

软件工程实验报告

实验五：交互式确认方法

院系：人工智能学院

姓名：方盛俊

学号：201300035

班级：人工智能 20 级 2 班

邮箱：201300035@smail.nju.edu.cn

时间：2022 年 11 月 16 日

目录

- 实验五：交互式确认方法
 - 目录
 - 一、版本控制
 - 1. Git Commit
 - 2. Git Diff
 - 3. Git Reset
 - 4. Git Revert
 - 5. Git Stash
 - 6. Git Checkout
 - 7. Git Merge
 - 8. Git Rebase
 - 9. Git Cherry Pick
 - 10. GitHub 远程仓库
 - 11. 分支合并图
 - 二、代码架构
 - 1. 后端模块 (Backend)
 - 1.1 主要执行模块
 - 1.2 基于 Flask 的服务器模块
 - 2. 前端模块 (Frontend)
 - 2.1 UI 设计
 - 2.2 MVC 设计模式
 - 2.3 用户交互
 - 3. 中间表示与等价类 (Cluster)
 - 三、运行流程
 - 1. 直接运行
 - 2. 更新前端
 - 3. 删除中间文件
 - 四、优秀设计
 - 1. 美观的 UI 以及拥有高亮的 Code Diff 界面
 - 2. 收起和展开侧边栏
 - 3. 用户自主切换 Diff 对的显示方式
 - 4. 简单易懂的用户提供交互选项
 - 5. CSV 文件下载按钮
 - 6. 自动保存 Clusters 当前数据
 - 6. 基于并查集的等价类

- 7. 测试样例的多样性
- 8. 并行多进程计算
- 9. 前后端分离

一、版本控制

1. Git Commit

执行 `git commit -m "feat: init"` 进行一次初始化的 Commit, 并附上 Commit 信息 `feat: init`。

```
(base) PS C:\Project\autodiff> git commit -m "feat: init"
[master (root-commit) ef06984] feat: init
35 files changed, 298 insertions(+)
create mode 100644 .gitignore
create mode 100644 cluster/__init__.py
create mode 100644 data/input/4A/101036360.cpp
create mode 100644 data/input/4A/117364748.cpp
create mode 100644 data/input/4A/127473352.cpp
create mode 100644 data/input/4A/134841308.cpp
```

执行前的 `git status` :

```
(base) PS C:\Project\autodiff> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   cluster/__init__.py
        new file:   data/input/4A/101036360.cpp
        new file:   data/input/4A/117364748.cpp
        new file:   data/input/4A/127473352.cpp
```

执行后的 `git status` :

```
(base) PS C:\Project\autodiff> 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:   lab4-report-201300035.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        images/2022-10-31-22-25-10.png

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

2. Git Diff

`main.py` 的内容原本为:

```
print('Hello World')
```

为了进行一次 Commit, 我们将其修改为:

```
print('Hello World')
print('Hello Software Engineering')
```

执行 `git add *` 后执行 `git commit -m "feat: change main.py"`, 将变动 Commit 到 Git 仓库中。

执行 `git diff` 我们可以发现我们的更改切实地被加入到了 Git 仓库中。

```
(base) PS C:\Project\autodiff> git diff
diff --git a/lab4-report-201300035.md b/lab4-report-201300035.md
index 9284f95..28c865a 100644
--- a/lab4-report-201300035.md
+++ b/lab4-report-201300035.md
@@ -176,3 +176,5 @@ print('Hello World')
    print('Hello Software Engineering')
\`\`\`
```

3. Git Reset

Git Reset 命令用于重置当前 HEAD 到指定的版本。

执行 `git log` 我们可以看出我们当前有 `feat: init` 和 `feat: change main.py` 这两条 Commit。

```
(base) PS C:\Project\autodiff> git log
commit 140993048c5c8d2464c4aa64b9d770c8cac45d84 (HEAD -> master)
Author: OrangeX4 <318483724@qq.com>
Date:   Mon Oct 31 22:29:34 2022 +0800

    feat: change main.py

commit 1ad1f9c5343a5ef3da9e099c090bda62970a0d1e
Author: OrangeX4 <318483724@qq.com>
Date:   Mon Oct 31 22:25:53 2022 +0800

    feat: init
```

执行 `git reset 1ad1f9` 我们就可以恢复到 `feat: init` 这一条 Commit 所在的位置。

```
(base) PS C:\Project\autodiff> git reset 1ad1f9
Unstaged changes after reset:
M      lab4-report-201300035.md
M      main.py
```

执行 `git log`，我们发现我们确实已经回到了只有一条 Commit 的状态。

```
(base) PS C:\Project\autodiff> git log
commit 1ad1f9c5343a5ef3da9e099c090bda62970a0d1e (HEAD -> master)
Author: OrangeX4 <318483724@qq.com>
Date:   Mon Oct 31 22:25:53 2022 +0800

    feat: init
```

4. Git Revert

使用 Git Revert 命令和 Git Reset 很类似，均是要恢复到之前的某些版本，但是 Git Revert 的好处在于，会把之前的 commit history 给保留下来，并把这次撤销作为一个新的 Commit。

执行 `git reset 1409930` 来恢复 `feat: change main.py` 这条 Commit，并执行 `git log` 显示：

```
(base) PS C:\Project\autodiff> git reset 1409930
Unstaged changes after reset:
M      lab4-report-201300035.md
(base) PS C:\Project\autodiff> git log
commit 140993048c5c8d2464c4aa64b9d770c8cac45d84 (HEAD -> master)
Author: OrangeX4 <318483724@qq.com>
Date:   Mon Oct 31 22:29:34 2022 +0800

    feat: change main.py

commit 1ad1f9c5343a5ef3da9e099c090bda62970a0d1e
Author: OrangeX4 <318483724@qq.com>
Date:   Mon Oct 31 22:25:53 2022 +0800

    feat: init
```

执行 `git revert HEAD`，我们就能撤销当前版本的修改，恢复上一个版本，并在不改变 commit history 的情况下，创建一个新的 Commit。

```
(base) PS C:\Project\autodiff> git revert HEAD
[master 800357d] Revert "feat: change main.py"
 4 files changed, 2 insertions(+), 29 deletions(-)
 delete mode 100644 images/2022-10-31-22-25-10.png
 delete mode 100644 images/2022-10-31-22-26-22.png
```

我们可以看出，`main.py` 文件的内容也成功回退到上一个版本了。

执行 `git log` 我们可以更清晰地看出我们做的操作：

```
(base) PS C:\Project\autodiff> git log
commit 800357d1fe7db268a45ba5681e8ffc3264330415 (HEAD -> master)
Author: OrangeX4 <318483724@qq.com>
Date: Mon Oct 31 23:20:58 2022 +0800

    Revert "feat: change main.py"

    This reverts commit 140993048c5c8d2464c4aa64b9d770c8cac45d84.

commit 140993048c5c8d2464c4aa64b9d770c8cac45d84
Author: OrangeX4 <318483724@qq.com>
Date: Mon Oct 31 22:29:34 2022 +0800

    feat: change main.py

commit 1ad1f9c5343a5ef3da9e099c090bda62970a0d1e
Author: OrangeX4 <318483724@qq.com>
Date: Mon Oct 31 22:25:53 2022 +0800

    feat: init
```

再次执行 `git revert HEAD`，我们可以发现 `main.py` 又恢复到了最新版本，这大概就是 `git revert` 操作的“负负得正”。

```
(base) PS C:\Project\autodiff> git revert HEAD
[master db3adbb] Revert "Revert "feat: change main.py""
4 files changed, 29 insertions(+), 2 deletions(-)
create mode 100644 images/2022-10-31-22-25-10.png
create mode 100644 images/2022-10-31-22-26-22.png
```

5. Git Stash

有时，当你在项目的一部分上已经工作一段时间后，所有东西都进入了混乱的状态，而这时你想要切换到另一个分支做一点别的事情。问题是，你不想仅仅因为过会儿回到这一点而为做了一半的工作创建一次提交。针对这个问题的答案是 `git stash` 命令。

这是 Git 官方文档对 Git Stash 命令的解释。事实上我也确实用到了这个命令。

在上一小节的 Git Revert 命令演示中，由于我是在同步编写实验报告，所以工作区实际上有一些变动，这导致我无法正常 `git revert`。

```
(base) PS C:\Project\autodiff> git revert HEAD
error: Your local changes to the following files would be overwritten by merge:
    lab4-report-201300035.md
Please commit your changes or stash them before you merge.
Aborting
fatal: revert failed
```

这种时候，我就可以执行 `git stash`，将我对实验报告的修改暂存了起来，这样工作区就又恢复了干净的状态，同时我们可以用 `git stash list` 查看我们放在栈上的暂存修改。

```
(base) PS C:\Project\autodiff> git stash
Saved working directory and index state WIP on master: 1409930 feat: change main.py
(base) PS C:\Project\autodiff> git stash list
stash@{0}: WIP on master: 1409930 feat: change main.py
```

在我们做完了其他工作，想要恢复暂存的修改的时候，就可以执行 `git stash apply` 将修改恢复过来了。

```
(base) PS C:\Project\autodiff> git stash apply
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:   lab4-report-201300035.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        images/2022-10-31-22-42-13.png
        images/2022-10-31-22-44-09.png
        images/2022-10-31-22-48-51.png
        images/2022-10-31-22-49-43.png
        images/2022-10-31-23-09-57.png

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

6. Git Checkout

执行 `git checkout -b dev` 创建 `dev` 分支，并使用 `git branch` 查看。

```
(base) PS C:\Project\autodiff> git checkout -b dev
Switched to a new branch 'dev'
(base) PS C:\Project\autodiff> git branch
* dev
master
```

使用 `git checkout master` 和 `git checkout dev` 可以切换分支。

```
(base) PS C:\Project\autodiff> git checkout master
Switched to branch 'master'
M       lab4-report-201300035.md
(base) PS C:\Project\autodiff> git checkout dev
Switched to branch 'dev'
M       lab4-report-201300035.md
```

7. Git Merge

我们修改 `main.py` 的内容为，并 Commit 到 `dev` 分支上：


```
print('Hello World')
print('Hello Software Engineering')
print('New Branch')
```

```
(base) PS C:\Project\autodiff> git add *
(base) PS C:\Project\autodiff> git commit -m "test: test branch function"
[dev fa05ee3] test: test branch function
15 files changed, 86 insertions(+), 1 deletion(-)
```

我们执行 `git checkout master` 后执行 `git merge dev` 进行合并。

```
(base) PS C:\Project\autodiff> git branch
* dev
  master
(base) PS C:\Project\autodiff> git checkout master
Switched to branch 'master'
(base) PS C:\Project\autodiff> git merge dev
Updating db3adbb..fa05ee3
Fast-forward
 images/2022-10-31-22-42-13.png | Bin 0 -> 24583 bytes
 images/2022-10-31-22-44-09.png | Bin 0 -> 34070 bytes
```

执行 `git log --graph --decorate --oneline --all` 可以看出中途 `dev` 分支被分离了出去，进行了一次 Commit 之后才被合并的。

```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* e501cbc (refs/stash) WIP on dev: fa05ee3 test: test branch function
| \
| * 09a4676 index on dev: fa05ee3 test: test branch function
| /
* fa05ee3 (HEAD -> master, dev) test: test branch function
* db3adbb Revert "Revert "feat: change main.py""
* 800357d Revert "feat: change main.py"
* 1409930 feat: change main.py
* 1ad1f9c feat: init
```

8. Git Rebase

除了 Git Merge 命令之外，还有一种可以对分支进行合并的命令，即 Git Rebase 命令。

要理解 Git Rebase 命令其实很简单，我们知道每个版本其实保存了一个类似于父节点的指针，通过父节点指针我们就可以从 HEAD 一路找到初始化版本，并通过 Git Log 显示出来。理论上来说这个父节点指针不应该由我们控制，应该由 Git 自己管理，但是执行 Git Rebase 操作可以让我们更改分支的父节点指针，让当前分支的父节点指向另一个分支的最新节点，也就是“变基”操作。

