

哈尔滨工业大学 计算学部

2024 年秋季学期 《开源软件开发实践》

Lab 2: 开源软件开发协作流程

姓名	学号	联系方式
何孟钊	2022111000	18107815322

目 录

1	实验要求.....	1
2	实验内容 1 发送 pull request.....	1
2.1	fork 项目	1
2.2	git 操作命令	2
2.3	代码修改.....	3
2.4	测试类代码.....	4
2.5	测试通过截图.....	9
3	实验内容 2 接受 pull request.....	10
4	实验内容 3 github 辅助工具	10
4.1	熟悉 GoodFirstIssue 工具	10
4.2	安装并使用 Hypercrx.....	11
4.3	利用 OpenLeaderboard 工具	13
5	小结.....	15

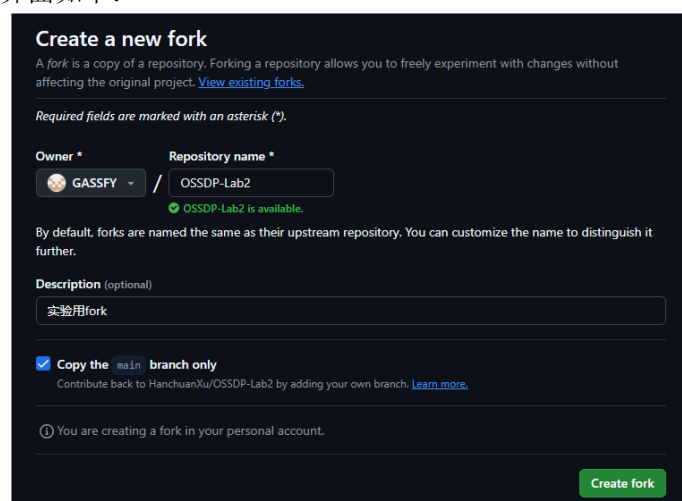
1 实验要求

简要复述实验手册中要求达到的实验目标与过程。

2 实验内容 1 发送 pull request

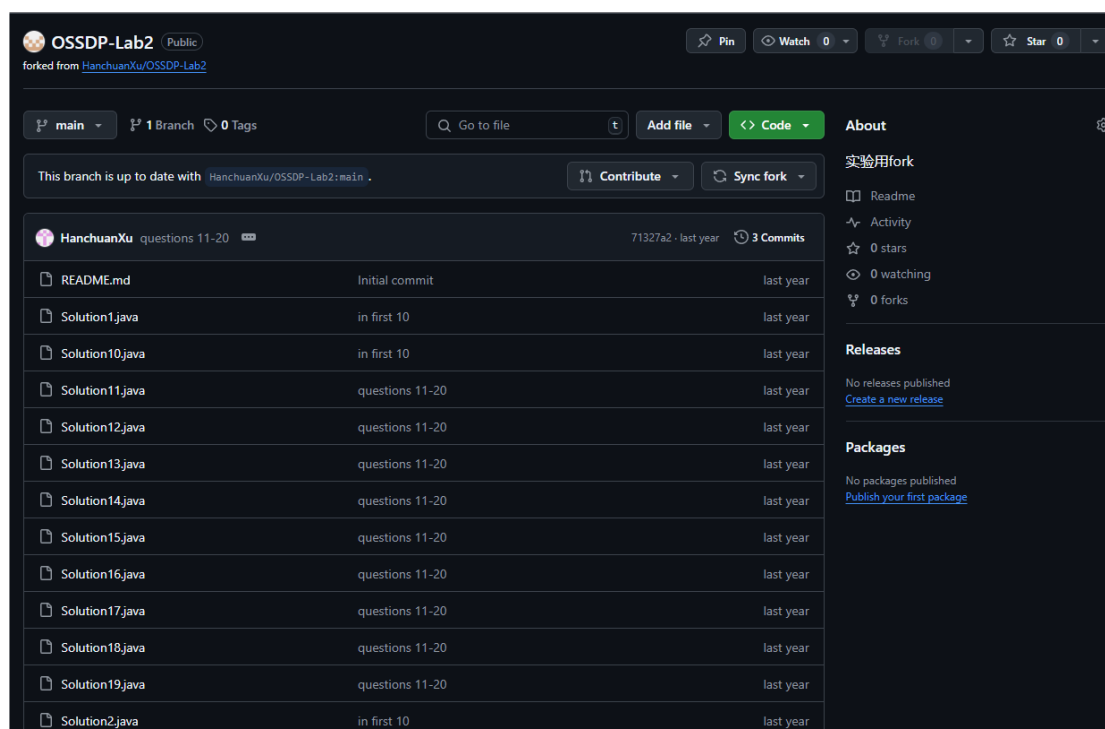
2.1 fork 项目

请求 fork 的界面如下：



The image shows the GitHub 'Create a new fork' form. It includes fields for 'Owner' (GASSFY) and 'Repository name' (OSSDP-Lab2), a 'Description' field with the text '实验用fork', and a checkbox for 'Copy the main branch only'. A green 'Create fork' button is at the bottom right.

Fork 成功的界面如下：



The image shows the GitHub repository page for OSSDP-Lab2, which is a fork of HanchuanXu/OSSDP-Lab2. The page displays the repository name, a list of files (README.md, Solution1.java, Solution10.java, Solution11.java, Solution12.java, Solution13.java, Solution14.java, Solution15.java, Solution16.java, Solution17.java, Solution18.java, Solution19.java, Solution2.java), and a list of commits. The repository is public and has 0 stars, 0 forks, and 0 releases.

