

软件工程实验报告

实验六: 软件分析与测试

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 12 月 6 日

目录

- 实验六: 软件分析与测试
 - 目录
 - 一、静态分析报告
 - 1.1 静态分析工具的选取及安装
 - 1.2 静态分析工具的使用说明
 - 1.3 静态分析工具的结果分析
 - 1.4 静态分析工具的代码修复
 - 二、单元测试报告
 - 2.1 测试工具: Pytest
 - 2.2 generator 模块单元测试
 - 2.3 diff 模块单元测试
 - 2.4 cluster 模块单元测试
 - 三、集成测试报告
 - 3.1 测试目的
 - 3.2 测试样例
 - 3.3 测试结果

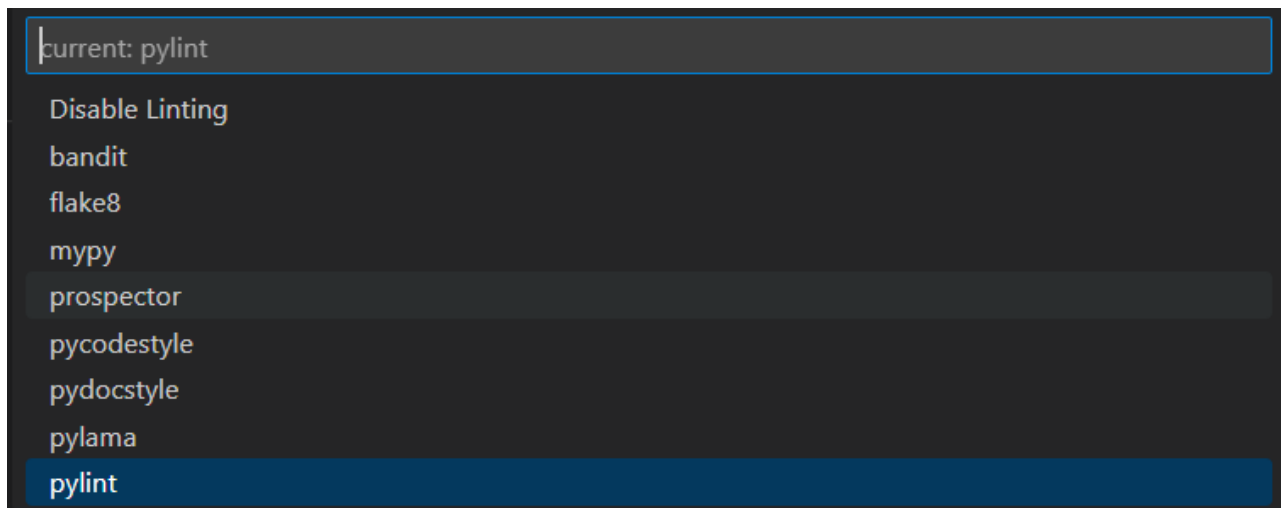
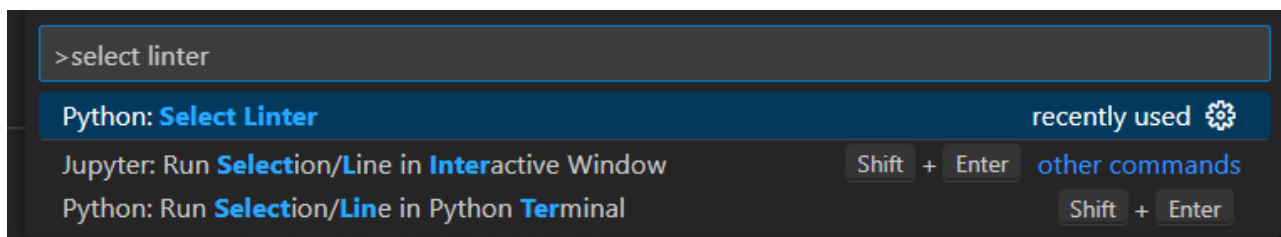
一、静态分析报告

1.1 静态分析工具的选取及安装

在实验 4 中，我使用的开发语言是 Python，因此这里我选取了 Pylint 作为我使用的静态分析工具。

在 VS Code 下 Pylint 的安装过程如下：

按下 `Ctrl + Shift + P`，输入 `select linter`，选择 `pylint`。

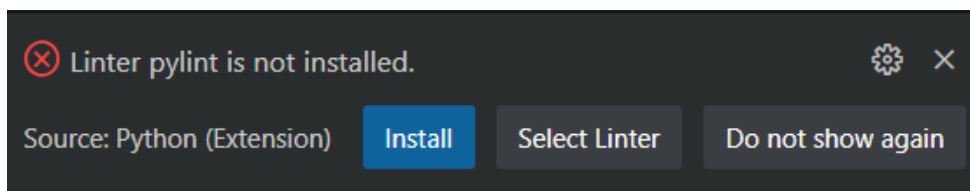


此时 `.vscode/settings.json` 文件内容如下：

```
{
  "python.analysis.typeCheckingMode": "basic",
  "python.linting.pylintEnabled": true,
  "python.linting.enabled": true
}
```