我们先在 master 分支对 `main.py` 进行一点修改：

```
print('Hello World')
print('Hello Software Engineering')
print('New Branch')
print('Change For Branch Master')
```

然后执行 `git add *` 和 `git commit -m "fix: change for branch master"` 进行 Commit。

执行 `git checkout dev` 切换到 `dev` 分支, 并且同样在 master 分支对 `main.py` 进行一点修改:

```
print('Hello World')
print('Hello Software Engineering')
print('New Branch')
print('Change For Branch Dev')
```

然后执行 `git add *` 和 `git commit -m "fix: change for branch dev"` 进行 Commit。

此时我们执行 `git log --graph --decorate --oneline --all` 看一下情况:

```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* f71d701 (HEAD -> dev) fix: change for branch dev
| * a937151 (refs/stash) WIP on master: 629e4e9 fix: change for branch master
| | \
| | * 81b020e index on master: 629e4e9 fix: change for branch master
| | /
| * 629e4e9 (master) fix: change for branch master
| /
* fa05ee3 test: test branch function
* db3adbb Revert "Revert "feat: change main.py""
* 800357d Revert "feat: change main.py"
* 1409930 feat: change main.py
* 1ad1f9c feat: init
```

最关键的一步来了, 我们执行 `git rebase master` 进行变基, 将 dev 的父节点设为 master 的最新节点。

```
(base) PS C:\Project\autodiff> git rebase master
Auto-merging lab4-report-201300035.md
CONFLICT (content): Merge conflict in lab4-report-201300035.md
Auto-merging main.py
CONFLICT (content): Merge conflict in main.py
error: could not apply f71d701... fix: change for branch dev
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply f71d701... fix: change for branch dev
(base) PS C:\Project\autodiff> git stash apply
lab4-report-201300035.md: needs merge
main.py: needs merge
```

我们再次执行 `git log --graph --decorate --oneline --all` 看一下情况:

```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* 20f3db5 (refs/stash) WIP on dev: f71d701 fix: change for branch dev
| \
| * a9f620e index on dev: f71d701 fix: change for branch dev
| /
* f71d701 (dev) fix: change for branch dev
| * 629e4e9 (HEAD, master) fix: change for branch master
| /
* fa05ee3 test: test branch function
* db3adbb Revert "Revert "feat: change main.py""
* 800357d Revert "feat: change main.py"
* 1409930 feat: change main.py
* 1ad1f9c feat: init
```

最后我们执行 `git checkout master` 和 `git merge dev` 让 master 分支行进到最新的节点。

```
(base) PS C:\Project\autodiff> git checkout master
Switched to branch 'master'
(base) PS C:\Project\autodiff> git merge dev
Updating 629e4e9..fe4c32f
Fast-forward
 images/2022-11-01-10-33-28.png | Bin 0 -> 48594 bytes
 images/2022-11-01-10-39-34.png | Bin 0 -> 51341 bytes
 lab4-report-201300035.md       | 22 ++++++
 main.py                       | 3 ++-
 4 files changed, 23 insertions(+), 2 deletions(-)
 create mode 100644 images/2022-11-01-10-33-28.png
 create mode 100644 images/2022-11-01-10-39-34.png
```

9. Git Cherry Pick

对于多分支的代码库，将代码从一个分支转移到另一个分支是常见需求。这时分两种情况。一种情况是，你需要另一个分支的所有代码变动，那么就采用合并（`git merge`）。另一种情况是，你只需要部分代码变动（某几个提交），这时可以采用 Cherry Pick。

因此我们可以知道，Git Cherry Pick 命令很类似 Git Merge 命令，只不过可以选择合并某个 Commit 而不是合并某个分支。

我们执行 `git checkout dev` 切换到 dev 分支后，添加两个 Commit。

可以看出，我们当前有两个新的 Commit: `feat: change 1` 和 `feat: change 2`。

```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* 81b4b19 (HEAD -> dev) feat: change 2
* 40fad27 feat: change 1
| * 2b0cad4 (refs/stash) WIP on master: fe4c32f fix: change for branch dev
|/|
| * 62795de index on master: fe4c32f fix: change for branch dev
|/
* fe4c32f (master) fix: change for branch dev
* 629e4e9 fix: change for branch master
* fa05ee3 test: test branch function
* db3adbb Revert "Revert "feat: change main.py""
* 800357d Revert "feat: change main.py"
* 1409930 feat: change main.py
* 1ad1f9c feat: init
```

我们执行 `git checkout master` 切换到 master 分支后，然后执行

`git cherry-pick 40fad27` 只合并 dev 分支的 `feat: change 1`。

```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* 64e5807 (HEAD -> master) feat: change 1
| * 0d2f779 (refs/stash) WIP on dev: 81b4b19 feat: change 2
| \
| | * 67b78ff index on dev: 81b4b19 feat: change 2
| | /
| | * 81b4b19 (dev) feat: change 2
| | * 40fad27 feat: change 1
| /
* fe4c32f fix: change for branch dev
* 629e4e9 fix: change for branch master
* fa05ee3 test: test branch function
* db3adbb Revert "Revert "feat: change main.py""
* 800357d Revert "feat: change main.py"
* 1409930 feat: change main.py
* 1ad1f9c feat: init
```

可以看到，我们的 master 分支确实合并了 `feat: change 1` Commit。


10. GitHub 远程仓库

我们先在 GitHub 上创建一个新的 Repo。

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 OrangeX4 ▾

Repository name *

/ autodiff ✓

Great repository names are short and memorable. Need inspiration? How about [animated-dollop](#)?

Description (optional)

autodiff - software engineering project



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

执行

```
git remote add origin https://github.com/OrangeX4/autodiff.git
git push -u origin master
git push origin dev
```

将所有分支推送上去。

```
(base) PS C:\Project\autodiff> git remote add origin https://github.com/OrangeX4/autodiff.git
(base) PS C:\Project\autodiff> git push -u origin master
Enumerating objects: 93, done.
Counting objects: 100% (93/93), done.
Delta compression using up to 8 threads
Compressing objects: 100% (84/84), done.
Writing objects: 100% (93/93), 900.35 KiB | 18.01 MiB/s, done.
Total 93 (delta 23), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (23/23), done.
To https://github.com/OrangeX4/autodiff.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

可以看见在地址 (<https://github.com/OrangeX4/autodiff>) 已经成功推送了上去。

dev had recent pushes 1 minute ago

Compare & pull request

master ▾

2 branches

0 tags

Go to file

Add file ▾

Code ▾

OrangeX4 docs: complete part of git	22e053b 12 minutes ago	9 commits
cluster	feat: init	13 hours ago
data/input	feat: init	13 hours ago
diff	feat: init	13 hours ago
generator	feat: init	13 hours ago
images	docs: complete part of git	12 minutes ago
input	feat: init	13 hours ago

About

autodiff - software engineering project

0 stars

1 watching

0 forks

Releases

No releases published

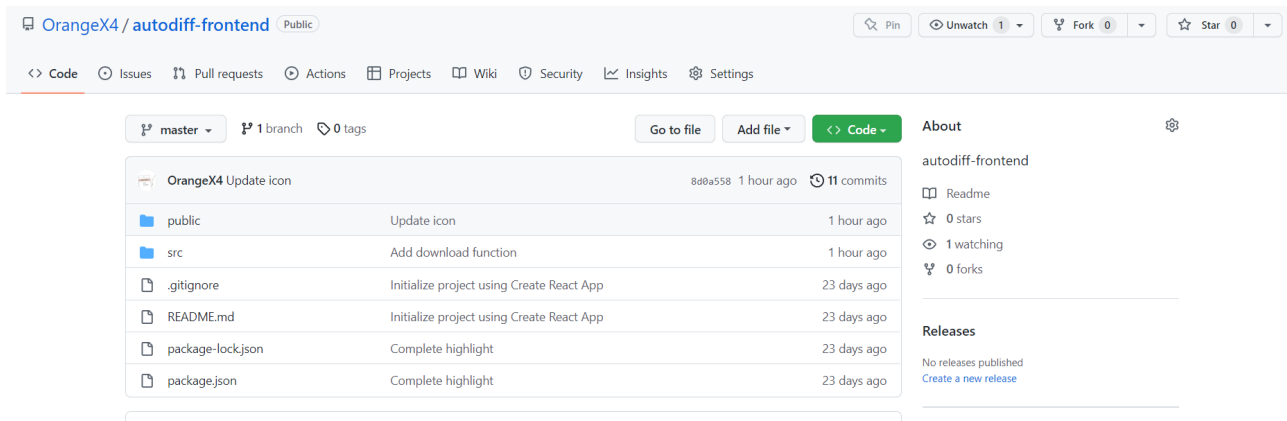
Create a new release

Packages

No packages published

Publish your first package

除了后端部分的 Repo，我还创建了一个前端部分的 Repo (<https://github.com/OrangeX4/autodiff-frontend>)。



11. 分支合并图

这次实验中，我一共使用了 8 个分支，每个分支对应一个模块，而 master 是主分支。

```
(base) PS C:\Project\autodiff> git branch
cluster
dev
diff
generator
input
* master
output
paracomp
```

使用 `git log --graph --decorate --oneline --all` 展示的分支合并图如下：

```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* 08c4742 (HEAD -> master, server) feat: complete server
* 9f84e20 Add CSV function
* e778f5f Merge branch 'output'
| \
| * e97a218 (output) feat: add cluster_to_diff_list
* | 68e697a (origin/master) docs: update README.md
* | c3f9b7a docs: complete report
| /
* 804c7f0 (origin/output) docs: complete output
* 7360bf6 feat: complete output
```



```
(base) PS C:\Project\autodiff> git log --graph --decorate --oneline --all
* 804c7f0 (HEAD -> master, output) docs: complete output
* 7360bf6 feat: complete output
* 933b4d5 (paracomp) feat: add clean for paracomp
* 70f8df8 (diff) feat: add clean_cmd for diff
* b5ed014 docs: complete paracomp part
* fa5dc56 fix: fix paracomp
* a1013d0 feat: complete paracomp
| * c88463c (refs/stash) WIP on diff: ce75689 Merge branch 'generator'
|/|
| * 1230f40 index on diff: ce75689 Merge branch 'generator'
|/
* ce75689 Merge branch 'generator'
|\
| * 8547816 (generator) feat: update generator
* | 7ed9fe1 (input) docs: complete input part
* | c9cfe2f feat: complete input
* | f80f9be (cluster) docs: complete cluster part
* | cb7cf3b feat: complete cluster
|/
* 8566c66 (origin/master) docs: complete generator part
* a2ef204 feat: complete generator
* 87b17d0 docs: complete diff part
* fdd264f feat: add more languages
* f488b74 feat: complete diff
* 3557b52 docs: add github
* 22e053b docs: complete part of git
* 64e5807 feat: change 1
| * 81b4b19 (origin/dev, dev) feat: change 2
| * 40fad27 feat: change 1
|/
* fe4c32f fix: change for branch dev
* 629e4e9 fix: change for branch master
* fa05ee3 test: test branch function
* db3adbb Revert "Revert "feat: change main.py""
* 800357d Revert "feat: change main.py"
* 1409930 feat: change main.py
* 1ad1f9c feat: init
```

前端部分为：