2.2 git 操作命令

首先使用 `git clone https://github.com/GASSFY/OSSDP-Lab2` 命令来将仓库克隆到本地，方便后续的修改。

之后按照实验手册的要求，新建并切换到 `fix` 分支上，开始做代码的修改：

```
Lenovo@LX MINGW64 ~/Desktop/Course/开源软件实践/实验二/OSSDP-Lab2 (main)
$ git checkout -b fix
Switched to a new branch 'fix'

Lenovo@LX MINGW64 ~/Desktop/Course/开源软件实践/实验二/OSSDP-Lab2 (fix)
$ |
```

在完成修改之后，我们查看现在的状态：

```
$ git status
On branch fix
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   L2022111000_Solution12_Test.java
    deleted:    Solution1.java
    deleted:    Solution10.java
    deleted:    Solution11.java
    deleted:    Solution13.java
    deleted:    Solution14.java
    deleted:    Solution15.java
    deleted:    Solution16.java
    deleted:    Solution17.java
    deleted:    Solution18.java
    deleted:    Solution19.java
    deleted:    Solution2.java
    deleted:    Solution20.java
    deleted:    Solution3.java
    deleted:    Solution4.java
    deleted:    Solution5.java
    deleted:    Solution6.java
    deleted:    Solution7.java
    deleted:    Solution8.java
    deleted:    Solution9.java

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:   L2022111000_Solution12_Test.java
    modified:   Solution12.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .idea/
    OSSDP-Lab2.iml
    out/
```

使用 `git add` 命令来把文件加入暂存区：

```
Lenovo@LX MINGW64 ~/Desktop/Course/开源软件实践/实验二/OSSDP-Lab2 (fix)
$ git add L2022111000_Solution12_Test.java

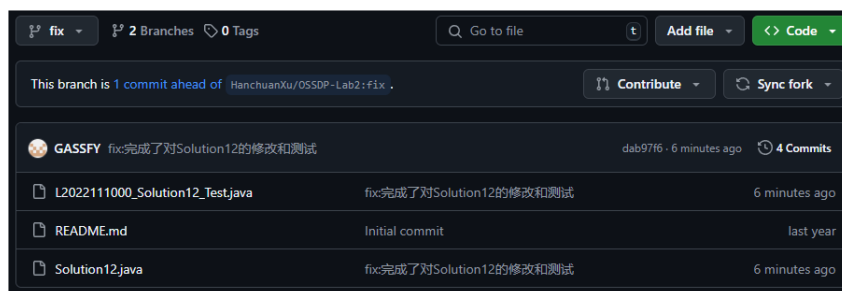
Lenovo@LX MINGW64 ~/Desktop/Course/开源软件实践/实验二/OSSDP-Lab2 (fix)
$ git add Solution12.java
```

输入 `git commit` 命令对修改进行提交：

```
Lenovo@LX MINGW64 ~/Desktop/Course/开源软件实践/实验二/
$ git commit -m"fix:完成了对Solution12的修改和测试"
[fix dab97f6] fix:完成了对Solution12的修改和测试
21 files changed, 147 insertions(+), 1314 deletions(-)
create mode 100644 L2022111000_Solution12_Test.java
delete mode 100644 Solution1.java
delete mode 100644 Solution10.java
delete mode 100644 Solution11.java
delete mode 100644 Solution13.java
delete mode 100644 Solution14.java
delete mode 100644 Solution15.java
delete mode 100644 Solution16.java
delete mode 100644 Solution17.java
delete mode 100644 Solution18.java
delete mode 100644 Solution19.java
delete mode 100644 Solution2.java
delete mode 100644 Solution20.java
delete mode 100644 Solution3.java
delete mode 100644 Solution4.java
delete mode 100644 Solution5.java
delete mode 100644 Solution6.java
delete mode 100644 Solution7.java
delete mode 100644 Solution8.java
delete mode 100644 Solution9.java
```

最后输入 `git push` 命令，将修改的文件推送上远程仓库：

```
$ git push origin fix
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.86 KiB | 1.86 MiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'fix' on GitHub by visiting:
remote:   https://github.com/GASSFY/OSSDP-Lab2/pull/new/fix
remote:
To https://github.com/GASSFY/OSSDP-Lab2
 * [new branch]      fix -> fix
```



2.3 代码修改

```
3      import java.util.*;
4      /*
5       * @Description:
6       * ## 字符串相乘
7       * 给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2
      的乘积，它们的乘积也表示为字符串形式。
8       * 注意：不能使用任何内置的 BigInteger 库或直接将输入转换为整数。
9       *
10      * 示例 1:
11      * 输入: num1 = "2", num2 = "3"
12      * 输出: "6"
13      *
14      * 示例 2:
15      * 输入: num1 = "123", num2 = "456"
16      * 输出: "56088"
17      *
18      */
19
20      class Solution12 {
21          public String multiply(String num1, String num2) {
22              if (num1.equals("0") || num2.equals("0")) {
23                  return "0";
24              }
25              String ans = "0";
26              int m = num1.length(), n = num2.length();
```

```
27         for (int i = n - 1; i >= 0; i--) {
28             StringBuffer curr = new StringBuffer();
29             int add = 0;
30             for (int j = n - 1; j > i; j--) {
31                 curr.append(0);
32             }
33             int y = num2.charAt(i) - '0';
34             for (int j = m - 1; j >= 0; j--) {
35                 int x = num1.charAt(j) - '0';
36                 int product = x * y + add;
37                 curr.append(product % 10);
38                 add = product / 10;
39             }
40             if (add != 0) {
41                 curr.append(add % 10);
42             }
43             ans = addStrings(ans, curr.reverse().toString());
44         }
45         return ans;
46     }
47
48     public String addStrings(String num1, String num2) {
49         int i = num1.length() - 1, j = num2.length() - 1, add = 0;
50         StringBuffer ans = new StringBuffer();
51         while (i >= 0 || j >= 0 || add != 0) {
52             int x = i >= 0 ? num1.charAt(i) - '0' : 0;
53             int y = j >= 0 ? num2.charAt(j) - '0' : 0;
54             int result = x + y + add;
55             ans.append(result % 10);
56             add = result / 10;
57             i--;
58             j--;
59         }
60         ans.reverse();
61         return ans.toString();
62     }
63 }
```

