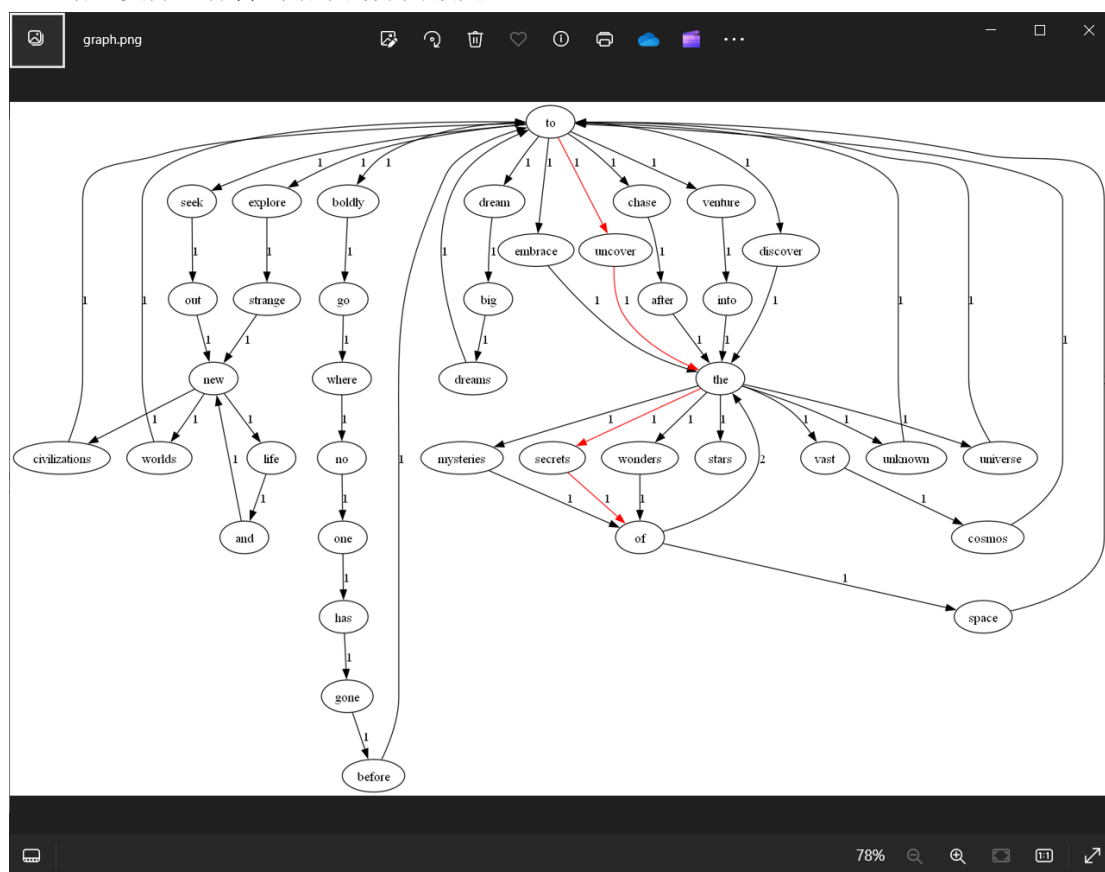
三个测试覆盖手册的两个功能:

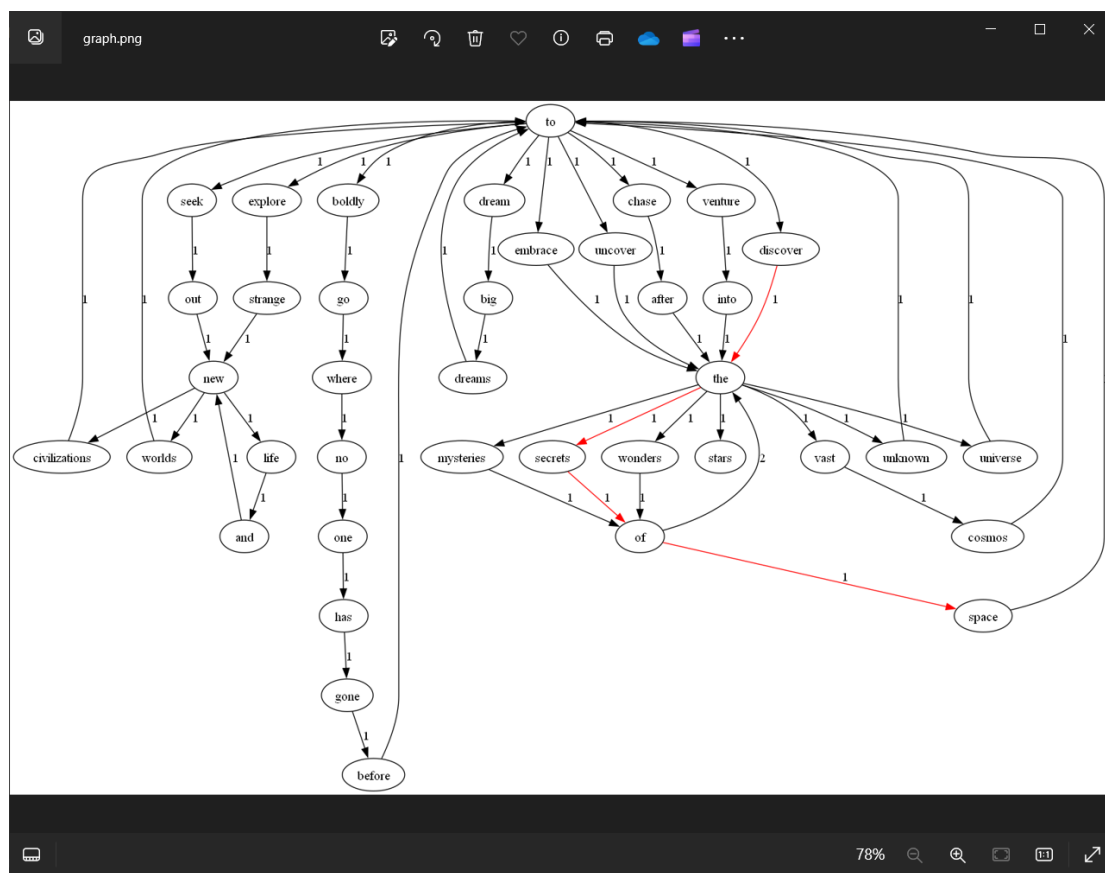
- 1) 用户输入两个单词, 程序计算它们之间在图中的最短路径(路径上所有边权值之和最小), 以某种突出的方式将路径标注在原图并展示在屏幕上同时展示路径的长度(所有边权值之和)。
- 2) 如果有多条最短路径, 只需要展示一条即可。
- 3) 如果输入的两个单词“不可达”, 则提示。
- 4) 如果用户只输入一个单词, 则程序计算出该单词到图中其他任一单词的最短路径, 并逐项展示出来。(该功能由于输出过长不在下方表格处书写, 输出见图示)

序号	输入 (两个单词、或一个单词)	期望输出	实际输出	运行是否正确
1	to, of	Shortest path: Shortest path from "to" to "of": to → 1 uncover → 1 the →	Shortest path: Shortest path from "to" to "of": to →	是

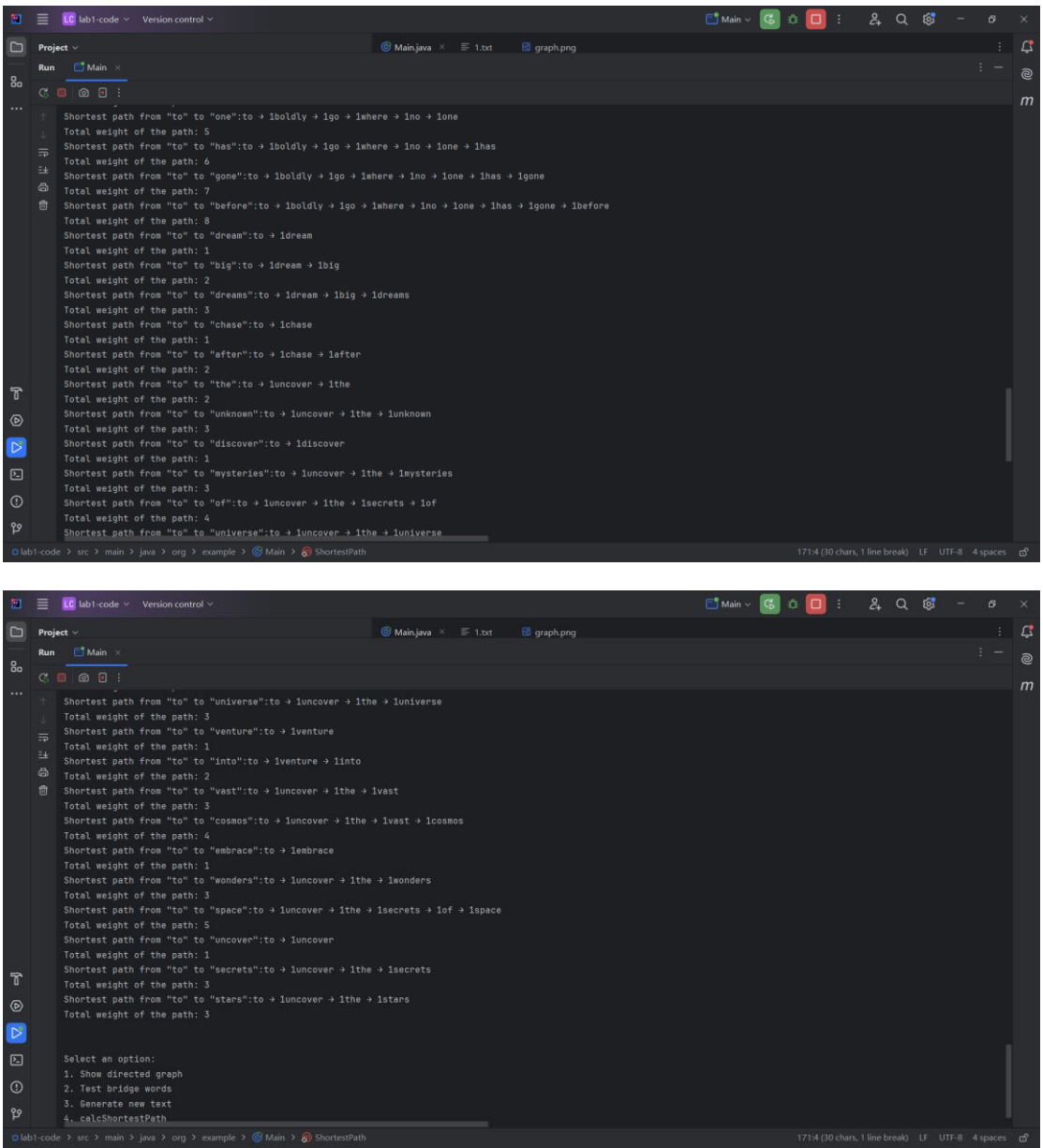
		1secrets → 1of Total weight of the path: 4	1uncover → 1the → 1secrets → 1of Total weight of the path: 4	
2	Discover, space	Shortest path: Shortest path from "discover" to "space":discover → 1the → 1secrets → 1of → 1space Total weight of the path: 4	Shortest path: Shortest path from "discover" to "space":discover → 1the → 1secrets → 1of → 1space Total weight of the path: 4	是
3	to		见图示	是