```
(base) PS C:\Project\autodiff-frontend> git log --graph --decorate --oneline --all
* 8d0a558 (HEAD -> master, origin/master) Update icon
* da398f6 Add download function
* 716d317 feat: complete almost all function
* 3600820 feat: complete basic ui
* 586a84f feat: add basic sidebar function
* 107aa55 Add sidebar
* 18c4df8 Add status
* 388d0ba Complete highlight
* f208164 Add code diff
* 329bc1d Complete basic ui
* 9f1dddd Initialize project using Create React App
```

二、代码架构

1. 后端模块 (Backend)

1.1 主要执行模块

后端主要执行模块已经在 lab4 中讲述了，这里就不过多赘述。

大体的架构如下：

- 输入模块 (InputModule) (从一个文件夹内读取)
 - 读取源文件 (readInputFiles)
 - 读取保存的中间表示 (readDataFromFiles)
- 中间表示模块 (Data)
 - Cluster
 - 目录名 (Cluster)
 - 测试样例生成 (RandomTestGenerator)
 - 自定义测试样例 (CustomTests)
 - 文件名 (FileName)
 - 文件内容 (FileContent)
 - 并查集等价类 (EquivClass)
 - Compare
 - 文件名 1 (FileName1)
 - 文件名 2 (FileName2)
 - 自动比较结果 (AutoResult)
 - 人工确认结果 (ManualResult)
- 输出模块 (OutputModule) (输出到一个文件夹)
 - 输出 CSV 文件 (writeCSVFiles)
 - 输出中间表示 (writeDataToFiles)
- 多进程并行计算模块 (ParallelComputing)
 - 根据 Cluster 和 FileName 生成工作 (generateWorks)
 - 调用 AutoCompare 进行文件的比较 (dispatch)
- 自动比较模块 (AutoCompare)
 - 根据文件后缀名判断文件类型 (c or cpp)
 - 通过 GCC 等方式编译然后运行 (compare)

1.2 基于 Flask 的服务器模块

为了前后端分离，我们还需要一个能够提供后端接口的服务器模块 (Server)，这里我选用了 Flask 模块来实现后端服务器。

Flask 模块可以很简单地就为前端模块提供必要的功能，例如读取 Clusters 数据文件等：


```
def get_cluster(cluster_name) -> Dict:
    ...
    获取 cluster dict 的函数, 并且更新 diff_list
    ...

    cluster = clusters[cluster_name]
    cluster_dict = cluster.cluster
    cluster_dict['diff_list'] = output.cluster_to_diff_list(cluster_name)
    return cluster_dict

@app.route("/clusters", methods=['GET'])
def handle_clusters():
    result = {}
    for cluster_name in clusters:
        result[cluster_name] = get_cluster(cluster_name)
    return result
```

其中方法注解 `@app.route("/clusters", methods=['GET'])` 代表着将这个 API 映射到 `http://host:port/clusters` 上, 可以通过 GET 方法来获取到所有的 clusters 数据。

同理, 我还实现了其他接口:

- `http://host:port/clusters` (GET): 获取到所有的 clusters 数据;
- `http://host:port/cluster/<cluster_name>` (GET): 获取某个 cluster 数据;
- `http://host:port/run` (POST): 为某个 cluster 执行自动并行比较;
- `http://host:port/update` (POST): 为某个 cluster 更新人工比较结果;
- `http://host:port/csv` (GET): 下载当前的 CSV 文件内容。

为了将前端后端模块合并, 这里我们需要使用 `npm run build` 生成编译后的前端静态文件, 然后将静态文件移动至后端 `./server/static` 文件夹下。

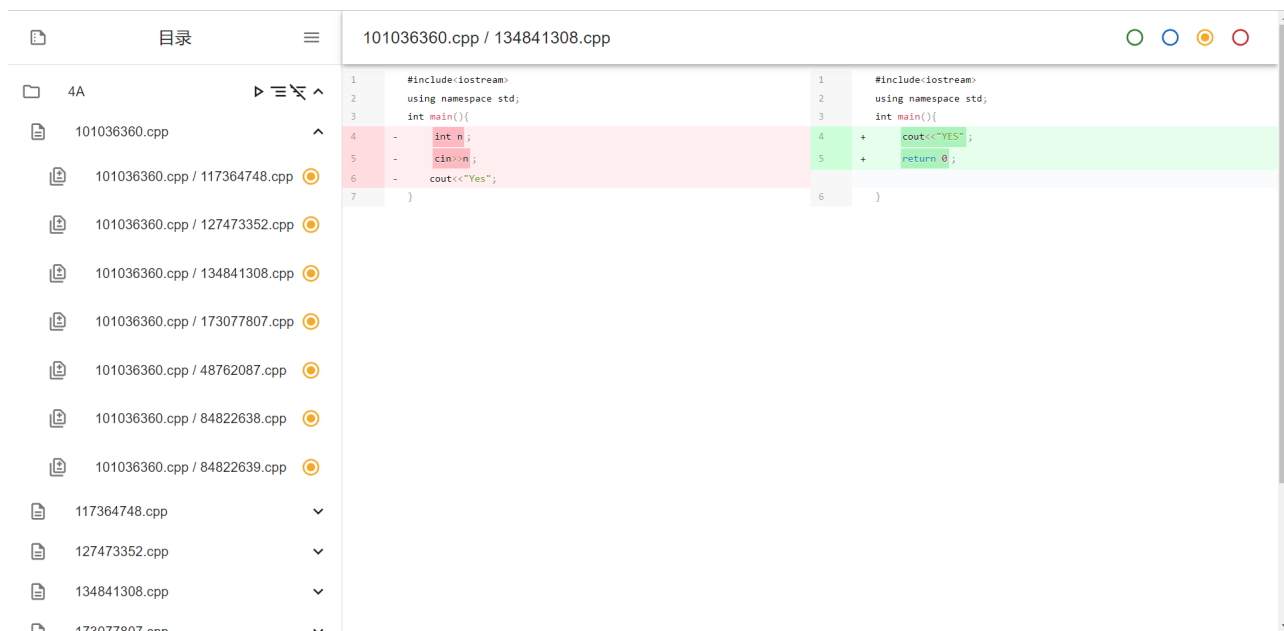
然后我们就可以通过 Flask 托管我们的前端静态文件了:

```
# Flask
app = Flask(__name__,
            static_url_path='',
            static_folder='static')
```

2. 前端模块 (Frontend)

2.1 UI 设计

前端模块使用 **React**、**material-ui**、**react-diff-viewer** 以及高亮库 **Prism** 实现。

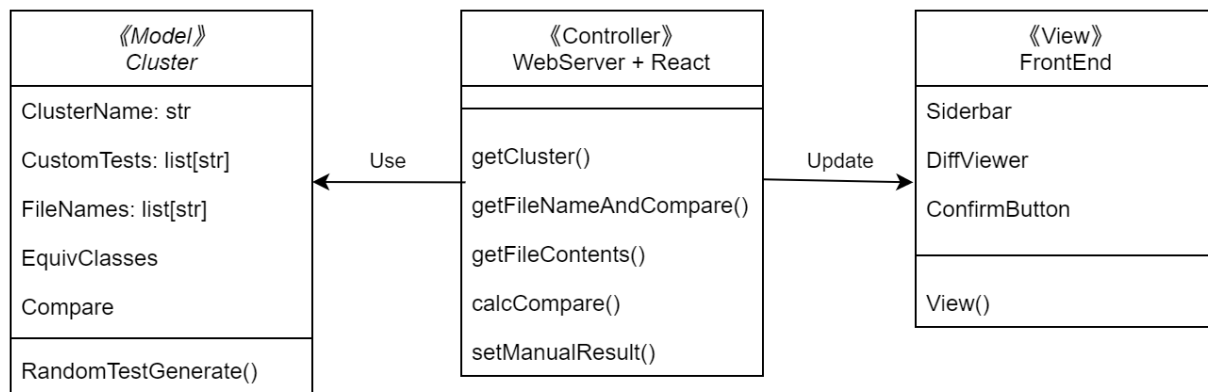


如图所示，主要分为三大部分：

- 侧边栏目录模块 (Sidebar)
 - 侧边栏左上角的按钮可以「**下载 CSV 文件**」
 - 侧边栏右上角的目录可以切换 「**是否收起侧边栏**」
 - 每个文件夹作为一个 Cluster
 - 每个 Cluster 有四个按钮
 - **执行按钮**，调用后端等价判断工具进行比较
 - **显示按钮**，切换显示模式 (Diff 模式 / Folder 模式)
 - **过滤按钮**，切换是否过滤已判定的文件对 (过滤掉绿色和红色 Diff)
 - **收起按钮**，切换展开或者收起
 - 每个 Diff 行后有一个“状态”
 - 绿色代表“人工确认等价”
 - 蓝色代表“自动判断等价”
 - 黄色代表“仍未进行判断”
 - 红色代表“不等价”
- Diff 显示模块 (DiffViewer)
 - 显示两个代码文件之间的差异
- 人工确认模块 (ConfirmButton)
 - 右上角有四个按钮，可以进行人工的等价确认
 - 绿色代表“人工确认等价”
 - 蓝色代表“自动判断等价”
 - 黄色代表“仍未进行判断”
 - 红色代表“不等价”

2.2 MVC 设计模式

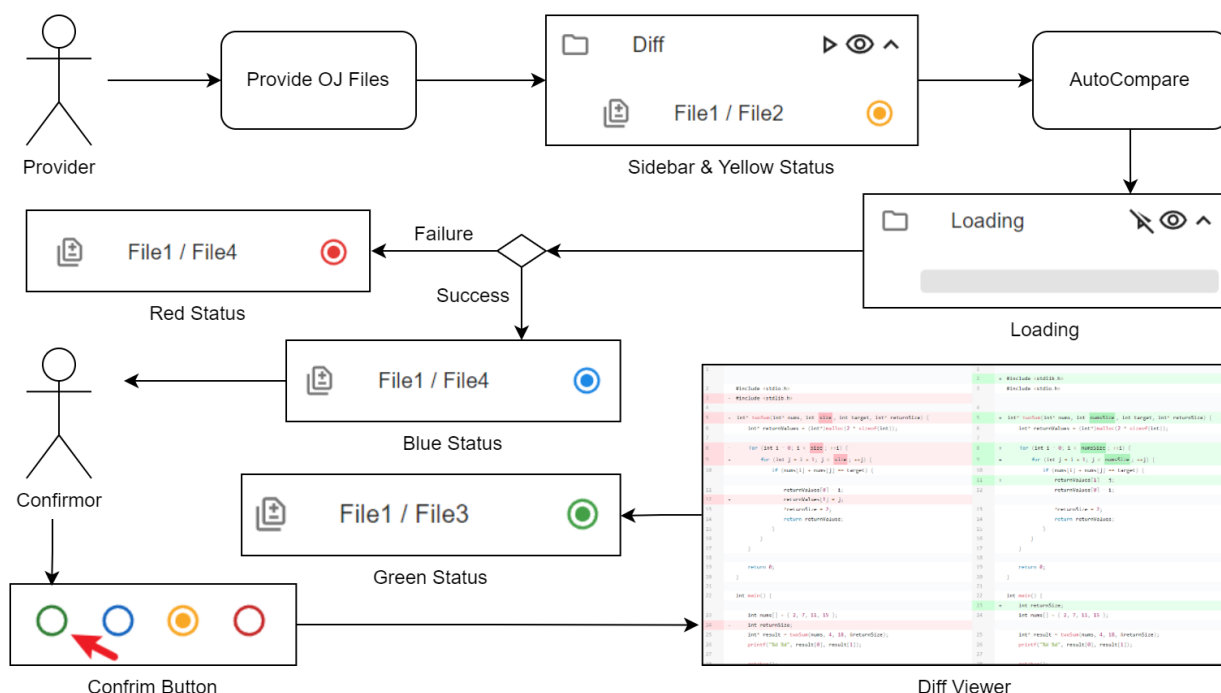
前端模块使用了 MVC 设计模式。



- **Model**: Cluster 类作为 Model，存储了用到的目录、文件以及等价判断结果等信息。
- **Controller**: WebServer 模块和 React 框架作为中间层，负责连接 Model 和 View。Controller 从 Model 中获取数据信息，并且实时地映射到前端的 View 中。
- **View**: 由侧边栏 (Sidebar)、Diff 显示 (DiffViewer) 以及等价确认按钮 (ConfirmButton) 等部分组成的 View，用于实时显示 Model 中拥有的数据，并对用户操作做出反馈。

2.3 用户交互

使用 **活动图** 表示具体的过程如下：



用户的关键在于处理好每个文件 Diff 对的四种状态：

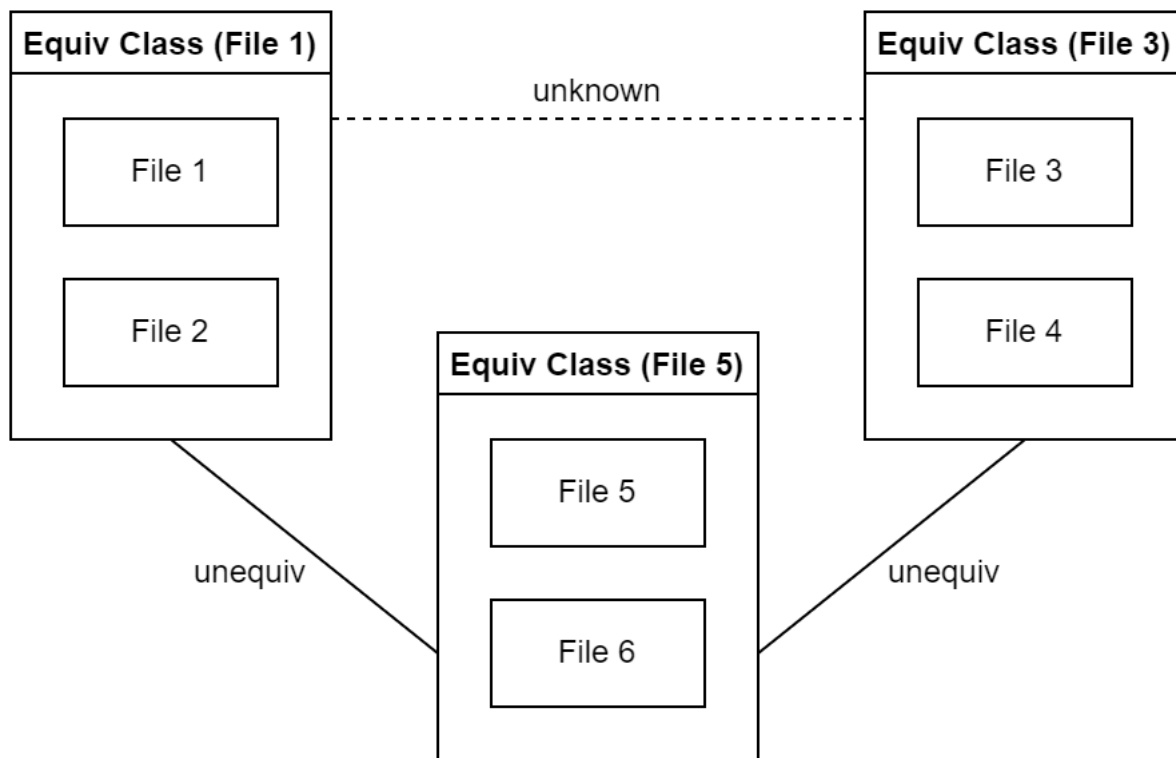
- 绿色代表“人工确认等价”
- 蓝色代表“自动判断等价”
- 黄色代表“仍未进行判断”
- 红色代表“不等价”

3. 中间表示与等价类 (Cluster)

我们使用一个中间表示类 Cluster，来表示我们当前处理的数据的状态。