说明此时已经开启了 Pylint。

我们打开一个 Python 文件，VS Code 会提醒我们还未安装 Pylint，需要执行安装。



```
(base) PS C:\Project\autodiff> & C:/Users/OrangeX4/miniconda3/python.exe -m pip install -U pylint
Collecting pylint
  Downloading pylint-2.15.8-py3-none-any.whl (509 kB)
    

509.1/509.1 kB 939.4 kB/s eta 0:00:00


Collecting mccabe<0.8,>=0.6
  Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Collecting tomli>=1.1.0
  Using cached tomli-2.0.1-py3-none-any.whl (12 kB)
Collecting typing-extensions>=3.10.0
  Downloading typing_extensions-4.4.0-py3-none-any.whl (26 kB)
Collecting isort<6,>=4.2.5
  Downloading isort-5.10.1-py3-none-any.whl (103 kB)
    

103.4/103.4 kB 1.5 MB/s eta 0:00:00


Collecting astroid<=2.14.0-dev0,>=2.12.13
  Downloading astroid-2.12.13-py3-none-any.whl (264 kB)
    

264.8/264.8 kB 1.8 MB/s eta 0:00:00


Collecting tomlkit>=0.10.1
  Downloading tomlkit-0.11.6-py3-none-any.whl (35 kB)
Collecting platformdirs>=2.2.0
  Downloading platformdirs-2.5.4-py3-none-any.whl (14 kB)
Requirement already satisfied: colorama>=0.4.5 in c:\users\orangex4\miniconda3\lib\site-packages (from pylint) (0.4.5)
Collecting dill>=0.2
    

110.5/110.5 kB 1.3 MB/s eta 0:00:00


Collecting lazy-object-proxy>=1.4.0
  Downloading lazy_object_proxy-1.8.0-cp39-cp39-win_amd64.whl (22 kB)
Collecting wrapt<2,>=1.11
  Downloading wrapt-1.14.1-cp39-cp39-win_amd64.whl (35 kB)
Installing collected packages: wrapt, typing-extensions, tomlkit, tomli, platformdirs, mccabe, lazy-object-proxy, isort,
Successfully installed astroid-2.12.13 dill-0.3.6 isort-5.10.1 lazy-object-proxy-1.8.0 mccabe-0.7.0 platformdirs-2.5.4 p
typing-extensions-4.4.0 wrapt-1.14.1
```

如图所示，此时即为安装完成了。

我们再回到 `main.py` 文件，可以看出 Pylint 已经正常工作了，给出了几个警告。

```
main.py 1, M
main.py > main > input
1  from input import Input
2  from output import Output
3  from paracomp import Paracomp
4
5  def main(path: str):
6      # 读取输入, from_clusters 表示会读取保存的 clusters
7      print('读取输入中...')
8      input = Input(path, from_clusters=True)
9      # 进行并行比较
10     print('执行比较中...')
11     paracomp = Paracomp(path, input)
12     for cluster_name in paracomp.get_cluster_names():
13         # 如果是加载的则跳过
14         if input.clusters[cluster_name].cluster['is_loaded']:
15             continue
16         paracomp.run(cluster_name)
17     # 对结果进行保存
18     output = Output(path, input.clusters)
19     print('保存 csv 文件中...')
20     output.save_diff_list_to_csv()
21     # 同时也保存 clusters 到 clusters 文件夹
22     print('保存 clusters 文件中...')
23     output.save_clusters()
24     print('完成!')
25
26 if __name__ == "__main__":
27     main("./data")
28
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

main.py 3

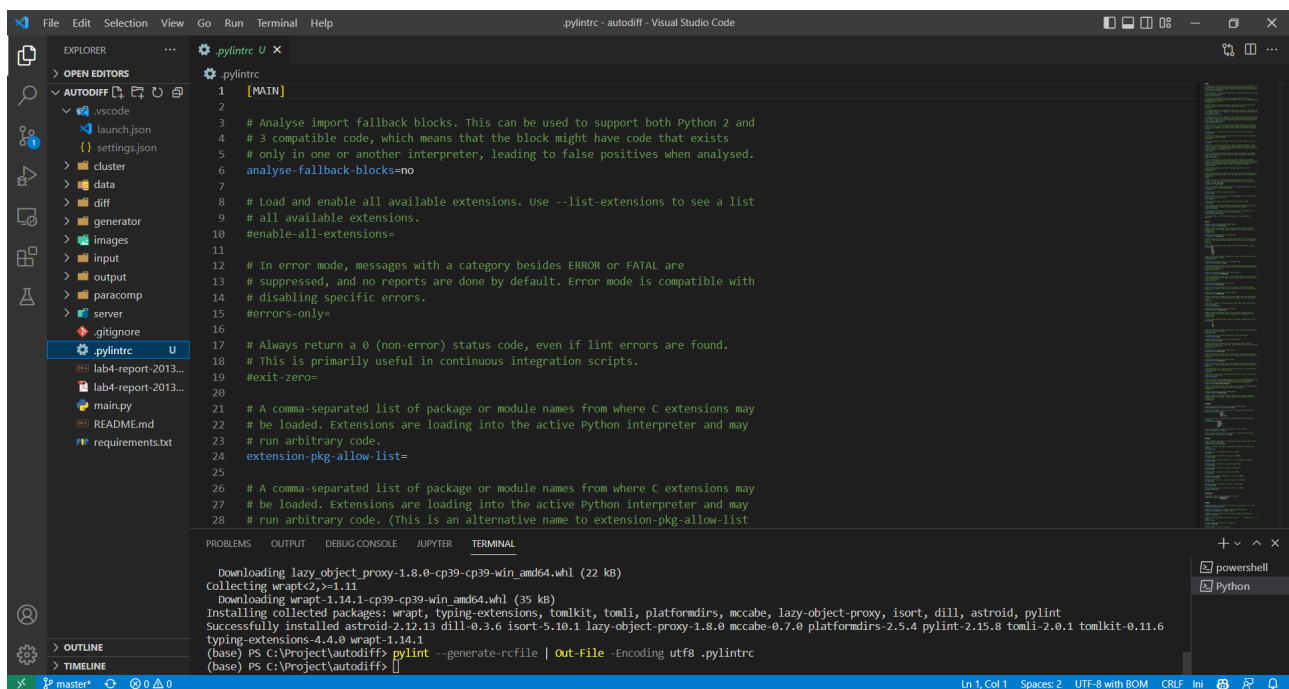
- ⚠ Redefining built-in 'input' pylint(redefined-builtin) [Ln 8, Col 5]
- ℹ Missing module docstring pylint(missing-module-docstring) [Ln 1, Col 1]
- ℹ Missing function or method docstring pylint(missing-function-docstring) [Ln 5, Col 1]

1.2 静态分析工具的使用说明

我们查询微软 VS Code 官方文档的说明, 可以知道, 我们可以控制 Pylint 的报错的警告的种类, 首先我们需要输出一个 `.pylintrc` 文件:

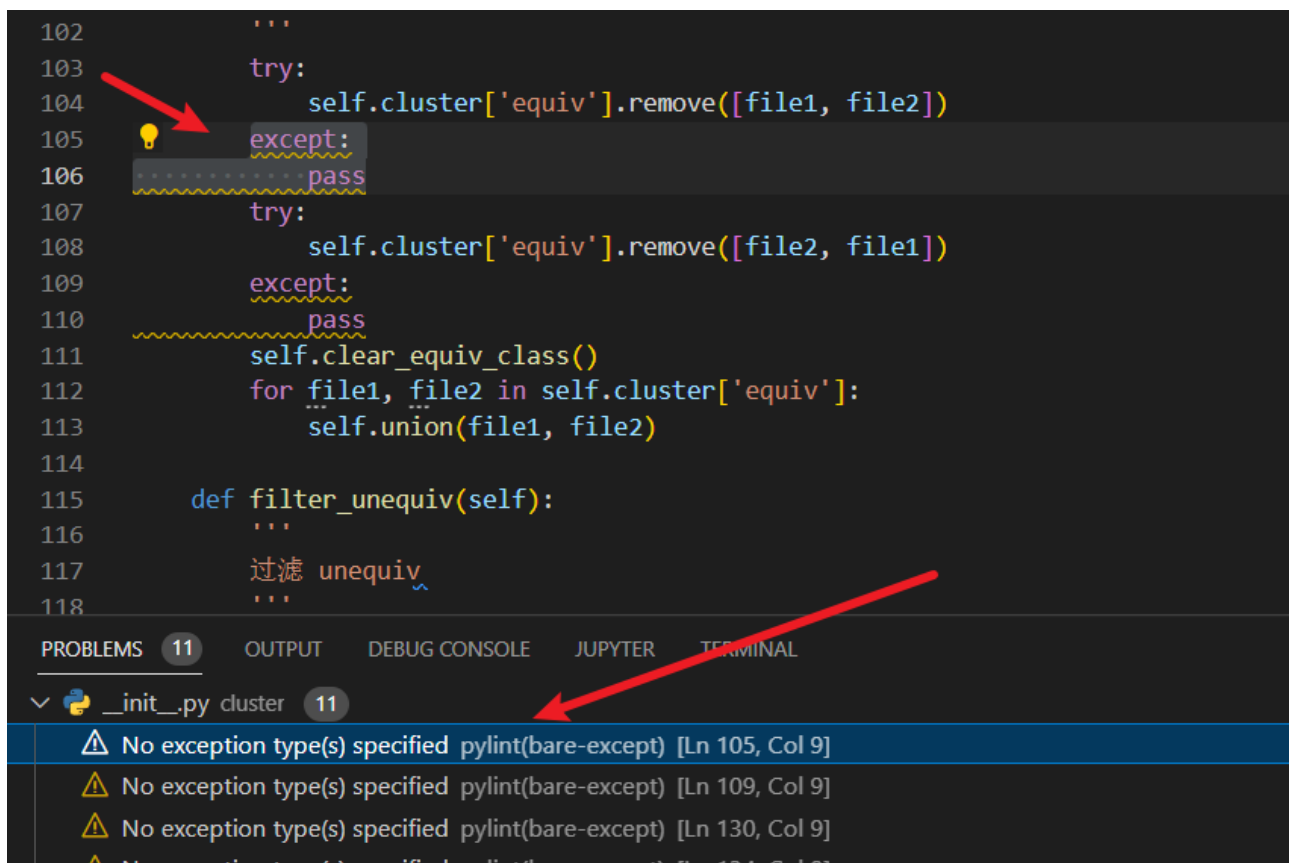
```
# Using an *nix shell or cmd on Windows
pylint --generate-rcfile > .pylintrc