给出实际运行得到结果的界面截图。





```
lab1-code  Version control
Project  Main
Run  Main
Enter the second word(default):
Shortest path: There is no path from "to" to "to".
Shortest path from "to" to "explore":to -> lexplore
Total weight of the path: 1
Shortest path from "to" to "strange":to -> lexplore -> lstrange
Total weight of the path: 2
Shortest path from "to" to "new":to -> lexplore -> lstrange -> lnew
Total weight of the path: 3
Shortest path from "to" to "worlds":to -> lexplore -> lstrange -> lnew -> lworlds
Total weight of the path: 4
Shortest path from "to" to "seek":to -> lseek
Total weight of the path: 1
Shortest path from "to" to "out":to -> lseek -> lout
Total weight of the path: 2
Shortest path from "to" to "life":to -> lexplore -> lstrange -> lnew -> llife
Total weight of the path: 4
Shortest path from "to" to "and":to -> lexplore -> lstrange -> lnew -> llife -> land
Total weight of the path: 5
Shortest path from "to" to "civilizations":to -> lexplore -> lstrange -> lnew -> lcivilizations
Total weight of the path: 4
Shortest path from "to" to "boldly":to -> lboldly
Total weight of the path: 1
Shortest path from "to" to "go":to -> lboldly -> lgo
Total weight of the path: 2
Shortest path from "to" to "where":to -> lboldly -> lgo -> lwhere
Total weight of the path: 3
Shortest path from "to" to "no":to -> lboldly -> lgo -> lwhere -> lno
Total weight of the path: 4
Shortest path from "to" to "one":to -> lboldly -> lgo -> lwhere -> lno -> lone
lab1-code > src > main > java > org > example > Main > ShortestPath
```

4.5 随机游走

三个测试覆盖手册的两个功能：

- 1) 进入该功能时，程序随机的从图中选择一个节点，以此为起点沿出边进行随机遍历，记录经过的所有节点和边，直到出现第一条重复的边为止，或者进入的某个节点不存在出边为止。在遍历过程中，用户也可随时停止遍历。
- 2) 将遍历的节点输出为文本，并以文件形式写入磁盘。

序号	实际输出	程序运行是否正确
1	life and new life and	是
2	gone before to explore strange new civilizations to boldly go where no one has gone before	是
3	where no one has gone	是

给出实际运行得到结果的界面截图。

```
Performing random walk...
Press Enter to pause...
Write to file successfully.
Content: life and new life and
Random walking: life and new life and
```

```
Content: gone before to explore strange new civilizations to boldly go where no one has gone before
Random walking: gone before to explore strange new civilizations to boldly go where no one has gone before
```

```
Write to file successfully.
Content: where no one has gone
Random walking: where no one has gone
```