```
{
  "cluster_name": "4A",
  "random_input_generator": {
    "type": "stdin_format.txt",
    "content": "int(1, 3)"
  },
  "custom_input": [ "1", "2" ],
  "config": {
    "random_test_times": 10,
    "random_seed": 0,
  },
  "files": {
    "48762087.cpp": {
      "content": "int main() { return 0; }",
      "equiv_class": "48762087.cpp"
    },
    "84822638.cpp": {
      "content": "int main() { return 0; }",
      "equiv_class": "48762087.cpp"
    }
  },
  "equiv": [ ["48762087.cpp", "84822638.cpp"] ],
  "unequiv": [],
  "diff": {
    "48762087.cpp": {
      "84822638.cpp": {
        "auto": "unknown",
        "manual": "equiv",
        "logic": "equiv"
      }
    },
    "84822638.cpp": {
      "48762087.cpp": {
        "auto": "unknown",
        "manual": "equiv",
        "logic": "equiv"
      }
    }
  }
}
```

其中 `files` 属性下的 `equiv_class` 用的是一个 **并查集结构**，我们用并查集来标识该文件所属的 **等价类**。



如图所示：

- 在同一个等价类里的文件被认为是 **相互等价** 的，通过一个根文件来标识等价类，例如下面的那个等价类通过 `File 5` 标识。
- 在不同等价类之间，通过实线连接的等价类被认为是 **不等价** 的，即两个等价类里的文件两两匹配都不等价。
- 在不同等价类之间，通过虚线连接的等价类被认为是 **未知等价关系** 的，需要用户进一步地判断 (虚线是初始化后就存在的)。

借助这幅图所示的等价类概念，以及并查集的算法知识，我们就可以写出一个能够让用户 **手动动态更改等价关系** 的 `Cluster` 类。

由于代码有数百行，就不在报告中过多展示，感兴趣可以查阅代码 `cluster/__init__.py`。

代码还包括了一个简易的单元测试：

```
cluster = Cluster('test', cluster)
cluster.clear()
cluster.set_manual('1', '2', 'equiv')
cluster.set_manual('2', '3', 'equiv')
cluster.set_manual('2', '3', 'unequiv')
cluster.set_manual('2', '3', 'unknown')
cluster.set_manual('3', '4', 'equiv')
cluster.set_auto('2', '3', 'unequiv')
cluster.update_diff()
print(cluster.cluster)
```

可以看出，接口是很易用的，在等价确认工具中便使用了这个接口。

三、运行流程

1. 直接运行

第一步，需要先将后端项目 clone 下来

(或者直接移动到 `./code/autodiff` 文件夹下，和重新 clone 效果等同)

```
git clone https://github.com/OrangeX4/autodiff.git
```

第二步，安装 Python 开发环境 (并且安装 Flask 模块)。

```
pip install Flask
```

第三步，往 `data/input` 里加入数据，例如 `4A/xxx.cpp`、`50A/xxx.cpp` 等。

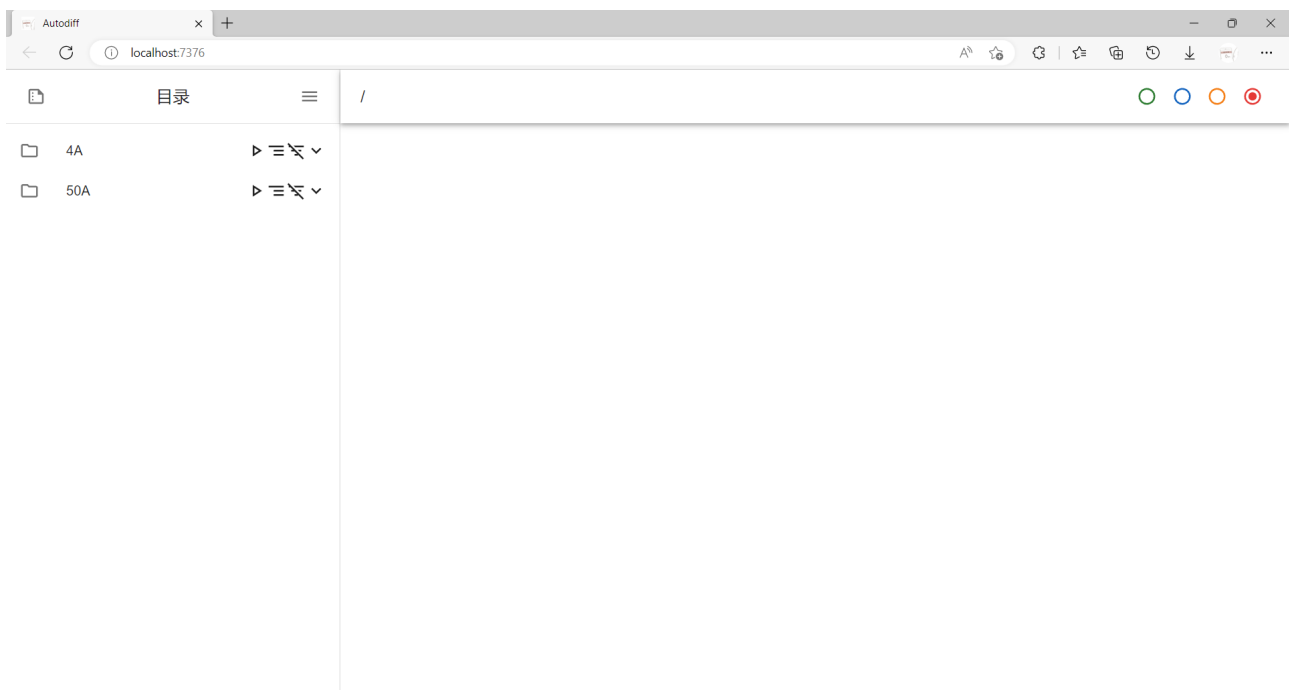
第四步，切换到 `server` 模块下。

```
cd ./server
```

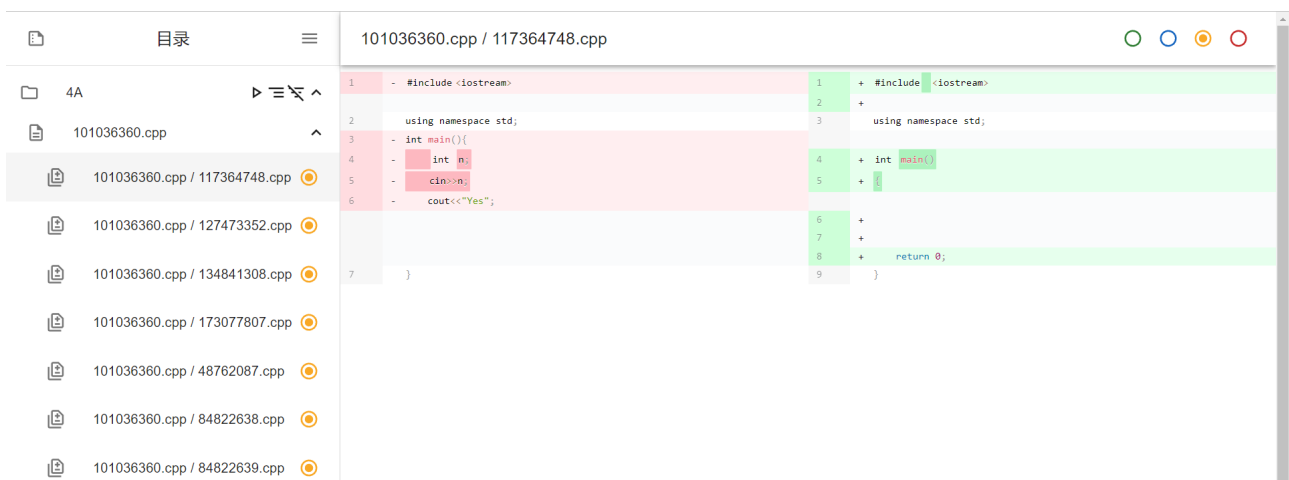
第五步，执行 `python __init__.py` 以运行服务器模块：