# Using PowerShell
pylint --generate-rcfile | Out-File -Encoding utf8 .pylintrc
```



然后我们就可以在 `.pylintrc` 进行 Pylint 的配置。

配置完成之后，只要我们任意打开一个 Python 文件，在打开和保存的时候，VS Code 均会自动地执行 Pylint 对打开的 Python 文件进行分析。



1.3 静态分析工具的结果分析

静态分析工具的结果表明，大部分文件没有严重的代码错误，但是存在着不良的编程习惯，例如：

没有说明异常的类型：


```
104         self.cluster['equiv'].remove([file1, file2])
105     except:
106         pass
107     try:
108         self.cluster['equiv'].remove([file2, file1])
109     except:
110         pass
111     self.clear_equiv_class()
112     for file1, file2 in self.cluster['equiv']:
113         self.union(file1, file2)
114
115     def filter_unequiv(self):
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

✓ __init__.py cluster 11

- ⚠ No exception type(s) specified pylint(bare-exception) [Ln 105, Col 9]
- ⚠ No exception type(s) specified pylint(bare-exception) [Ln 109, Col 9]

重新定义了现有函数 input:

```
54
55     def execute(self, file: str, input: str) -> str:
56         cmd = format_string_with_file(self.execute_cmd, file)
57         process = Popen(cmd, stdout=PIPE, stderr=PIPE, stdin=PIPE)
58         output, err = process.communicate(input=input.encode())
59         if err:
60             # raise Exception(err.decode())
61             return err.decode()
62         return output.decode()
63
64
65     class Diff:
66         ...
67         Diff 模块，用于比较两个文件执行的差异
68         ...
```

PROBLEMS 14 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

✓ __init__.py diff 14

- ⚠ Redefining built-in 'input' pylint(redefined-builtin) [Ln 55, Col 34]
- ⚠ Redefining built-in 'input' pylint(redefined-builtin) [Ln 114, Col 44]
- ⚠ Redefining built-in 'input' pylint(redefined-builtin) [Ln 139, Col 5]