5 编程语言与开发环境

JDK17; Idea2022.3.2。

6 结对编程

6.1 分组依据

我们两人组成小组是因为我们在性格，能力，和编程技能方面都可以互补。

具体依据如下：

1) 性格互补：

高浚豪具有更为冷静、沉稳的性格，善于分析问题和制定计划。更倾向于深入思考和理性分析，能够保持冷静并处理复杂情况。

徐柯炎更加充满活力和创造力，具有更强的执行力和实践能力。更倾向于积极行动和寻找创新解决方案。

2) 能力互补：

高浚豪在逻辑思维和问题解决能力方面较为突出，擅长分析和解决复杂的编程难题，善于理清思路和规划项目进程。

徐柯炎在创意和想法方面更有优势，能够提供新颖的思路和创造性的解决方案，同时具有较强的沟通和表达能力。

3) 编程技能互补：

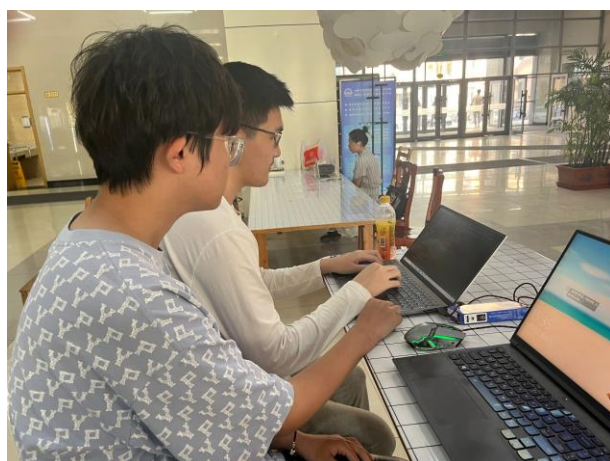
高浚豪在界面设计和用户体验等方面有所专长，能够为项目增添美观的外观和良好的用户交互体验。

徐柯炎更擅长算法和数据结构等底层编程技能，能够设计高效的程序和实现复杂的算法逻辑。

6.2 角色切换与任务分工

日期	时间(HH:MM -- HH:MM)	“驾驶员”	“领航员”	本段时间的任务
24/5/18	14:00-18:00	徐柯炎	高浚豪	深入分析实验要求,设计代码框架,并完成生成有向图,展示有向图,测试桥接词功能
24/5/18	19:00-22:40	高浚豪	徐柯炎	在下午编写的基础上,继续完成生成新文本,计算最短路径,随机游走功能
24/5/19	18:30-22:30	徐柯炎	高浚豪	对代码进行充分测试,发现其具有最短路径未考虑权值,随机游走过程无法随时停止等缺陷,进行修改并通过最终测试

6.3 工作照片



6.4 工作日志

日期/时间	问题描述	最终解决方法	两人如何通过交流找到解决方法
24/5/18 15:30	设计数据结构时出现问题	节点和单词相互进行哈希映射	充分交流经验并会想起数据结构相关知识
24/5/18 17:00	不确定采用什么方法生成有向图	使用 graphviz	共同查阅资料，交流后选择该工具
24/5/18 21:00	设计最短路径时发现数据结构存在缺陷	在原有节点定义的基础上增加关于距离的定义	共同查阅资料得出应该在何处修改

7 Git 操作过程

7.1 实验场景(1): 仓库创建与提交

首先初始化本地仓库: `git init`

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code
$ git init
Initialized empty Git repository in E:/git/lab1-code/.git/
```

然后将所有源文件加入缓存: `git add .`

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git add .
warning: in the working copy of '.idea/compiler.xml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '.idea/inspectionProfiles/Project_Default.xml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '.idea/misc.xml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'pom.xml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/main/java/org/example/Main.java', LF will be replaced by CRLF the next time Git touches it
```

用 `git status` 查看当前状态，发现所有文件都加入了缓存，但还尚未提交。

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .idea/.gitignore
    new file:   .idea/compiler.xml
    new file:   .idea/encodings.xml
    new file:   .idea/inspectionProfiles/Project_Default.xml
    new file:   .idea/jarRepositories.xml
    new file:   .idea/misc.xml
```

接着提交所有文件: `git commit -m "first commit"`

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git commit -m "first commit"
[master (root-commit) b2ac80c] first commit
32 files changed, 1063 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/compiler.xml
create mode 100644 .idea/encodings.xml
create mode 100644 .idea/inspectionProfiles/Project_Default.xml
create mode 100644 .idea/jarRepositories.xml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/uiDesigner.xml
create mode 100644 graph.png
```

再次查看当前状态，发现所有文件已被提交。

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git status
On branch master
nothing to commit, working tree clean
```

查看提交日志，发现当前提交已被记录。

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git log
commit b2ac80c78172b4b74b59564d02ecca40cba680c2 (HEAD -> master)
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 08:24:35 2024 +0800

    first commit
```