```
python __init__.py
```

第六步，此时会自动打开一个页面，这就是我们的操作界面，对应的地址为 <http://localhost:7376/>。

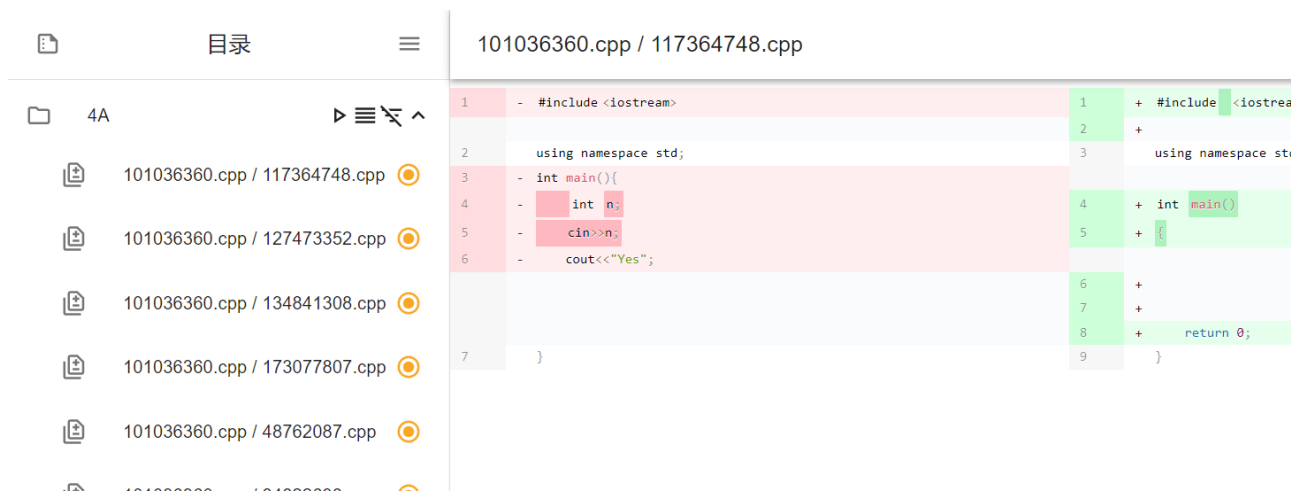


第七步，打开某个 Cluster 对应的 Diff，此时发现每一行 Diff 显示都还有「黄色」标志，这代表着还未进行「程序自动随机测试比对」。

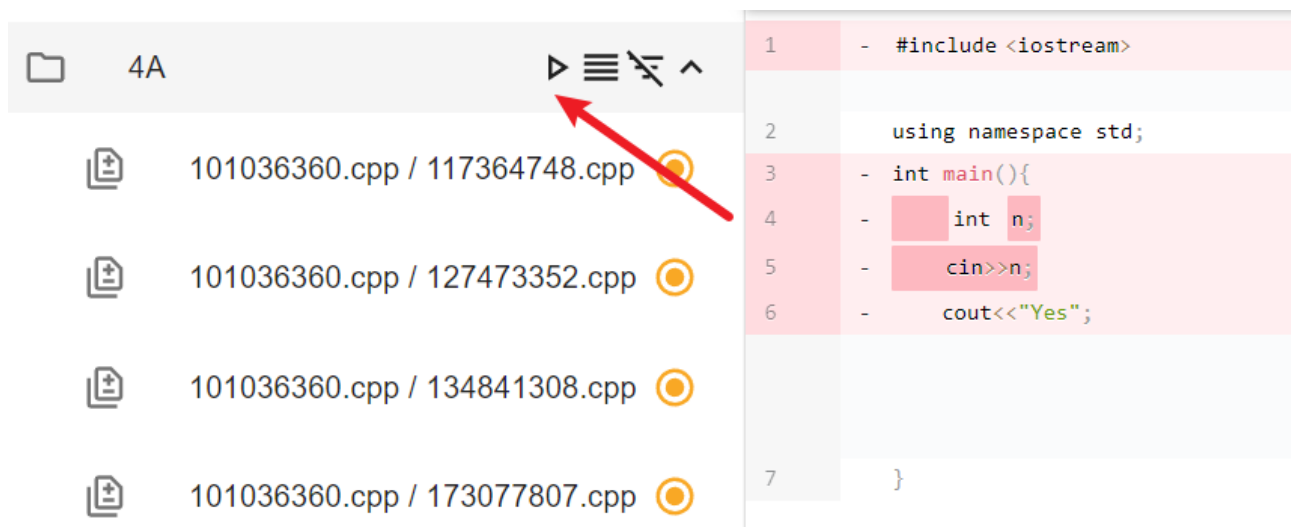


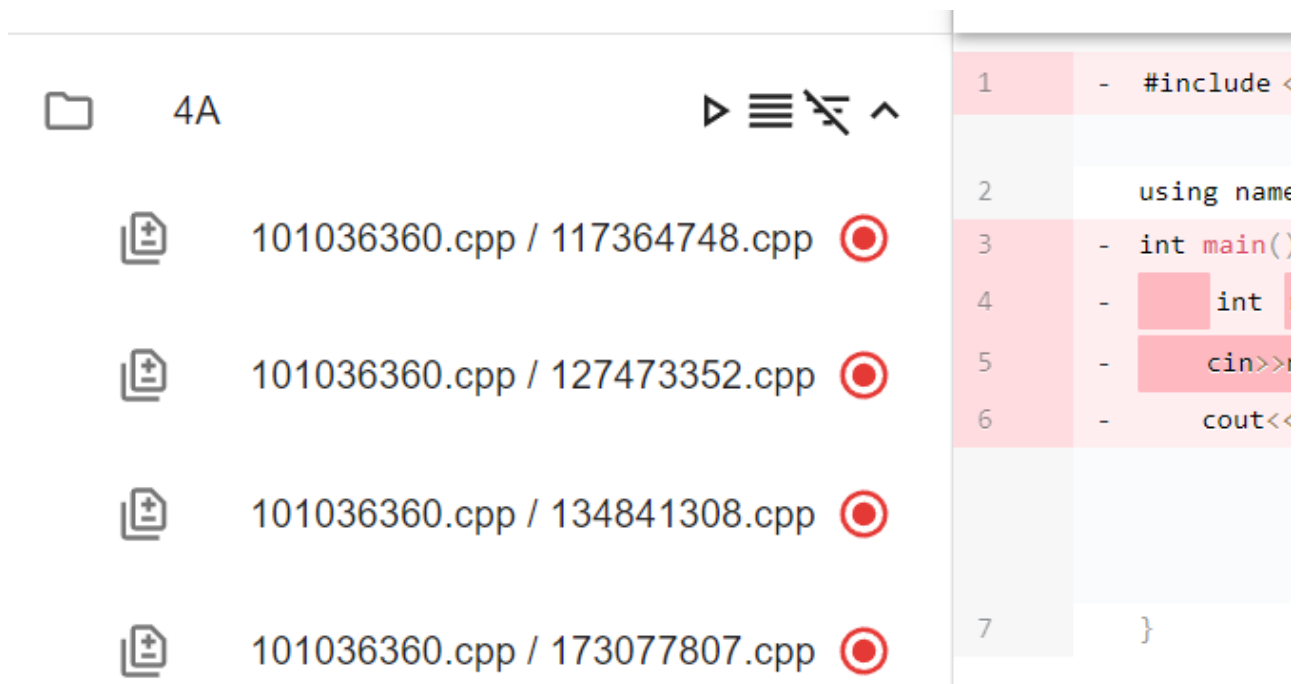
第八步，点击切换显示方式按钮，可以从「文件树模式」转换为「列表模式」。



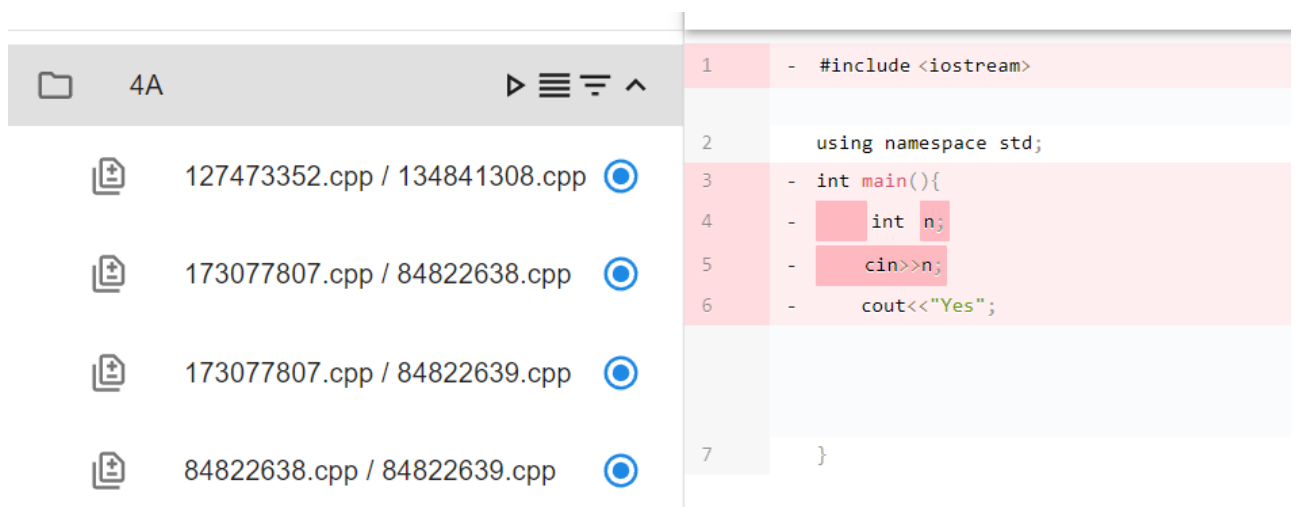
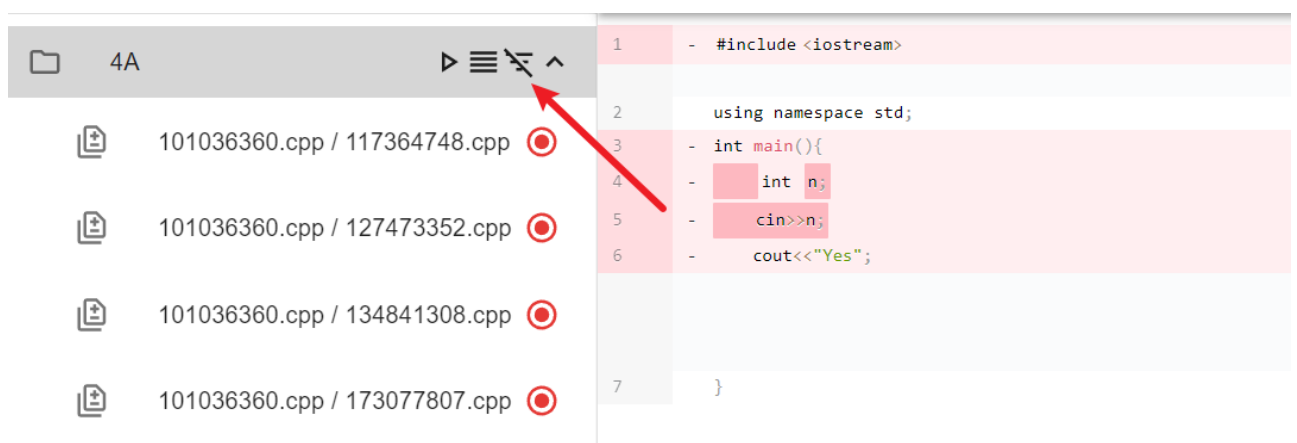


第九步，点击运行按钮，可以进行 **自动执行比对**。

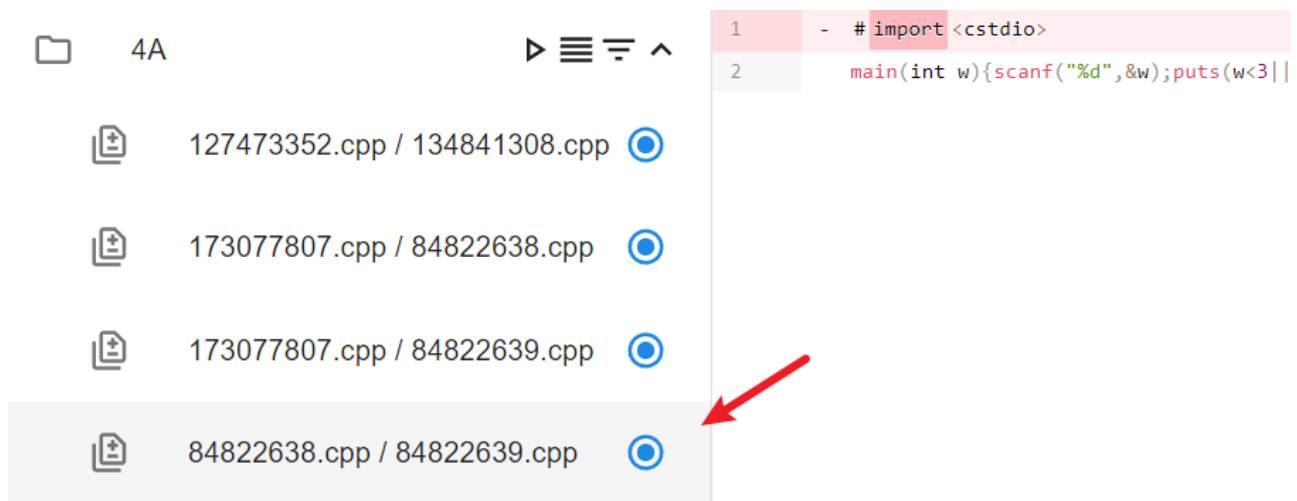




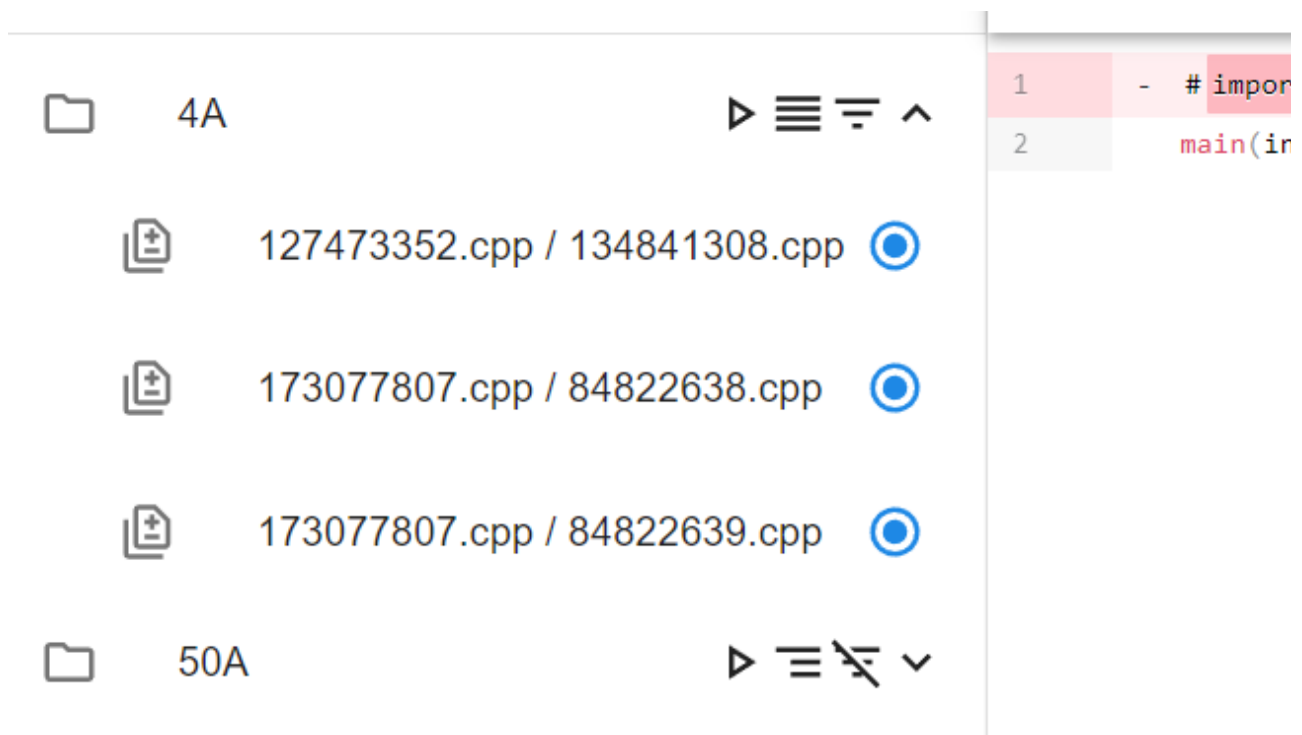
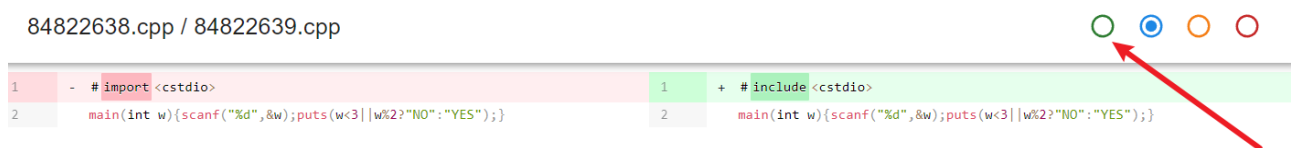
第十步，此时可以看见已经出现了红色的状态，我们可以点击过滤按钮，过滤掉「绿色」和「红色」的 Diff。