2.4 测试类代码

```
3     import org.junit.Test;
4     import static org.junit.Assert.assertEquals;
5     import static org.junit.Assert.assertFalse;
6
```

```

7      public class L2022111000_Solution12_Test {
8
9          /**
10             * 在测试multiply方法时，定义两个参数分别为A和B，可以进行如下等
              价类的划分：
11             * 基于乘数是否为0的划分：A为0B不为0，A不为0B为0，A和B都不
              为0，A和B都为0；
12             * 基于字符串长度的划分：A的长度大于B、B的长度大于A、A和B的长度
              相等；
13             * 基于A的B的值是否超过Long.MAX_VALUE的最大值的划分：A与B均
              不超过、A超过B不超过、B超过A不超过、A和B均超过
14             * 基于A与B的值相差的划分：A远大于B（10000000倍差距），A远小
              于B，A与B相差不大
15             * 基于乘积是否产生进位的划分：有进位、无进位
16             */
17
18          /**
19             * 测试目的：测试数值是否为0是否会影响计算的正确性
20             * 测试用例：A为“0”，B为“0”；
21             *      A为“7678324”，B为“21”；
22             *      A为“0”，B为“8490276”；
23             *      A为“2412”，B为“0”
24             */
25          @Test
26          public void multiply_1() {
27              Solution12 solution12 = new Solution12();
28              assertEquals("0",solution12.multiply("0","0"));
29
30              assertEquals("161244804",solution12.multiply("7678324","21"));
31              assertEquals("0",solution12.multiply("0","8490276"));
32              assertEquals("0",solution12.multiply("2412","0"));
33          }
34
35          /**
36             * 测试目的：测试两个参数长度的相差是否会影响计算结果的正确性
37             * 测试用例：A为“2398634”，B为“946”；
38             *      A为“6352”，B为“987474321”；
39             *      A为“44266”，B为“85024”。
40             */
41          @Test
42          public void multiply_2(){
43              Solution12 solution12 = new Solution12();
44
45              assertEquals("2269107764",solution12.multiply("2398634","946"));

```

```
44     assertEquals("6272436886992",solution12.multiply("6352","987474321")
45     );
46 }
47
48 /**
49  * 测试目的: 测试计算是否产生进位是否会对计算结果的正确性造成影响
50  * 测试用例: A 为“879656”, B 为“946”;
51  *          A 为“11111”, B 为“6423476”;
52  */
53 @Test
54 public void multiply_3(){
55     Solution12 solution12 = new Solution12();
56
57     assertEquals("832154576",solution12.multiply("879656","946"));
58
59     assertEquals("71371241836",solution12.multiply("11111","6423476"));
60 }
61
62 /**
63  * 测试目的: A 和 B 的值是否超过整型的最大值、A 与 B 的值是否相差巨大,
64  * 这两个因素计算的正确性是否造成影响
65  * 测试用例: A 为“112”, B 为“115”;
66  *          A 为“9223372036854775808”, B 为“827”;
67  *          A 为“53”, B 为“9223372036854775808”;
68  *          A 为“9223372036854775808”, B 为“9223372036854775809”
69  */
70 @Test
71 public void multiply_4(){
72     Solution12 solution12 = new Solution12();
73
74     assertEquals("7627728674478899593216",solution12.multiply("922337203
75     6854775808","827"));
76
77     assertEquals("488838717953303117824",solution12.multiply("53","92233
78     72036854775808"));
79
80     assertEquals("85070591730234615875067023894796828672",solution12.mul
81     tiply("9223372036854775808","9223372036854775809"));
82 }
83
84 }
```



```

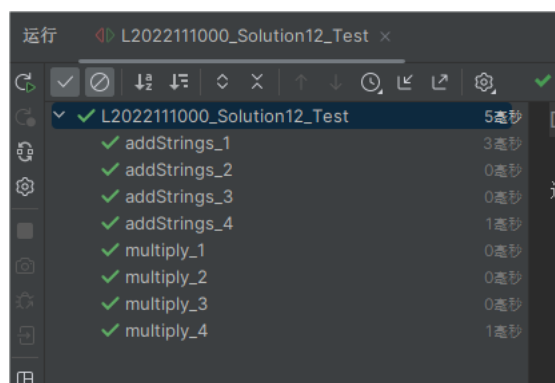
75  /*****
76  *****/
77  /**
78   * 在测试 addString 方法时，定义两个参数分别为A 和B，可以进行如下
    等价类的划分：
79   * 基于字符串长度的划分：A 的长度大于B、B 的长度大于A、A 和B 的长度
    相等；
80   * 基于数值是否为0 的划分：A 和B 都为0、A 和B 都不为0、A 为0B 不为
    0、B 为0A 不为0；
81   * 基于加和是否发生进位的划分：A 加B 不存在进位、A 加B 有一部分存在
    进位一部分不进位、A 和B 的每一位相加都发生进位
82   * 基于A 的B 的值是否超过 Long.MAX_VALUE 的最大值的划分：A 与B 均
    不超过、A 超过B 不超过、B 超过A 不超过、A 和B 均超过
83   * 基于A 与B 的值相差的划分：A 远大于B（10000000 倍差距），A 远小
    于B，A 与B 相差不大
84   */
85
86  /**
87   * 测试目的：测试字符串长度是否会对方法的正确性造成影响
88   * 测试用例：A 为“2398634”，B 为“946”；
89   *      A 为“6352”，B 为“987474321”；
90   *      A 为“44266”，B 为“85024”。
91   */
92  @Test
93  public void addStrings_1() {
94      Solution12 solution12 = new Solution12();
95      assertEquals("2399580", solution12.addStrings("2398634",
    "946"));
96      assertEquals("987480673", solution12.addStrings("6352",
    "987474321"));
97      assertEquals("129290", solution12.addStrings("44266",
    "85024"));
98  }
99
100 /**
101  * 测试目的：测试数值是否为0 是否会影响计算的正确性
102  * 测试用例：A 为“0”，B 为“0”；
103  *      A 为“98546”，B 为“221”；
104  *      A 为“0”，B 为“435720”；
105  *      A 为“44167”，B 为“0”
106  */
107 @Test