修改文件，查看上次提交之后修改过的文件：

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   random_walk.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

查看具体修改的内容：

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git diff
diff --git a/random_walk.txt b/random_walk.txt
index 8ece618..a69300f 100644
--- a/random_walk.txt
+++ b/random_walk.txt
@@ -1,2 @@
     life and new worlds to
+life and new worlds to
\ No newline at end of file
```

可以看到，random_walk.txt 后面新增了一行。

重新提交：

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git commit -am "second commit"
[master c11649a] second commit
1 file changed, 1 insertion(+)
```

再次修改：

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   random_walk.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

再次提交：

```
86151@LAPTOP-QNQOQ8M8 MINGW64 /e/git/lab1-code (master)
$ git commit -am "third commit"
[master 0dc6ece] third commit
1 file changed, 1 deletion(-)
```

撤销最后一次提交：

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git reset HEAD~1
Unstaged changes after reset:
M   random_walk.txt
```

此时查看文件修改，发现确实撤回了最后一次提交。

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   random_walk.txt

no changes added to commit (use "git add" and/or "git commit -a")

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git diff
diff --git a/random_walk.txt b/random_walk.txt
index a69300f..1af4bce 100644
--- a/random_walk.txt
+++ b/random_walk.txt
@@ -1,2 +1 @@
-life and new worlds to
+life and new worlds to
\ No newline at end of file
```

查询该项目的全部提交记录，发现确实只有两次提交。


```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git log
commit c11649afce0b60cfe21aae1bd36b70db87c87c33 (HEAD -> master)
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 08:36:49 2024 +0800

    second commit

commit b2ac80c78172b4b74b59564d02ecca40cba680c2
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 08:24:35 2024 +0800

    first commit
```

在 GitHub 上创建名为“Lab1-2021110683”的仓库，并为本地仓库配置远程仓库。



The screenshot shows the GitHub interface for a repository named 'Lab1-2021110683'. The repository is marked as 'Private'. At the top, there are buttons for 'Unwatch' and '1'. Below this, there's a navigation bar with 'master' selected, '1 Branch', and '0 Tags'. A search bar 'Go to file' is present, along with 'Add file' and 'Code' buttons. Below the navigation bar, there's a terminal window showing the following commands and output:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git remote add origin https://github.com/starryneigh/Lab1-2021110683.git

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git remote
origin
```

将之前各步骤得到的本地仓库全部内容推送到 GitHub 的仓库中。

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git push -u origin master
Enumerating objects: 57, done.
Counting objects: 100% (57/57), done.
Delta compression using up to 16 threads
Compressing objects: 100% (47/47), done.
Writing objects: 100% (57/57), 230.75 KiB | 3.78 MiB/s, done.
Total 57 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/starryneigh/Lab1-2021110683.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

查看 github 界面，提交成功。

Lab1-2021110683 Private Unwatch 1

master 1 Branch 0 Tags Go to file Add file Code

starryneigh second commit c11649a · 19 minutes ago 2 Commits

.idea	first commit	32 minutes ago
src	first commit	32 minutes ago
target	first commit	32 minutes ago
graph.png	first commit	32 minutes ago
pom.xml	first commit	32 minutes ago
random_walk.txt	second commit	19 minutes ago

7.2 实验场景(2): 分支管理

获得本地 Lab1 仓库的全部分支，切换至分支 master。

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git branch
* master
```

当前已经在 master 分支，无需切换。

在 master 基础上建立两个分支 B1、B2；

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git branch B1
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git branch B2
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git branch
B1
B2
* master
```

在 B2 分支基础上创建一个新分支 C4；

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (master)
$ git checkout B2
Switched to branch 'B2'
M random_walk.txt
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B2)
$ git branch c4
```

在 C4 上，对 2 个文件进行修改并提交；

切换到 C4：

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B2)
$ git checkout c4
Switched to branch 'c4'
M random_walk.txt
```

修改两个文件：

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (c4)
$ git status
On branch c4
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   1.txt
        modified:   random_walk.txt
no changes added to commit (use "git add" and/or "git commit -a")
```


提交这两个文件:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (C4)
$ git commit -am "C4 first commit"
[c4 4156ba9] C4 first commit
2 files changed, 1 insertion(+), 2 deletions(-)
```

在 B1 分支上对同样的 2 个文件做不同修改并提交;

切换到 B1:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (C4)
$ git checkout B1
Switched to branch 'B1'
```

修改并提交:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git status
On branch B1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   1.txt
        modified:   random_walk.txt

no changes added to commit (use "git add" and/or "git commit -a")