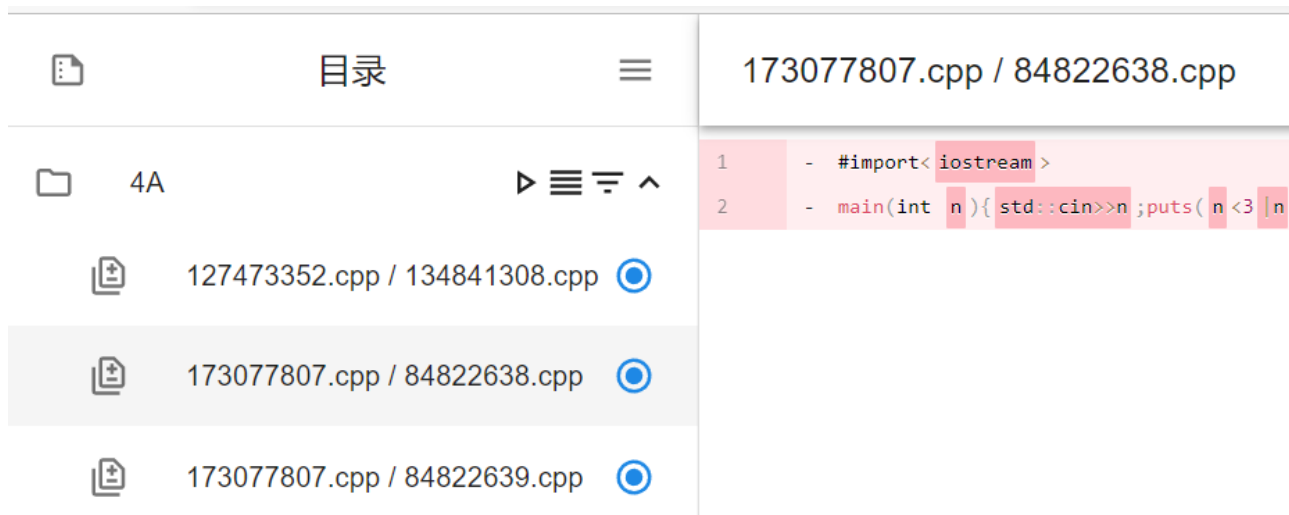
第十一步，点开某个 Diff，例如这里的 84822638.cpp / 84822639.cpp，这里显示为蓝色，证明其已经通过了自动随机测试，对于相同的输出有着相同的输出结果。



第十二步，我们人工判断这个 Diff 相等，我们点击「绿色按钮」，将其人工分类为「等价」，此时侧边栏要人工确认的文件对从 4 个变为 3 个。



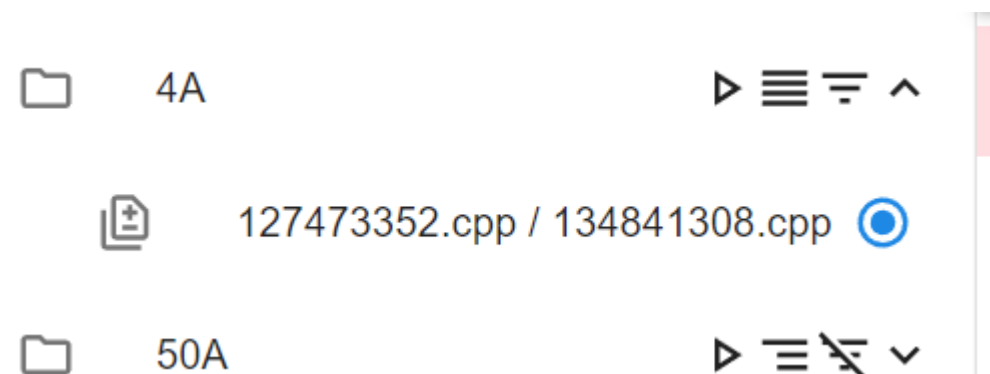
第十三步，同理，如果我们将 173077807.cpp / 84822638.cpp 其人工分类为「等价」。



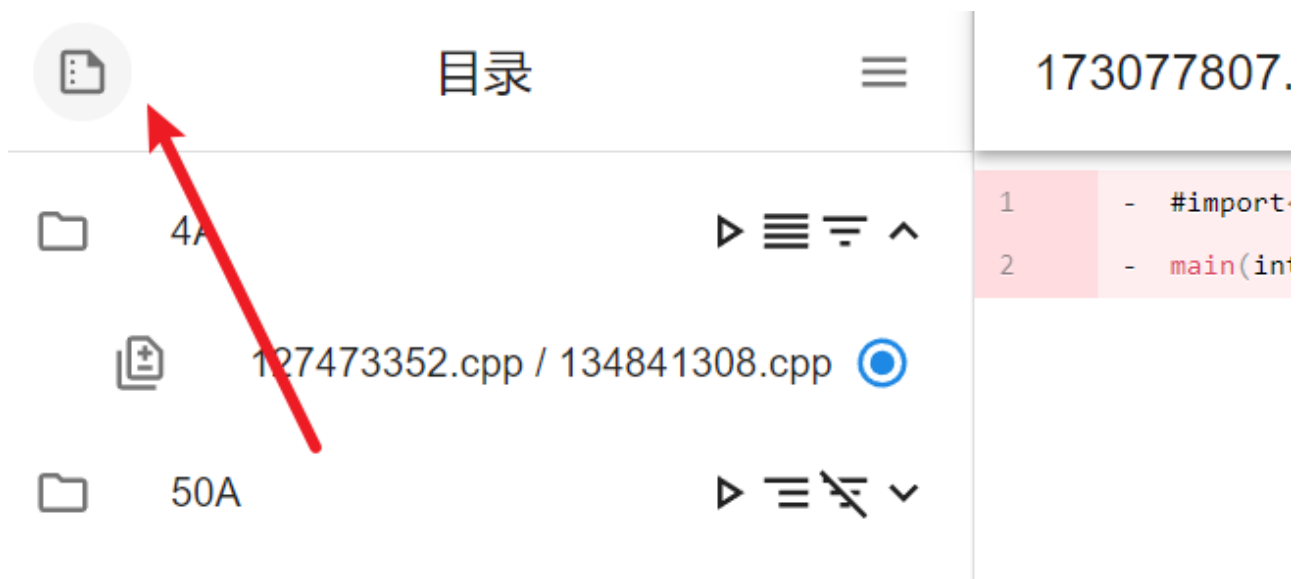
由于 84822638.cpp / 84822639.cpp 也是等价的，因此 173077807.cpp / 84822639.cpp 也会自动判定为等价。

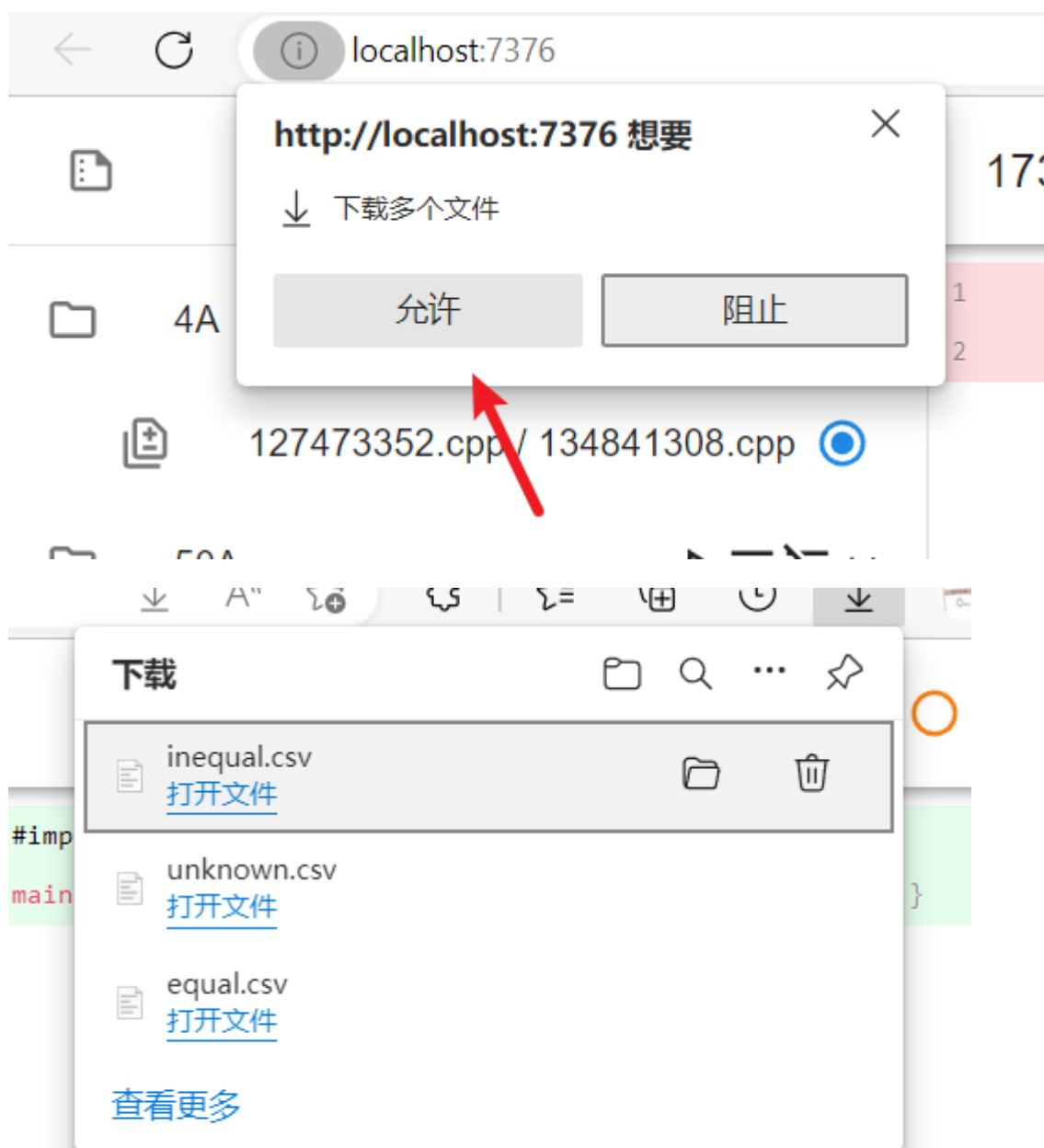
这说明我们通过等价类的方式，实现了根据用户的人工判定结果，自动推荐下一个需人工确认的等价对的功能。

此时侧边栏要人工确认的文件对从 3 个变为 1 个。



第十四步，通过左上角的「下载 CSV 文件」按键下载当前的 CSV 文件。





2. 更新前端

如果需要更新前端代码，可以执行