代码行数过长:

```
76         'c': {
77             'windows': Executor('{fileNoExtension}.exe', 'gcc {file} -o "{fileNoExtension}.exe"', 'del "{fileNoExtension}.exe"'),
78             'linux': Executor('./{fileNoExtension}.out', 'gcc {file} -o "{fileNoExtension}.out"', 'rm "{fileNoExtension}.out"')
79         },
80         'cpp': {
81             'windows': Executor('{fileNoExtension}.exe', 'g++ "{file}" -o "{fileNoExtension}.exe"', 'del "{fileNoExtension}.exe"'),
82             'linux': Executor('{fileNoExtension}.out', 'g++ "{file}" -o "{fileNoExtension}.out"', 'rm "{fileNoExtension}.out"')
83         },
84         'py': {
85             'windows': Executor('python "{file}"'),
86             'linux': Executor('python3 "{file}"')
87         },
88     },
89
90     def build(self, file: str) -> None:
91         ...
```

PROBLEMS 14 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL Filter (e.g. text, **/*.ts, !**/node_modules)

- Attribute name "os" doesn't conform to snake_case naming style pylint(invalid-name) [Ln 73, Col 9]
- Line too long (133/100) pylint(line-too-long) [Ln 77, Col 1]
- Line too long (131/100) pylint(line-too-long) [Ln 78, Col 1]
- Line too long (135/100) pylint(line-too-long) [Ln 81, Col 1]
- Line too long (131/100) pylint(line-too-long) [Ln 82, Col 1]

1.4 静态分析工具的代码修复

没有说明异常的类型（修复）：

```
103         try:
104             self.cluster['equiv'].remove([file1, file2])
105         except ValueError:
106             pass
107         try:
108             self.cluster['equiv'].remove([file2, file1])
109         except ValueError:
110             pass
111         self.clear_equiv_class()
112         for file1, file2 in self.cluster['equiv']:
113             self.union(file1, file2)
114
115     def filter_unequiv(self):
116         ...
117     过滤 unequiv
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

__init__.py cluster 5

重新定义了现有函数 input（修复）：

```
55     def execute(self, file: str, _input: str) -> str:
56         cmd = format_string_with_file(self.execute_cmd, file)
57         process = Popen(cmd, stdout=PIPE, stderr=PIPE, stdin=PIPE)
58         output, err = process.communicate(input=_input.encode())
59         if err:
60             # raise Exception(err.decode())
61             return err.decode()
62         return output.decode()
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

__init__.py cluster 5

代码行数过长（修复）：

```
75     self.executor_map = {}
76         'c': {
77             'Windows': Executor('{fileNoExtension}.exe',
78                 'gcc {file} -o "{fileNoExtension}.exe"',
79                 'del "{fileNoExtension}.exe"'),
80             'Linux': Executor('{fileNoExtension}.out',
81                 'gcc {file} -o "{fileNoExtension}.out"',
82                 'rm "{fileNoExtension}.out"')
83         },
84         'cpp': {
85             'Windows': Executor('{fileNoExtension}.exe',
86                 'g++ "{file}" -o "{fileNoExtension}.exe"',
87                 'del "{fileNoExtension}.exe"'),
88             'Linux': Executor('{fileNoExtension}.out',
89                 'g++ "{file}" -o "{fileNoExtension}.out"',
90                 'rm "{fileNoExtension}.out"')
91         },
92         'py': {
93             'Windows': Executor('python "{file}"'),
94             'Linux': Executor('python3 "{file}"')
95         },
96     }
97
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

__init__.py diff 7

二、单元测试报告

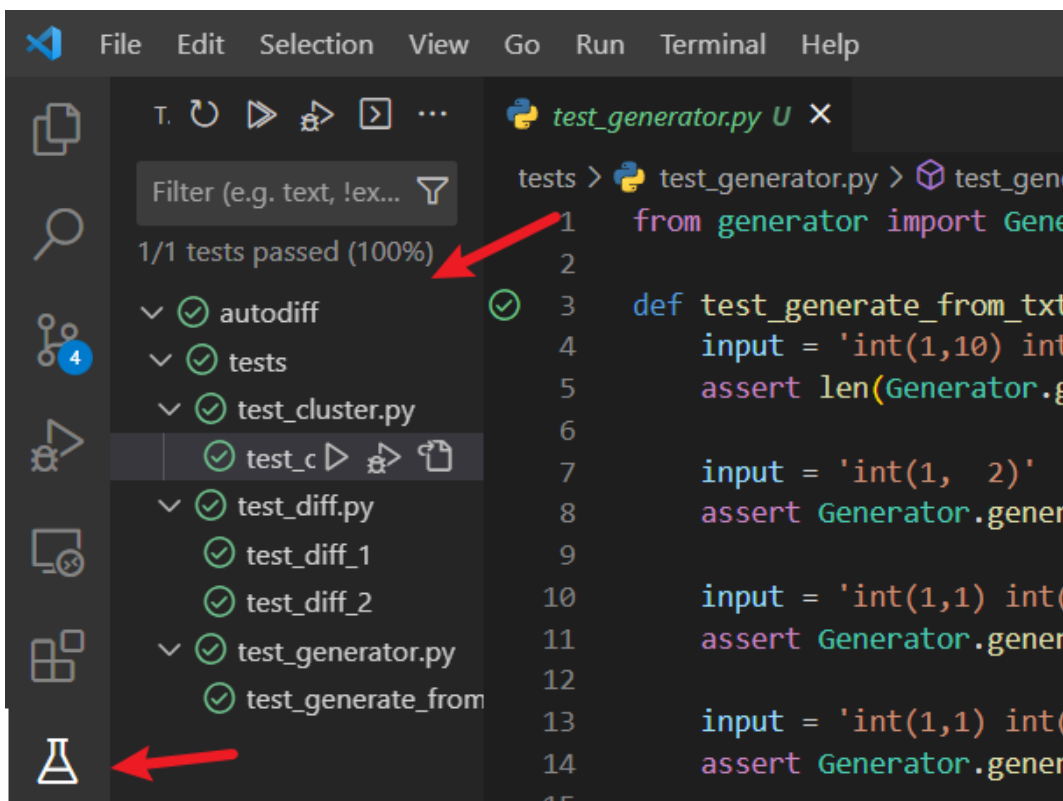
2.1 测试工具：Pytest

由于我选择的开发语言是 Python，因此这里我使用了 Pytest 作为测试工具。

通过下列命令安装 Pytest：

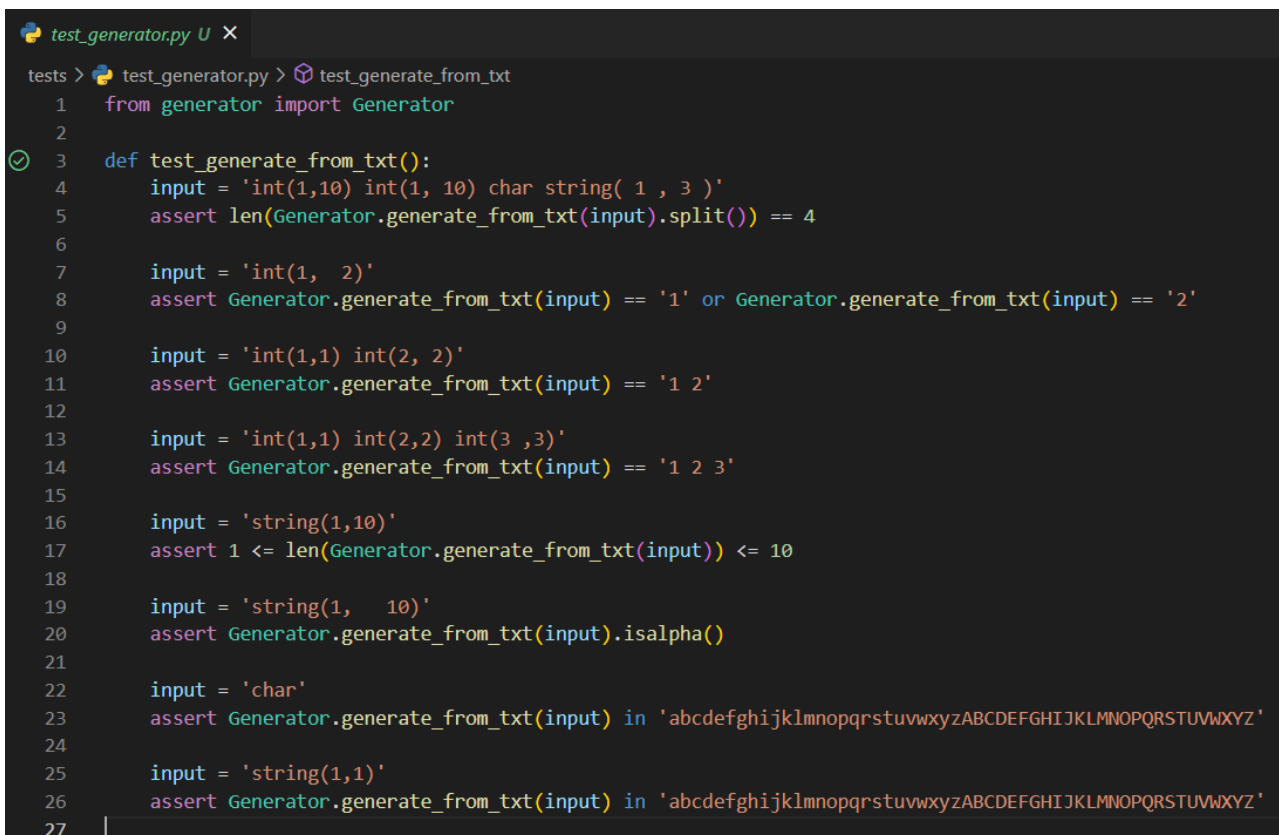
```
pip install pytest
```

然后在 VS Code 中的测试面板选择 Pytest 开始测试：



2.2 generator 模块单元测试

我们需要测试样例生成器模块，我写了 **超过 10 个测试样例** 用于测试 generator 生成器模块。



并且我使用了多种手段来测试，例如使用生成样例的分割后长度：