```

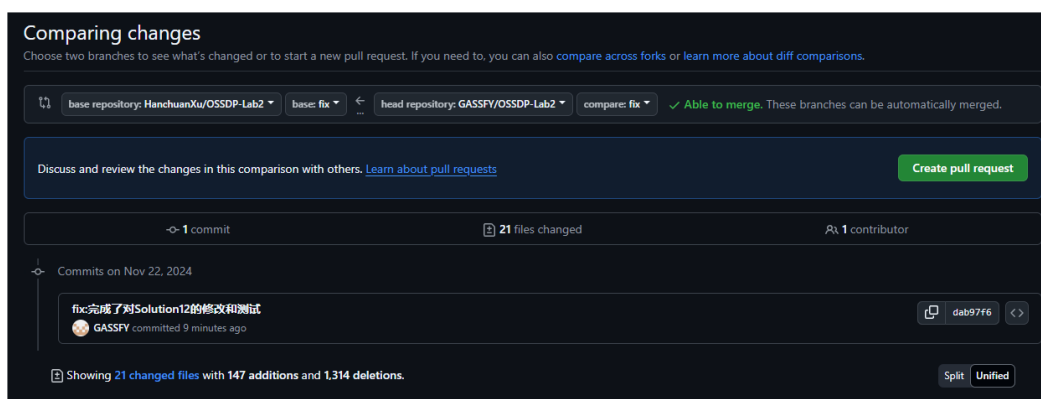
```
108     public void addStrings_2(){
109         Solution12 solution12 = new Solution12();
110         assertEquals("0",solution12.addStrings("0","0"));
111
112         assertEquals("98767",solution12.addStrings("98546","221"));
113
114         assertEquals("435720",solution12.addStrings("0","435720"));
115
116         assertEquals("44167",solution12.addStrings("44167","0"));
117     }
118
119     /**
120      * 测试目的: 加和是否产生进位是否对计算的正确性造成影响
121      * 测试用例: A 为“24113”, B 为“1341122”;
122      *           A 为“1199911”, B 为“2228888”;
123      *           A 为“989898988”, B 为“77777777”
124      */
125     @Test
126     public void addStrings_3(){
127         Solution12 solution12 = new Solution12();
128
129         assertEquals("1365235",solution12.addStrings("24113","1341122"));
130
131         assertEquals("3428799",solution12.addStrings("1199911","2228888"));
132
133         assertEquals("1067676765",solution12.addStrings("989898988","77777777"));
134     }
135
136     /**
137      * 测试目的: A 和 B 的值是否超过整型的最大值、A 与 B 的值是否相差巨大,
138      *           这两个因素计算的正确性是否造成影响
139      * 测试用例: A 为“112”, B 为“115”;
140      *           A 为“9223372036854775808”, B 为“827”;
141      *           A 为“53”, B 为“9223372036854775808”;
142      *           A 为“9223372036854775808”, B 为“9223372036854775809”
143      */
144     @Test
145     public void addStrings_4(){
146         Solution12 solution12 = new Solution12();
147
148         assertEquals("227",solution12.addStrings("112","115"));
149
150         assertEquals("9223372036854776635",solution12.addStrings("9223372036854775808","827"));
151     }
```

```
142 assertEquals("9223372036854775861",solution12.addStrings("53","92233
143 assertEquals("18446744073709551617",solution12.addStrings("922337203
144     }
145 }
```

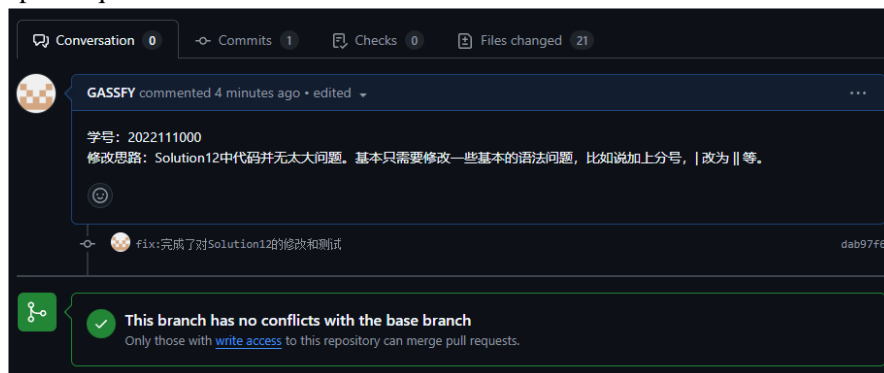
2.5 测试通过截图



在测试通过后，我们提交 PR：
之后创建 pull request:

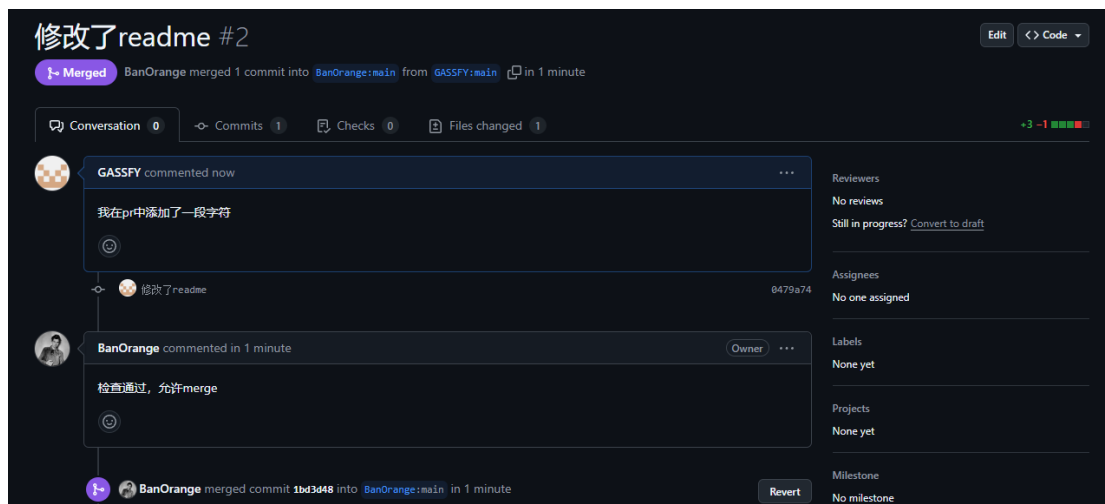


提交 pull request 如下:

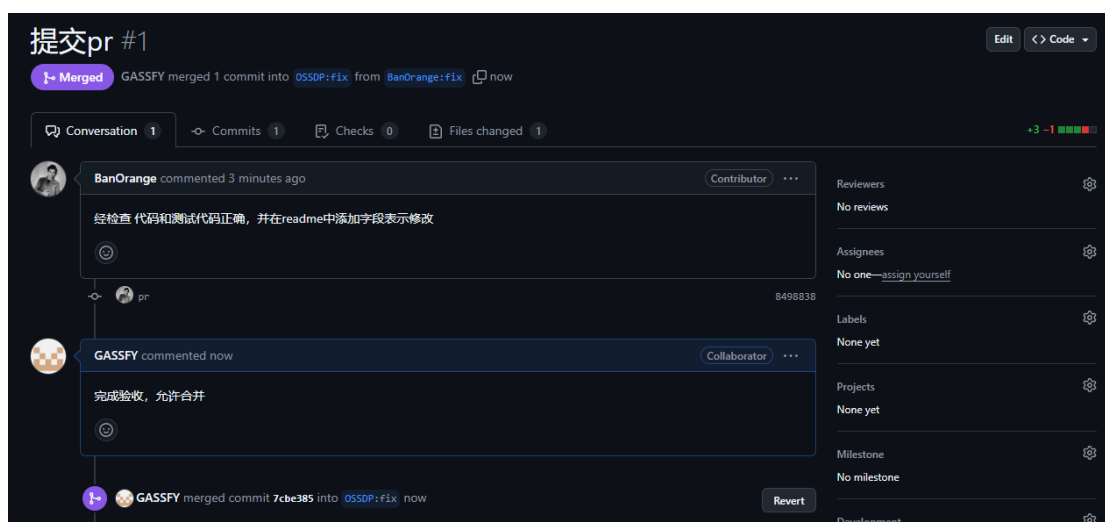


3 实验内容 2 接受 pull request

己方提交 PR，对方接受：



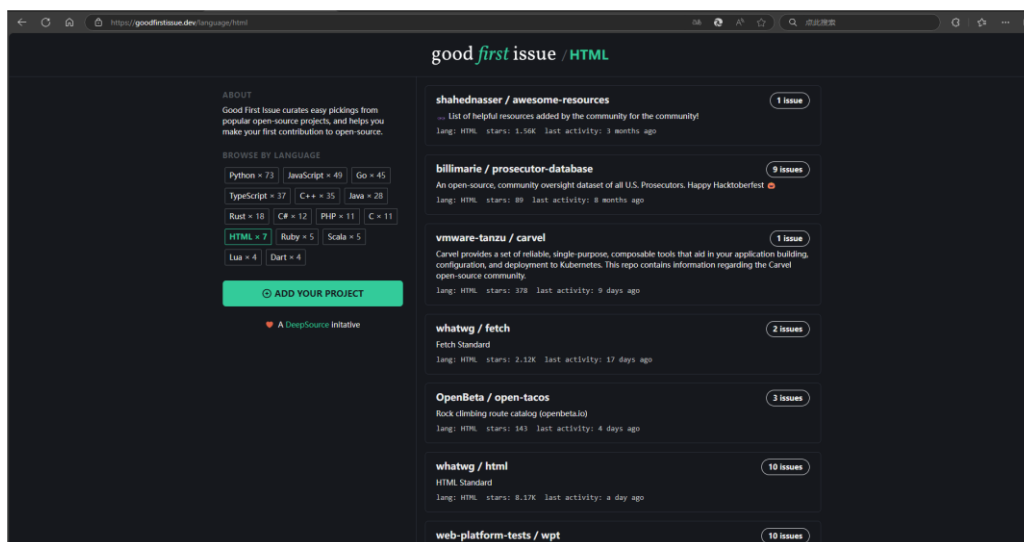
对方提交 PR，己方接受：



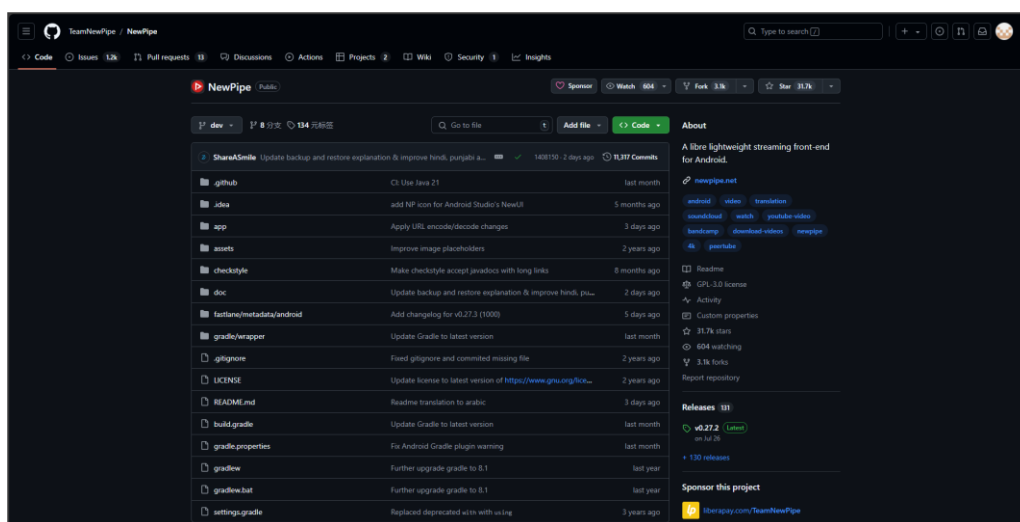
4 实验内容 3 github 辅助工具

4.1 熟悉 GoodFirstIssue 工具

首先我们打开 <https://goodfirstissue.dev/> 网站，可见显示如下：



网站的使用十分简单。用户可以通过选择标签筛选出感兴趣的项目，点击该项目即可尝试做出贡献：



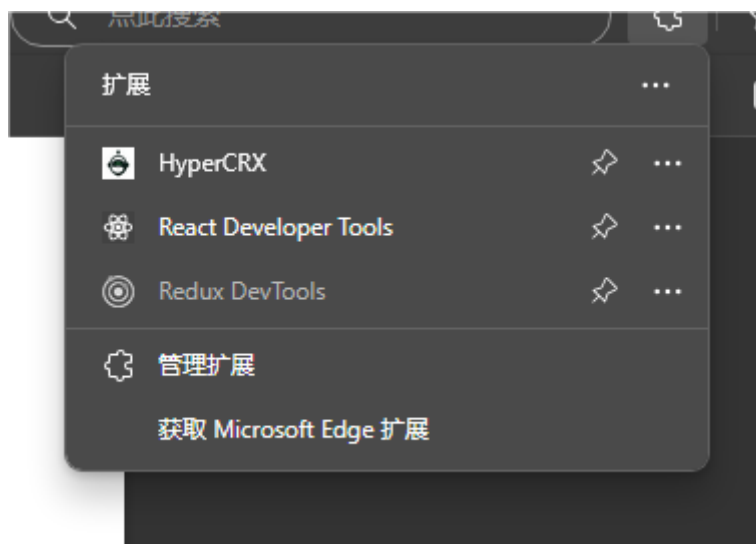
此外，你还可以添加你的开源项目到此网站中，但是需要符合以下要求：

1. 确保项目的活跃，至少应该由 10 个贡献者，并且能够快速回答提出的问题；
2. 在所有列出的 issue 中应该至少有三个标注为 good first issue；
3. 应该有详细的 README 文档和 CONTRIBUTING 指南，帮助新贡献者顺利进行设置和贡献流程；

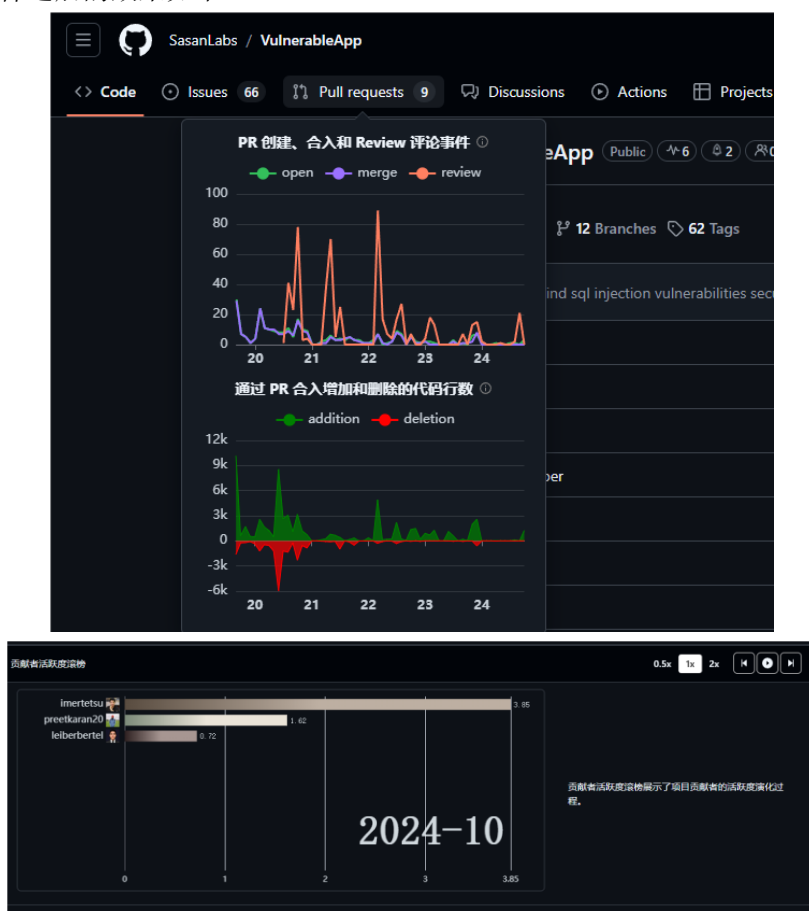
可以说 good first issue 这个平台是能够帮助刚刚接触开源的人快速上手的一个好平台。