```
cd ./code/autodiff-frontend
```

切换到前端项目，然后执行

```
npm start
```

即可开启「开发模式」，此时修改代码的话，页面会自动「热更新」。

编辑完成后，执行

```
npm run build
```

可以生成编译后的前端静态文件 `./build`，这时我们将其移动到后端的

```
./server/static
```

目录中，注意要将 `build` 整个文件夹移动过去，删除原来的 `./server/static`，然后将 `build` 重命名为 `static`。

3. 删除中间文件

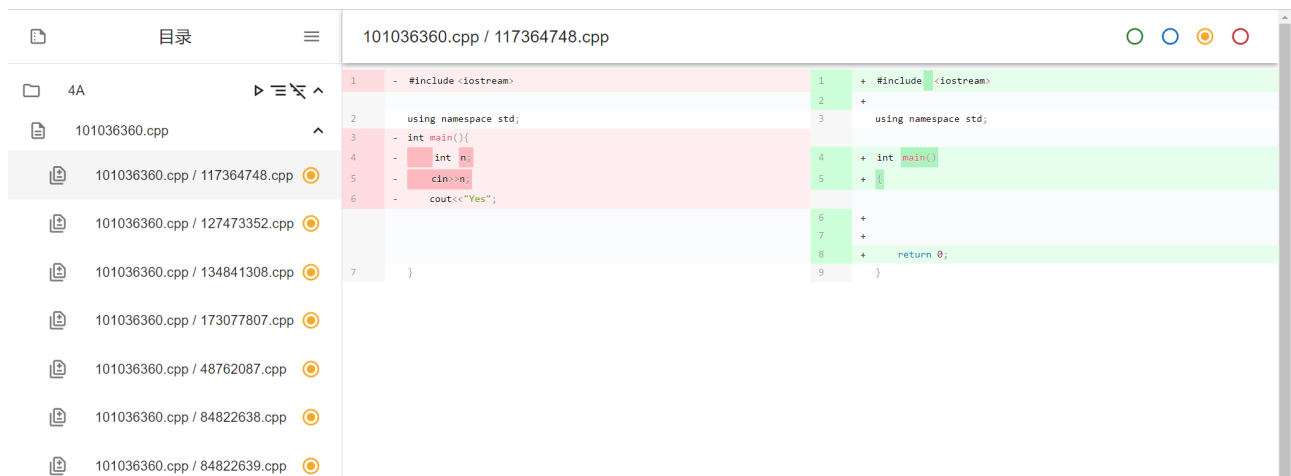
如果你想要消除你已经执行的操作，或者想要删除自动随机测试的结果，你需要删除 `./data/clusters` 下的所有文件，例如