```
input = 'int(1,10) int(1, 10) char string( 1 , 3 )'
assert len(Generator.generate_from_txt(input).split()) == 4
```

再例如测试了 **边界条件**：

```
input = 'int(1,1) int(2,2) int(3 ,3)'\nassert Generator.generate_from_txt(input) == '1 2 3'
```

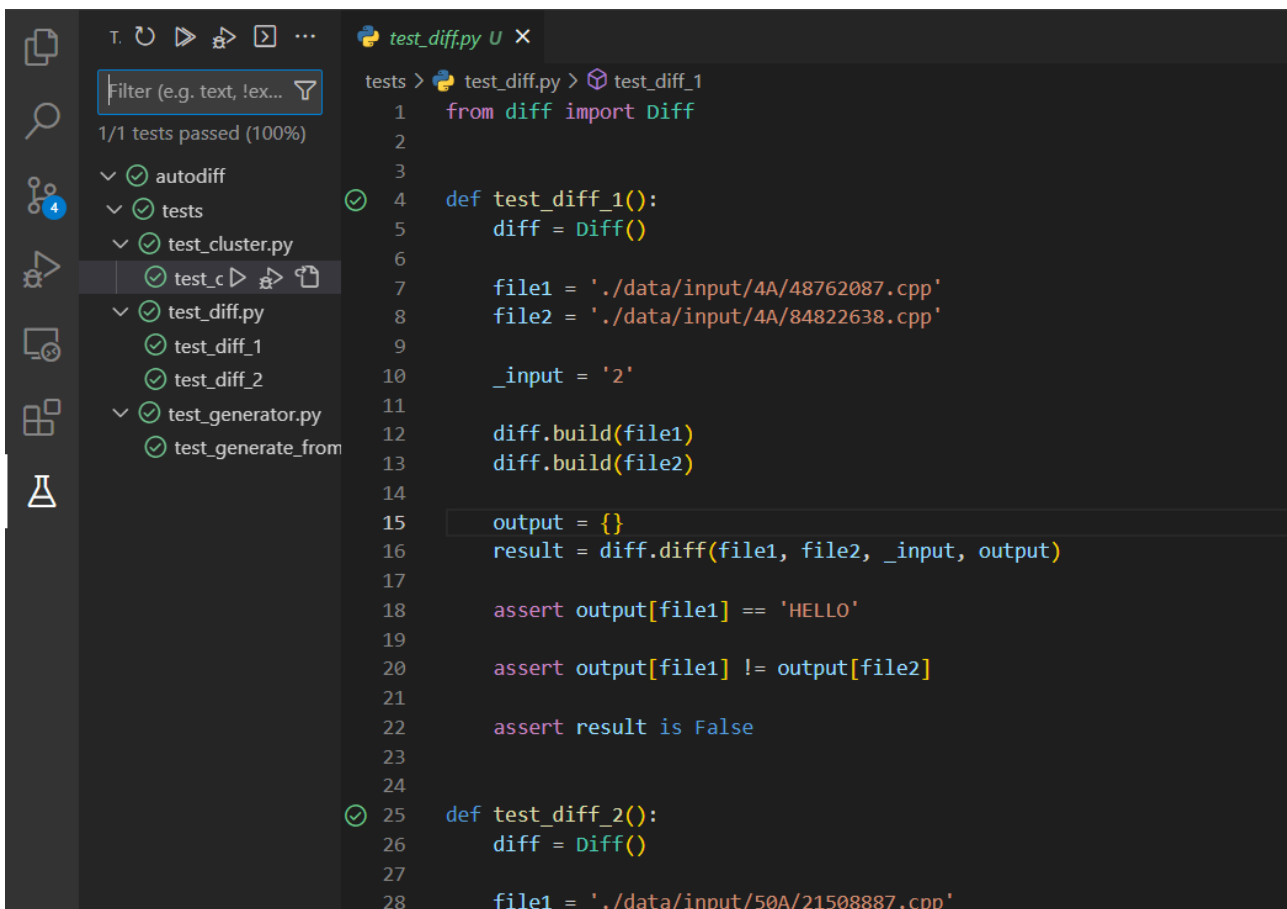
并且验证了字符的范围：