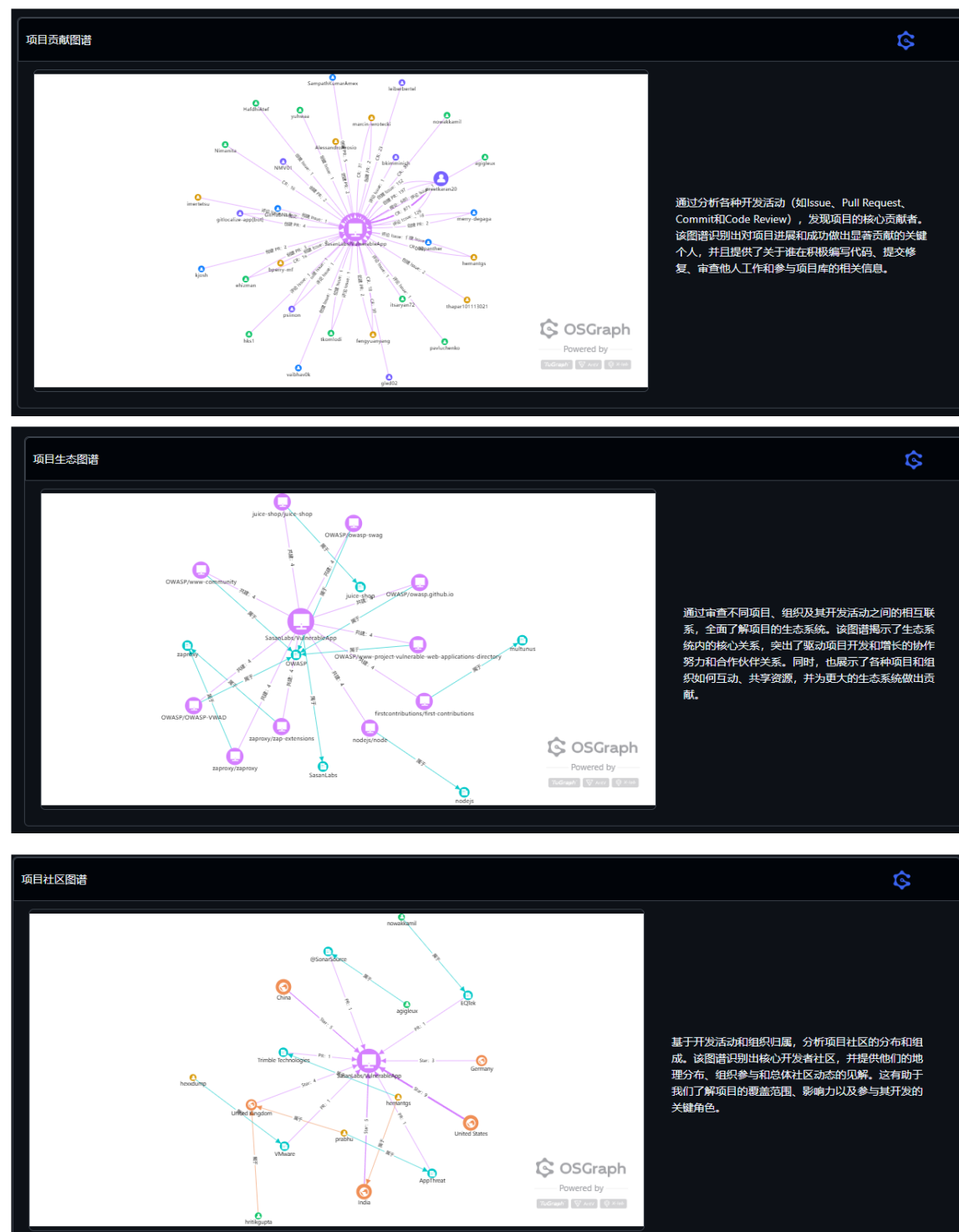
4.2 安装并使用 Hypercrx

访问实验指导书中提及的网址，即可下载对应的插件：



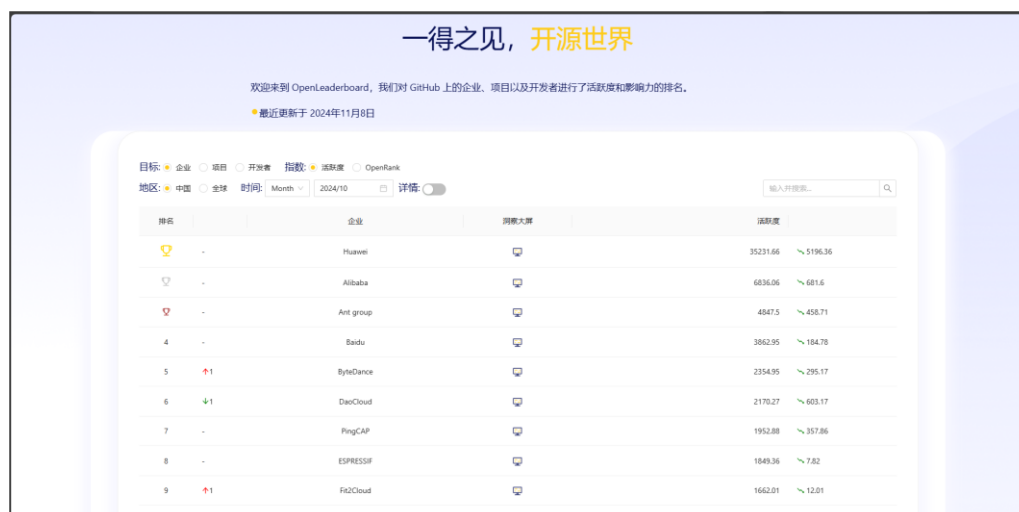
使用插件之后的效果如下：





4.3 利用 OpenLeaderboard 工具

访问实验指导书中的网址，界面如下：



该工具是一个源项目分析工具，它通过数据可视化的方式帮助用户直观地展示和分析开源项目的质量、进展、问题以及关注度和活跃度。其原理是基于对 GitHub 上所有仓库和开发者行为的统计和加权算法。它计算开发者活跃度（Ad），通过五种行为事件（如 Issue 评论、发起 Issue、发起 PR、PR 上的代码 review 评论、PR 合入）的发生次数（ci）和相应的加权比例（wi）来评估。项目的活跃度（Ar）则是所有开发者活跃度的开方和，这样做可以减少核心开发者过高活跃度对项目活跃度的影响。此外，OpenLeaderboard 还引入了 OpenRank 算法，这是一种基于全域开发者协作网络的项目影响力评估方法，能够评估项目的协作影响力和协作关联度。

该网站的使用方法也十分简单，用户只需要进入网站，选择对应的 tag 即可。

活跃度指标：

活跃度指标通过统计开发者在项目中的行为（如发起 Issue、PR，以及代码 review 等）来计算。每种行为都被赋予一个权重，权重反映了该行为对项目活跃度的贡献程度。例如，Issue 评论可能被赋予 1 分，发起 Issue 赋予 2 分，发起 PR 赋予 3 分，PR 上的代码 review 评论赋予 4 分，PR 合入赋予 5 分。

对于每个开发者，其活跃度是通过其触发的上述行为的发生次数乘以相应的权重来计算的。而对于项目而言，其活跃度是所有开发者活跃度的加权开的开方，开方操作是为了减少核心开发者过高活跃度对项目活跃度的影响。

活跃度指标的设计带有一定的价值导向，它鼓励开发者参与社区讨论、代码 review 等活动，而不仅仅是代码贡献。

然而，不可忽视的是，活跃度指标也存在一些问题。例如，权重的设定具有一定的主观性，活跃度缺乏基线使得不同时间段的比较变得困难，以及活跃度计算在仓库层面不具有线性可加性等。

协作影响力：