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git commit -am "B1 first commit"
[B1 78cc78e] B1 first commit
2 files changed, 2 insertions(+), 1 deletion(-)
```

将 C4 合并到 B1 分支, 若有冲突, 手工消解;

合并分支, 发现有冲突。

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git merge C4
Auto-merging 1.txt
CONFLICT (add/add): Merge conflict in 1.txt
Auto-merging random_walk.txt
CONFLICT (content): Merge conflict in random_walk.txt
Automatic merge failed; fix conflicts and then commit the result.
```

手动消解:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1|MERGING)
$ git status
On branch B1
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both added:   1.txt
        both modified: random_walk.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

再次提交:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1|MERGING)
$ git commit -am "merge B1 C4"
[B1 60594bd] merge B1 C4
```

在 B2 分支上对 2 个文件做修改并提交;

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git checkout B2
Switched to branch 'B2'

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B2)
$ git commit -am "B2 first commit"
[B2 9c81b50] B2 first commit
1 file changed, 1 deletion(-)
```

查看目前哪些分支已经合并、哪些分支尚未合并;

在 B1 分支上查看当前已经合并的节点和尚未合并的节点:


```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git branch --merged
* B1
  C4
  master

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git branch --no-merged
B2
```

将已经合并的分支删除,将尚未合并的分支合并到一个新分支上,分支名字为你的学号;
删除 C4:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git branch -d C4
Deleted branch C4 (was 4156ba9).
```

将尚未合并的分支合并到一个新分支上:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (B1)
$ git checkout 2021110683
Switched to branch '2021110683'

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683)
$ git branch --no-merged
B1
B2

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683)
$ git merge B1
Auto-merging 1.txt
CONFLICT (content): Merge conflict in 1.txt
Automatic merge failed; fix conflicts and then commit the result.

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683|MERGING)
$ git commit -am "a"
[2021110683 9f1f33a] a

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683)
$ git merge B2
Auto-merging 1.txt
CONFLICT (add/add): Merge conflict in 1.txt
Automatic merge failed; fix conflicts and then commit the result.

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683|MERGING)
$ git commit -am "a"
[2021110683 3f93e02] a
```

查看未合并的分支:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683)
$ git branch --no-merged

86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683)
$
```

发现已经全部合并。

将本地以你的学号命名的分支推送到 GitHub 上自己的仓库内:

```
86151@LAPTOP-QNQQQ8M8 MINGW64 /e/git/lab1-code (2021110683)
$ git push -u origin 2021110683
Enumerating objects: 26, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 16 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (24/24), 1.87 KiB | 239.00 KiB/s, done.
Total 24 (delta 11), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (11/11), completed with 1 local object.
remote:
remote: Create a pull request for '2021110683' on GitHub by visiting:
remote:   https://github.com/starryneigh/Lab1-2021110683/pull/new/2021110683
remote:
To https://github.com/starryneigh/Lab1-2021110683.git
 * [new branch]      2021110683 -> 2021110683
branch '2021110683' set up to track 'origin/2021110683'.
```

Lab1-2021110683

Private

Unwatch1

2021110683

2 Branches

0 Tags

Go to file

Add file

Code

This branch is 10 commits ahead of master.

Contribute

查看完整的版本变迁树；

```
commit 3f93e02edeed59c2d0c8833d34fe475bb6687e0f (HEAD -> 2021110683, origin/2021110683)
Merge: 9f1f33a 628d0fa
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 09:41:07 2024 +0800

    a

* commit 628d0fa97e19612fe4a8ab45b77d9a703dd2a299 (B2)
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 09:25:11 2024 +0800

    a

* commit 9c81b50d0ee17f6dd11b4785adbd49d5fb5c17c2
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 09:18:30 2024 +0800

    B2 first commit

* commit 9f1f33a2485a5f4b0a9b8c5e8e724bed9e36b362
Merge: 60594bd 8fad9bf
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 09:40:43 2024 +0800

    a

* commit 8fad9bf8333266ee3b55a190027e35f180d018e6 (B1)
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 09:37:07 2024 +0800