```
input = 'char'\nassert Generator.generate_from_txt(input) in \\\n    'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

以上的测试均完美通过了。

2.3 diff 模块单元测试

我们需要测试执行对比的 diff 模块，我写了 2 到 3 个用于测试 diff 比较模块的单元测试。



```
tests > test_diff.py > test_diff_1\n1  from diff import Diff\n2\n3\n4  def test_diff_1():\n5      diff = Diff()\n6\n7      file1 = './data/input/4A/48762087.cpp'\n8      file2 = './data/input/4A/84822638.cpp'\n9\n10     _input = '2'\n11\n12     diff.build(file1)\n13     diff.build(file2)\n14\n15     output = {}\n16     result = diff.diff(file1, file2, _input, output)\n17\n18     assert output[file1] == 'HELLO'\n19\n20     assert output[file1] != output[file2]\n21\n22     assert result is False\n23\n24\n25  def test_diff_2():\n26      diff = Diff()\n27\n28      file1 = './data/input/50A/21508887.cpp'
```

我们可以对执行的确定性输出进行确认，也可以对对比结果进行确认：

```
def test_diff_1():
    diff = Diff()

    file1 = './data/input/4A/48762087.cpp'
    file2 = './data/input/4A/84822638.cpp'

    _input = '2'

    diff.build(file1)
    diff.build(file2)

    output = {}
    result = diff.diff(file1, file2, _input, output)

    assert output[file1] == 'HELLO'

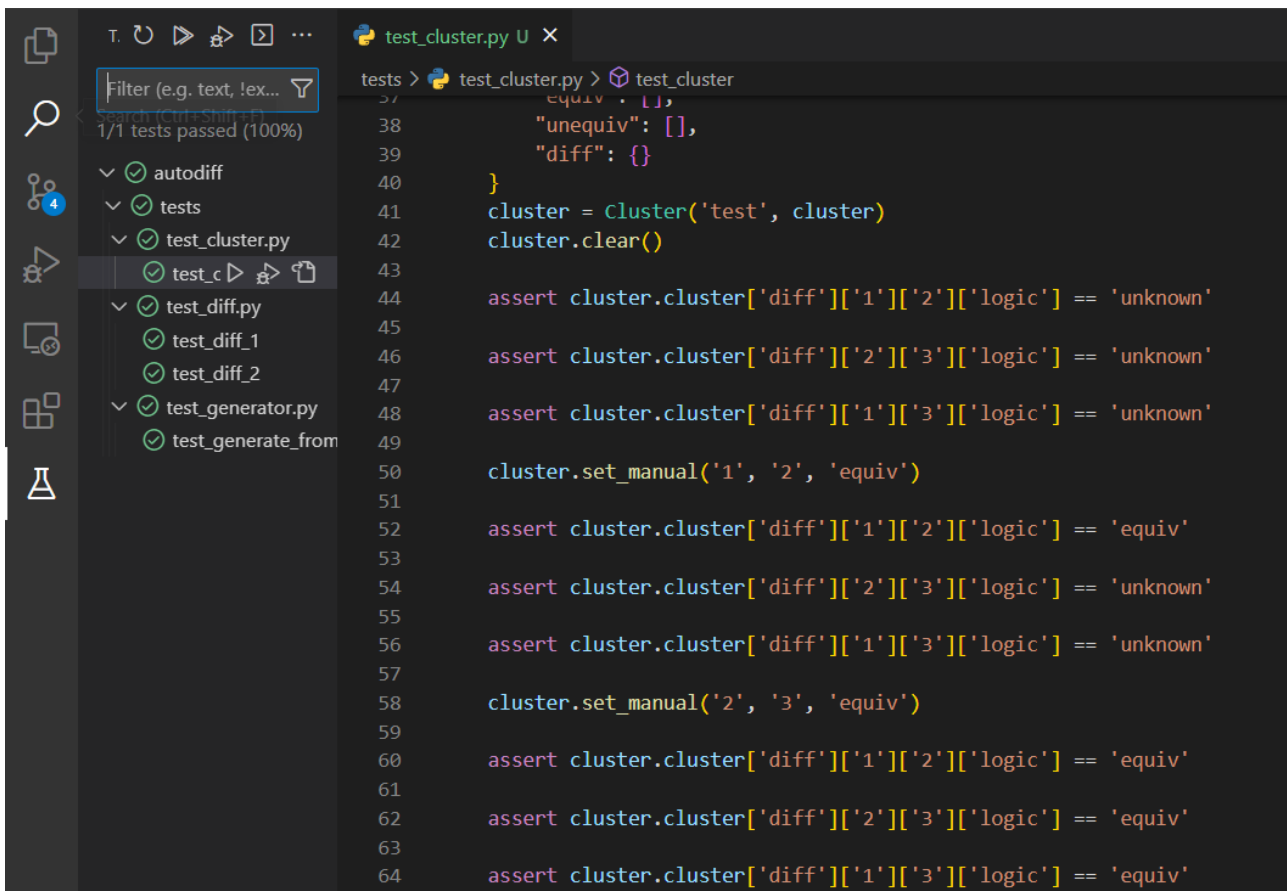
    assert output[file1] != output[file2]

    assert result is False
```