尽管上文所提及的直接基于活跃度的统计型指标对特定的项目具有较好的分析效果，但是现实中开源情况往往会更加复杂。为了解决这样的问题，开源协作网络的概念应运而生。

开源协作网络是一种评估开源项目影响力的方法，它通过分析开发者在不同项目间的活跃度来构建网络。项目间的关联度基于开发者在这些项目上的活跃情况，使用调和平均计算。利用 PageRank 算法，根据项目间的协作关系评估每个项目的影响力。这种方法能更准确地反映项目在开源生态中的重要性，规避了仅基于活跃度的评估问题。

价值流网络：

价值流网络通过量化开源生态中各个实体之间的价值流动来衡量项目的价值。该模型

考虑了两个主要维度：生产侧和消费侧。在前者的领域中，模型关注开发者对项目的贡献质量；而在消费侧，模型关注开源软件的实际使用情况

5 小结

通过完成这次开源软件开发实践的实验，我收获颇丰。这次实验不仅让我深入了解了基于代码托管平台的开源软件协作开发过程，还让我掌握了 GitHub 上软件项目协作开发的核心命令和方法。通过亲自操作，我更加熟悉了 GitHub 这个强大的工具，包括 fork 项目、创建分支、提交 PR（Pull Request）等关键操作，这些都是参与开源项目不可或缺的技能。

在实验内容 1 中，我通过解决一个编程题目并修复其中的 bug，加深了对 Java 编程和单元测试的理解。编写测试用例的过程锻炼了我的逻辑思维能力，也让我意识到了测试在软件开发中的重要性。此外，我还学习了如何有效地使用 git 命令来管理代码变更，这对于代码版本的控制和多人协作开发至关重要。

实验内容 2 模拟了接受他人 PR 的过程，这让我学会了如何评审代码和测试用例，以及如何通过 PR 与他人交流。这个过程中，我不仅提升了自己的代码质量，也学会了如何提供建设性的反馈，这对于提升团队协作效率和项目质量非常有帮助。

在实验内容 3 中，我探索了 GitHub 辅助工具的使用，如 GoodFirstIssue 和 Hypercrx 插件，这些工具为开源项目的发现和贡献提供了便捷的途径。同时，我也了解了 OpenLeaderboard 工具，它通过数据可视化帮助我分析开源项目的质量、进展和活跃度，这对我的项目选择和参与决策有着重要的影响。

总的来说，这次实验不仅提升了我的技术技能，也加深了我对开源文化和社区协作的理解。我感到非常兴奋和满足，因为我不仅学会了如何贡献开源项目，还体会到了作为开源社区一员的乐趣和挑战。这些经验对我未来的学习和职业生涯都将产生深远的影响。