    a

* commit 60594bd17536eb6bce9ea51491c33f4da0be3973
Merge: 78cc78e 4156ba9
Author: starryneigh <2259245769@qq.com>
Date: Thu May 23 09:17:00 2024 +0800
```

在 Github 上以 web 页面的方式查看你的 Lab1 仓库的当前状态。
Master 分支：

Lab1-2021110683

Private

Unwatch1

2021110683

2 Branches

0 Tags

Go to file

Add file

Code

starryneigh second commit

c11649a · 1 hour ago

2 Commits

.idea	first commit	1 hour ago
src	first commit	1 hour ago
target	first commit	1 hour ago
graph.png	first commit	1 hour ago
pom.xml	first commit	1 hour ago
random_walk.txt	second commit	1 hour ago

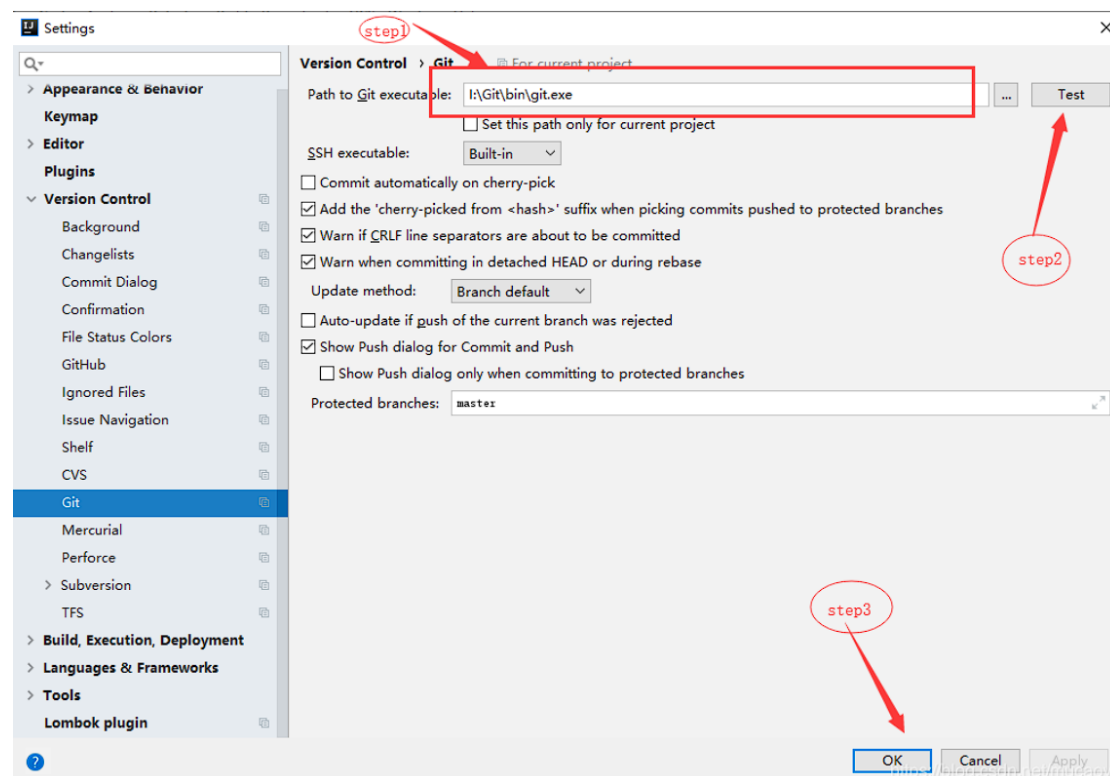
2021110683 分支:

The screenshot shows a GitHub repository page for 'Lab1-2021110683'. At the top, it indicates '2021110683' had recent pushes 11 minutes ago, with a 'Compare & pull request' button. Below this, it shows '2021110683' with '2 Branches' and '0 Tags'. A search bar and 'Add file' button are present. A status bar indicates 'This branch is 10 commits ahead of master'. The file list shows:

File	Commit	Time
.idea	first commit	1 hour ago
src	first commit	1 hour ago
target	first commit	1 hour ago
1.txt	a	18 minutes ago
graph.png	first commit	1 hour ago
pom.xml	first commit	1 hour ago
random_walk.txt	C4 first commit	49 minutes ago

8 在 IDE 中使用 Git Plugin

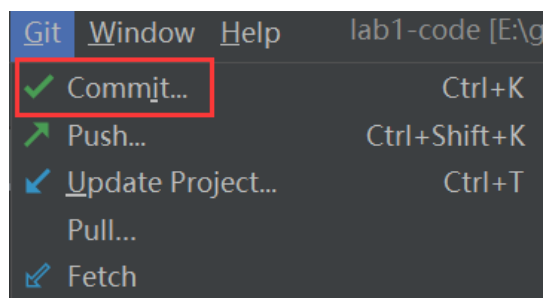
首先将 idea 连接 git:



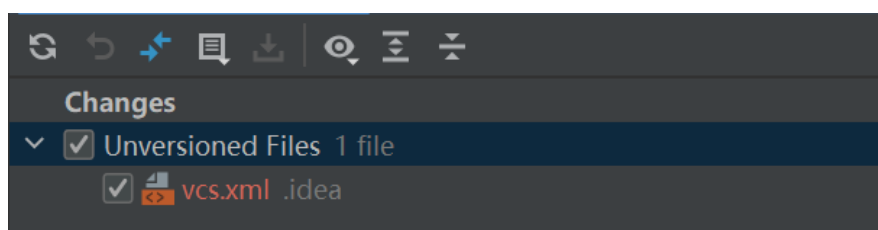
然后在 VCS 中开启版本控制，选择 git，即可在 idea 中使用 git。

接着向本地仓库提交:

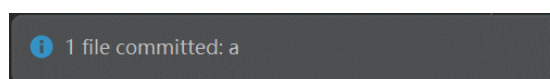
点击以下按钮:



点击需要 commit 的文件:

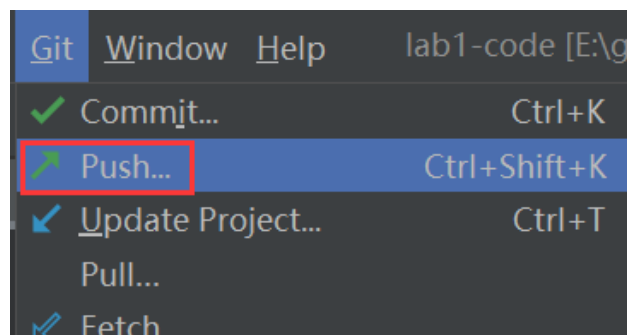


提交成功。

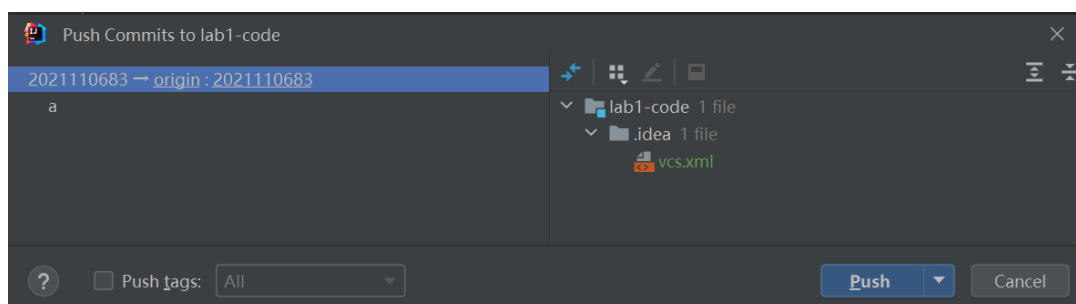


向 github 提交:

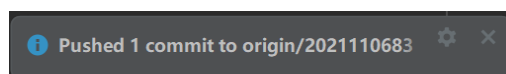
点击以下按钮:



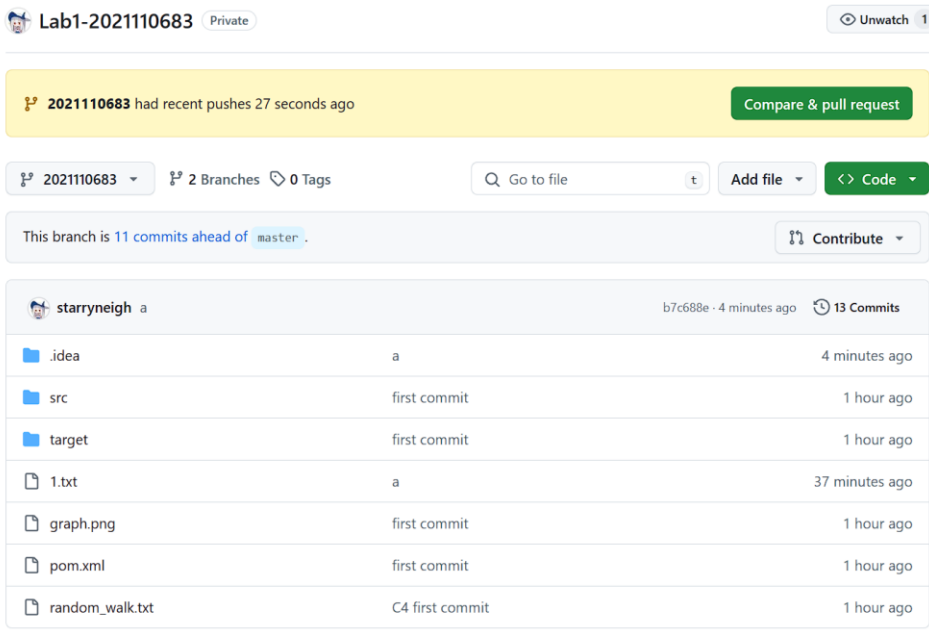
点击 push:



push 成功:



查看远程仓库确认 push。



9 小结

通过本次实验，我获得了以下收获：

- 1) 学习了 java 语言的基本语法，了解了 java 项目的结构；
- 2) 了解了结对编程的基本步骤，提高了开发的速度和质量；
- 3) 熟练掌握 Git 的基本指令和分支管理指令；
- 4) 掌握 Git 支持软件配置管理的核心机理；
- 5) 在实践项目中使用 Git /Github 管理自己的项目源代码。