以上的测试均完美通过了。

2.4 cluster 模块单元测试

我们需要测试动态更新的 cluster 数据模块，其有一项很重要的任务，就是通过并查集的方式，根据当前已有的确认等价结果，自动确认一些其他等价结果，以减少工作量。



The screenshot shows a code editor with a test runner on the left and code on the right. The test runner shows that all tests passed (100%). The code on the right is a test function for the cluster module, which sets up a cluster and performs various assertions on its state.

```
test_cluster.py U X
tests > test_cluster.py > test_cluster
equiv: [],
38     "unequiv": [],
39     "diff": {}
40 }
41 cluster = Cluster('test', cluster)
42 cluster.clear()
43
44 assert cluster.cluster['diff']['1']['2']['logic'] == 'unknown'
45
46 assert cluster.cluster['diff']['2']['3']['logic'] == 'unknown'
47
48 assert cluster.cluster['diff']['1']['3']['logic'] == 'unknown'
49
50 cluster.set_manual('1', '2', 'equiv')
51
52 assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
53
54 assert cluster.cluster['diff']['2']['3']['logic'] == 'unknown'
55
56 assert cluster.cluster['diff']['1']['3']['logic'] == 'unknown'
57
58 cluster.set_manual('2', '3', 'equiv')
59
60 assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
61
62 assert cluster.cluster['diff']['2']['3']['logic'] == 'equiv'
63
64 assert cluster.cluster['diff']['1']['3']['logic'] == 'equiv'
```

例如这里的文件 1 和 2 等价，文件 2 和 3 等价，则自动推出 1 和 3 等价：

```

cluster = Cluster('test', cluster)
cluster.clear()

assert cluster.cluster['diff']['1']['2']['logic'] == 'unknown'
assert cluster.cluster['diff']['2']['3']['logic'] == 'unknown'
assert cluster.cluster['diff']['1']['3']['logic'] == 'unknown'

cluster.set_manual('1', '2', 'equiv')

assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
assert cluster.cluster['diff']['2']['3']['logic'] == 'unknown'
assert cluster.cluster['diff']['1']['3']['logic'] == 'unknown'

cluster.set_manual('2', '3', 'equiv')

assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
assert cluster.cluster['diff']['2']['3']['logic'] == 'equiv'
assert cluster.cluster['diff']['1']['3']['logic'] == 'equiv'

```

除此之外，还要验证对等价操作取消的逻辑：

```

cluster.set_manual('2', '3', 'unequiv')

assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
assert cluster.cluster['diff']['2']['3']['logic'] == 'unequiv'
assert cluster.cluster['diff']['1']['3']['logic'] == 'unequiv'

cluster.set_manual('2', '3', 'unknown')

assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
assert cluster.cluster['diff']['2']['3']['logic'] == 'unknown'
assert cluster.cluster['diff']['1']['3']['logic'] == 'unknown'

cluster.set_manual('3', '4', 'equiv')

assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
assert cluster.cluster['diff']['2']['3']['logic'] == 'unknown'
assert cluster.cluster['diff']['1']['3']['logic'] == 'unknown'
assert cluster.cluster['diff']['3']['4']['logic'] == 'equiv'

cluster.set_auto('2', '3', 'unequiv')
cluster.update_diff()

assert cluster.cluster['diff']['1']['2']['logic'] == 'equiv'
assert cluster.cluster['diff']['2']['3']['logic'] == 'unequiv'
assert cluster.cluster['diff']['1']['3']['logic'] == 'unequiv'
assert cluster.cluster['diff']['3']['4']['logic'] == 'equiv'

```

以上的测试均完美通过了。

三、集成测试报告

测试目的、测试对象、测试环境、测试工具、测试方法等。

以适合的形式给出各个测试的测试目的、测试用例、预期输出、实际输出等。

测试结果分析。

要求提供证明进行了集成测试的截图，如测试用例、测试结果等。

3.1 测试目的

我们进行集成测试，是为了检测各个模块之间整合后形成的统一的系统的正确性。

这里我们使用了真实的数据 4A 和 50A，集成了 input, output, diff, paracomp, clusters, genenrator 等模块的内容，进行的一个统合系统的测试。