```
./data/clusters/4A.json  
./data/clusters/50A.json
```

这些文件是给不同 Clusters 保存的中间结果。

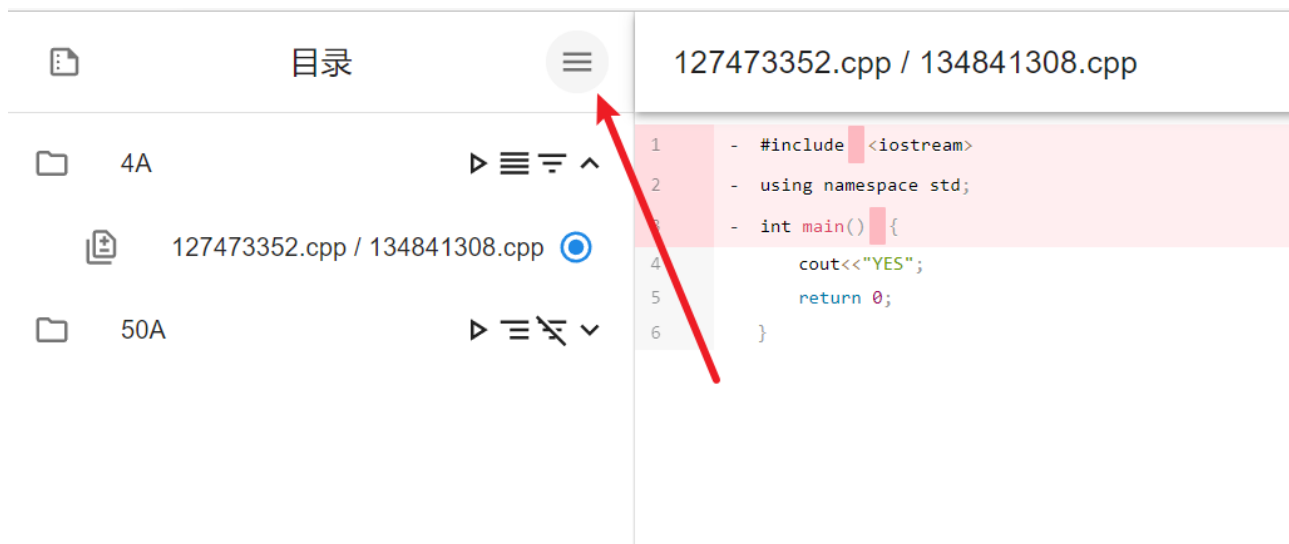
四、优秀设计

1. 美观的 UI 以及拥有高亮的 Code Diff 界面

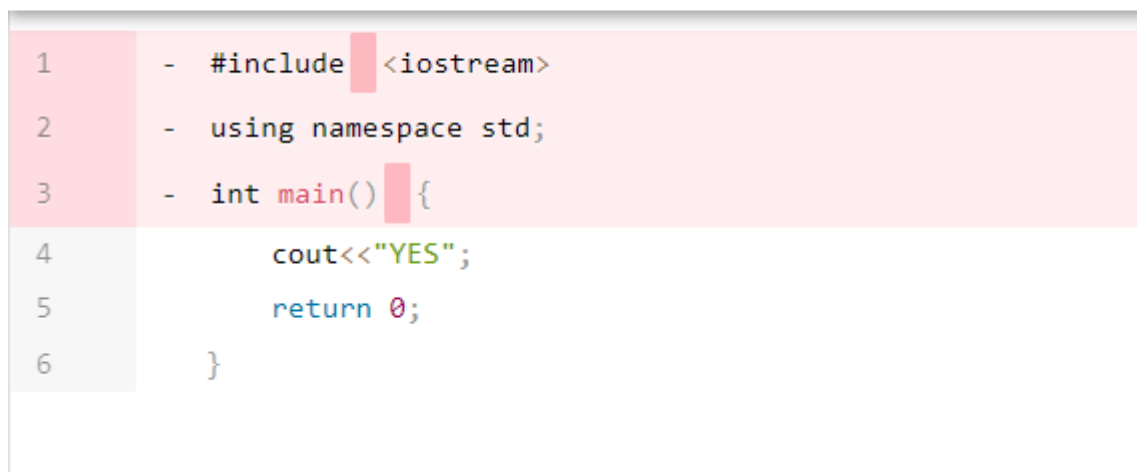


我为项目加入了类似 VS Code 的 Code Diff 界面，可以让用户更简单地进行人工确认。

2. 收起和展开侧边栏



≡ 127473352.cpp / 134841308.cpp



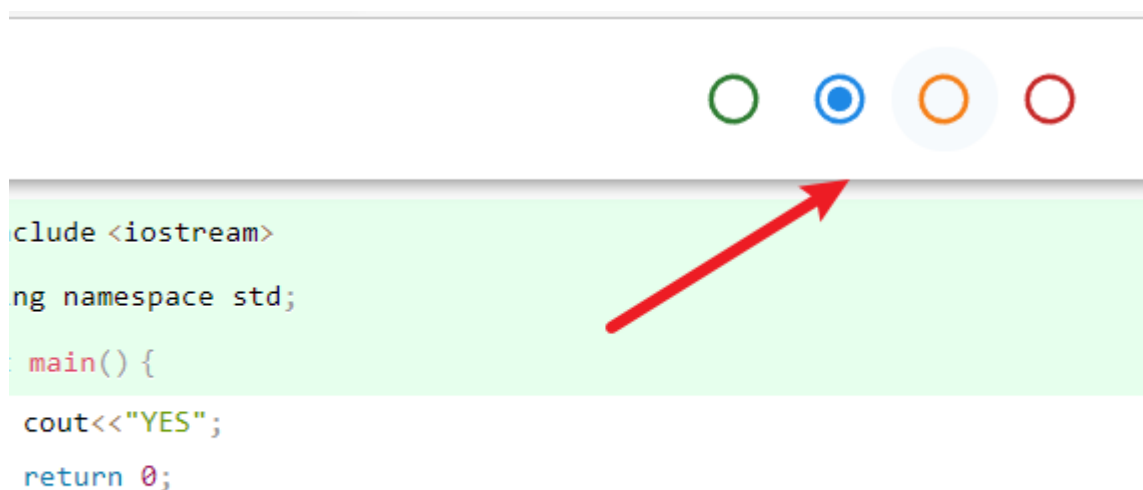
这个按钮可以用于收起和展开侧边栏。

3. 用户自主切换 Diff 对的显示方式



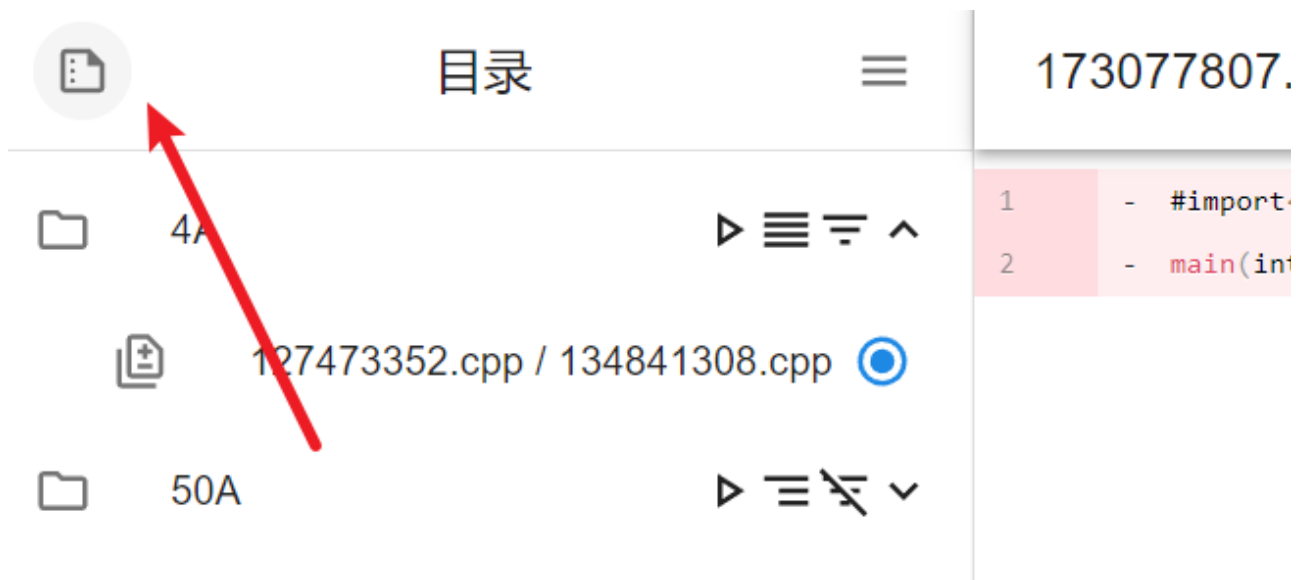
「显示按钮」和「过滤按钮」可以让用户自主切换 Diff 对的显示方式。

4. 简单易懂的用户提供交互选项



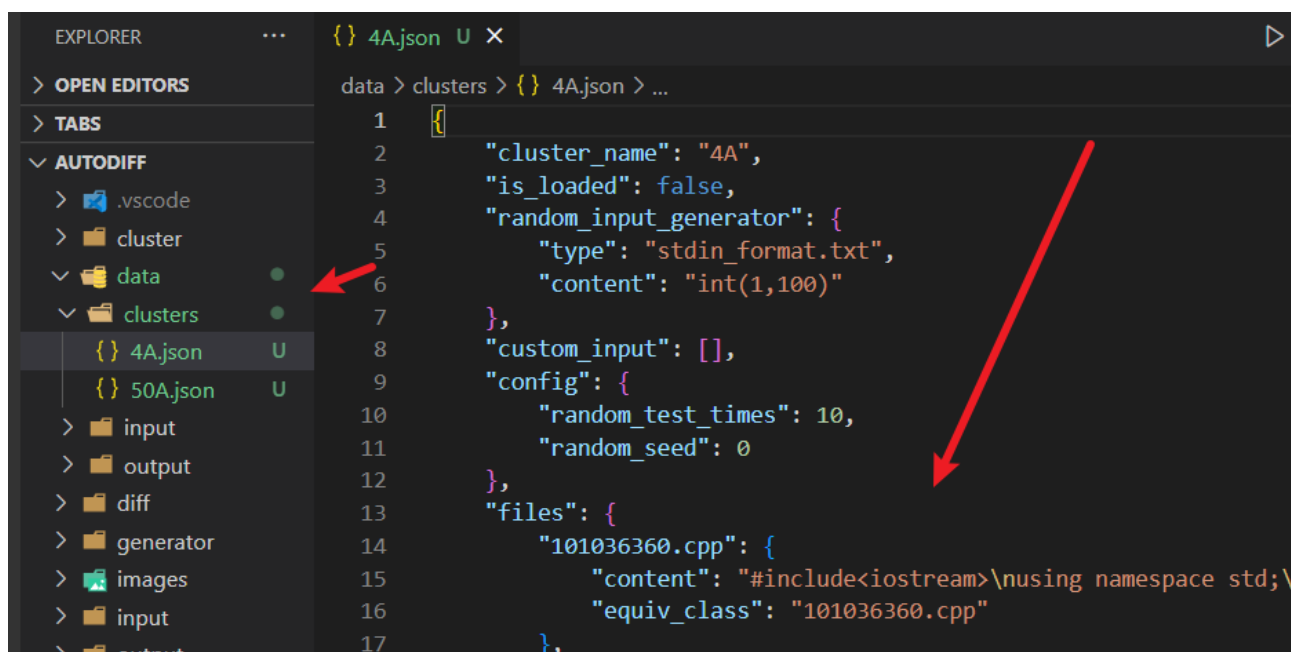
为用户提供交互选项 (例：等价，不等价，存疑等)。

5. CSV 文件下载按钮



通过左上角的「下载 CSV 文件」按钮下载当前的 CSV 文件。

6. 自动保存 Clusters 当前数据

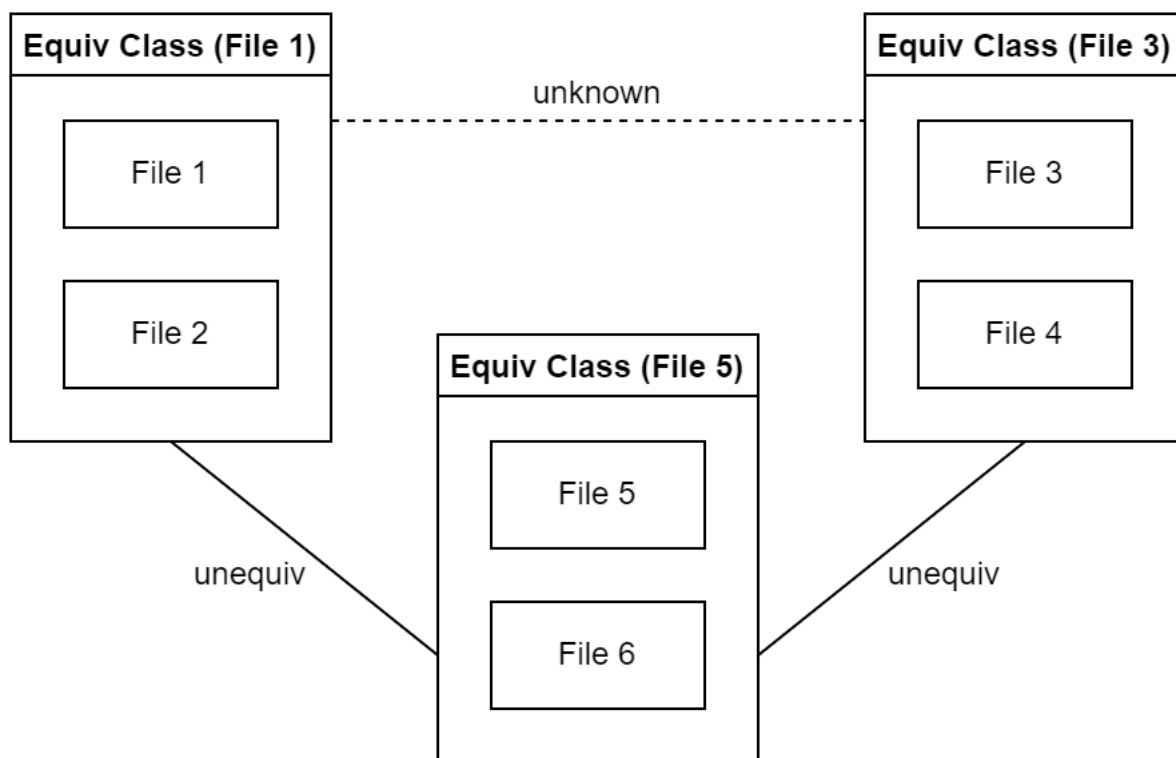


我们会自动保存当前的 Clusters 数据到 `./data/clusters` 目录下。

因此我们不需要每次都重新执行「自动随机测试比对」和「人工确认」等工作，我们会随时保存当前 Clusters 的数据，用户不用担心数据的遗失。

6. 基于并查集的等价类

我们使用 `cluster` 模块 `files` 属性下的 `equiv_class` 的一个 **并查集结构** 来标识该文件所属的 **等价类**。



如图所示：

- 在同一个等价类里的文件被认为是 **相互等价** 的，通过一个根文件来标识等价类，例如下面的那个等价类通过 `File 5` 标识。
- 在不同等价类之间，通过实线连接的等价类被认为是 **不等价** 的，即两个等价类里的文件两两匹配都不等价。
- 在不同等价类之间，通过虚线连接的等价类被认为是 **未知等价关系** 的，需要用户进一步地判断 (虚线是初始化后就存在的)。

借助这幅图所示的等价类概念，以及并查集的算法知识，我们就可以写出一个能够让用户 **手动动态更改等价关系** 的 Cluster 类。

7. 测试样例的多样性

除了 `stdin_format.txt` 外，我还添加了 `stdin_format.py` 和 `config.json` 等额外的配置文件，以及自定义测试样例，可以进行更灵活的随机样例生成和配置。

- `stdin_format.txt`：使用正则表达式实现的文本替换与随机生成。
- `stdin_format.py`：通过用户自定义 Python 程序来实现更复杂的 (如数组、矩阵) 的输入样例生成。
- `config.json`：可以配置 **生成随机样例的个数** 以及 **随机数种子** 等配置。

8. 并行多进程计算

并行地进行「**文件编译**」和「**文件比对**」，并且采用了 **多进程** 与 **进程池** 的策略来实现并行计算。

设定了进程池最大为 8 个进程，因此执行速度是原来的 8 倍。

9. 前后端分离

这次项目的前后端是分离的，因此可以通过接口的方式为更多的界面提供支持：

- `http://host:port/clusters` (GET): 获取到所有的 clusters 数据；
- `http://host:port/cluster/<cluster_name>` (GET): 获取某个 cluster 数据；
- `http://host:port/run` (POST): 为某个 cluster 执行自动并行比较；
- `http://host:port/update` (POST): 为某个 cluster 更新人工比较结果；
- `http://host:port/csv` (GET): 下载当前的 CSV 文件内容。

无论是 Web 界面，还是 Android 程序，还是 Qt 界面，只要使用了这些 API 接口，都可以兼容这个后端项目，实现「N + N」到「1 + N」的项目开发效率优化。