3.2 测试样例

我们使用的测试样例为：


```

from input import Input
from output import Output
from paracomp import Paracomp

def test_integration():
    path = './data'
    # 读取输入, from_clusters 表示会读取保存的 clusters
    input = Input(path, from_clusters=True)
    # 进行并行比较
    paracomp = Paracomp(path, input)
    for cluster_name in paracomp.get_cluster_names():
        # 如果是加载的则跳过
        if input.clusters[cluster_name].cluster['is_loaded']:
            continue
        paracomp.run(cluster_name)
    # 对结果进行保存
    output = Output(path, input.clusters)

    # 确认比对结果
    assert output.clusters['4A'].cluster['diff']['101036360.cpp'] \
        ['117364748.cpp']['auto'] == 'unequiv'

    assert output.clusters['4A'].cluster['diff']['101036360.cpp'] \
        ['117364748.cpp']['logic'] == 'unequiv'

    assert output.clusters['4A'].cluster['diff']['173077807.cpp'] \
        ['84822639.cpp']['auto'] == 'equiv'

    assert output.clusters['4A'].cluster['diff']['173077807.cpp'] \
        ['84822639.cpp']['logic'] == 'unknown'

    assert output.clusters['50A'].cluster['diff']['138805414.cpp'] \
        ['21508898.cpp']['auto'] == 'unequiv'

    assert output.clusters['50A'].cluster['diff']['138805414.cpp'] \
        ['21508898.cpp']['logic'] == 'unequiv'

    assert output.clusters['50A'].cluster['diff']['138805414.cpp'] \
        ['142890373.cpp']['auto'] == 'equiv'

    assert output.clusters['50A'].cluster['diff']['138805414.cpp'] \
        ['142890373.cpp']['logic'] == 'unknown'

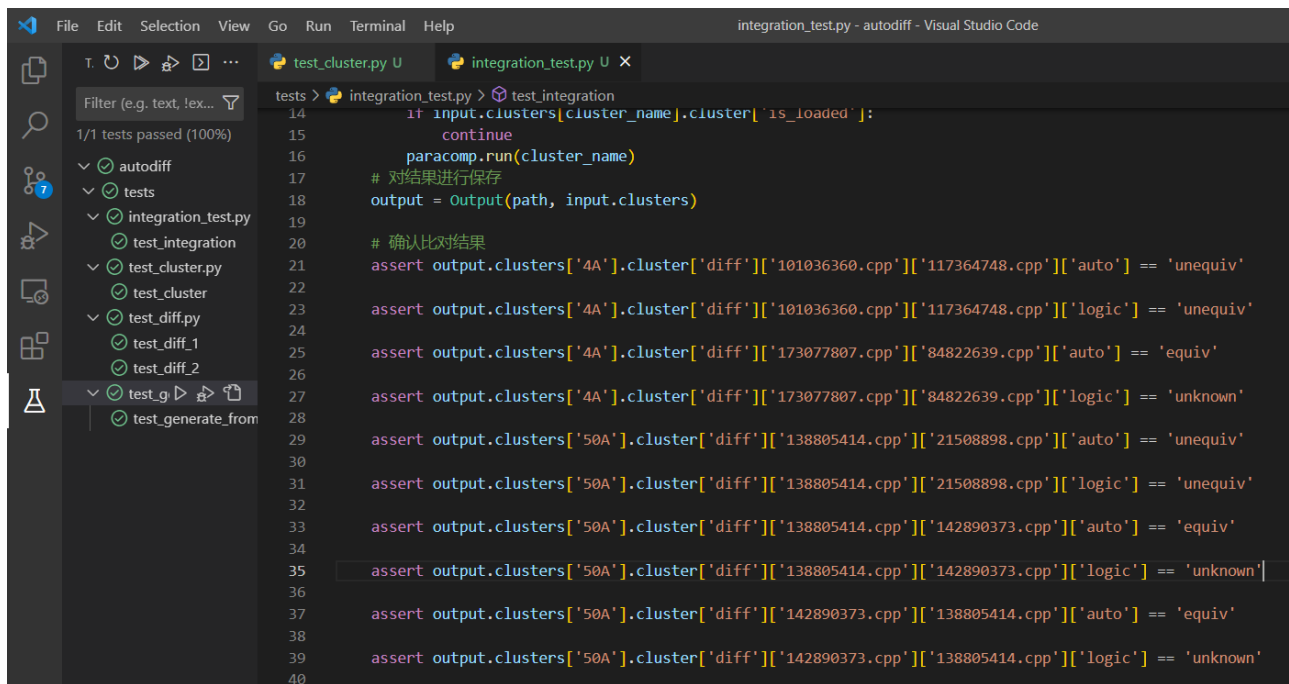
    assert output.clusters['50A'].cluster['diff']['142890373.cpp'] \
        ['138805414.cpp']['auto'] == 'equiv'

    assert output.clusters['50A'].cluster['diff']['142890373.cpp'] \
        ['138805414.cpp']['logic'] == 'unknown'

```

可以看出，我们使用 input 模块对代码文件数据进行了导入，然后通过 paracomp 模块执行了并行的计算，最后将结果导出到 output 模块中，并对结果进行判断，进行最后的集成测试。

3.3 测试结果



```
File Edit Selection View Go Run Terminal Help
integration_test.py - autodiff - Visual Studio Code

tests > integration_test.py > test_integration
14     if input.clusters[cluster_name].cluster['is_loaded']:
15         continue
16     paracomp.run(cluster_name)
17     # 对结果进行保存
18     output = Output(path, input.clusters)
19
20     # 确认比对结果
21     assert output.clusters['4A'].cluster['diff']['101036360.cpp']['117364748.cpp']['auto'] == 'unequiv'
22
23     assert output.clusters['4A'].cluster['diff']['101036360.cpp']['117364748.cpp']['logic'] == 'unequiv'
24
25     assert output.clusters['4A'].cluster['diff']['173077807.cpp']['84822639.cpp']['auto'] == 'equiv'
26
27     assert output.clusters['4A'].cluster['diff']['173077807.cpp']['84822639.cpp']['logic'] == 'unknown'
28
29     assert output.clusters['50A'].cluster['diff']['138805414.cpp']['21508898.cpp']['auto'] == 'unequiv'
30
31     assert output.clusters['50A'].cluster['diff']['138805414.cpp']['21508898.cpp']['logic'] == 'unequiv'
32
33     assert output.clusters['50A'].cluster['diff']['138805414.cpp']['142890373.cpp']['auto'] == 'equiv'
34
35     assert output.clusters['50A'].cluster['diff']['138805414.cpp']['142890373.cpp']['logic'] == 'unknown'
36
37     assert output.clusters['50A'].cluster['diff']['142890373.cpp']['138805414.cpp']['auto'] == 'equiv'
38
39     assert output.clusters['50A'].cluster['diff']['142890373.cpp']['138805414.cpp']['logic'] == 'unknown'
40
```

由图中可见，集成测试的所有测试样例均通过了。