软件工程实验报告

201840058 蒋潇鹏

一、项目概述

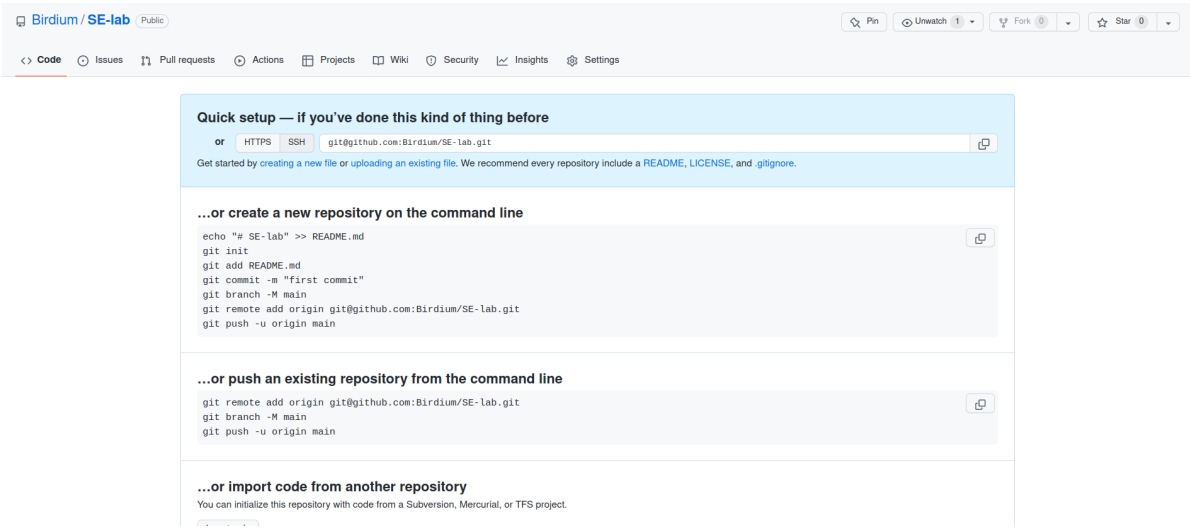
本次实验目的是完成一个等价判断工具，输入为若干个文件夹，包括一些样例程序和一个文件格式，输出为若干个csv文件，记录等价程序对和不等价程序对。

实验使用的语言为Python，考虑到如下原因：

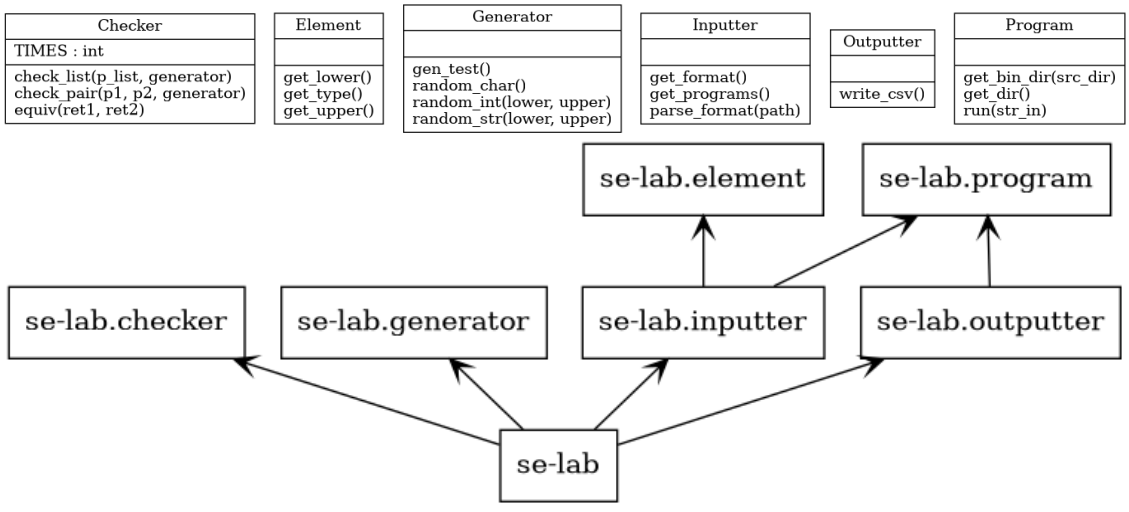
1. 程序等价判断程序行为比较贴近脚本，Python更为方便
2. 性能瓶颈主要来源于执行文件比对，python的性能缺陷会被降低
3. python具有基本的面向对象特性

二、项目创建:

1. 在github上创建空项目:



2. Clone到本地，现在SE-lab是一个包含了.git的文件夹（省去了在本地初始化的工作）
3. 使用PyCharm打开，配置环境
4. 设计基本的项目结构：我把整体项目**划分成了几个模块**：输入模块、输出模块、测试生成模块、比较模块、数据表示模块以及主模块。下图展示了最后生成的模块结构：



5. 上面每一部分的开发在不同分支上进行，并最终merge到main分支上。（没有遇到过冲突，因为不同类的实现是在不同文件里的）

   使用命令 `git checkout -b [branch]` 来建立新的分支，并用上面介绍的`git merge` 来合并。

   `git log --all --oneline --all`显示如下：

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git log --graph --oneline --all
* 0082acc (HEAD -> main, origin/main) modified .gitignore and add pic
* 769f999 debug
* 9f6a4e9 merge Element
*   466ff39 Merge branch 'element'
|\
| * 1fa569c (origin/element, element) element
* | af2f923 (origin/output, output) D D outputtter
* | 0c3d970 inputter uses Program()
* |   bc1f5a2 Merge branch 'program'
|\ \
| * | af3bf16 (origin/program, program) program
| |/
* | 75e0e33 integrate checker and generator
* |   dc52a25 Merge branch 'generator'
|\ \
| * | 317e226 (origin/generator, generator) generator
| |/
* |   9d91ee8 Merge branch 'checker'
|\ \
| * | 7cbf7a6 (origin/checker, checker) checker
| |/
* | 35f2e08 main.py
* | a8891b6 (origin/input, input) inputter
|/
| * 6d57f10 (refs/stash) WIP on checker: 1779e99 initialize
|/|
| * dbf3534 index on checker: 1779e99 initialize
|/
* 1779e99 initialize
```

三、Git使用

1. git add:

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git add .gitignore
(venv) [birdium@birdium-ms7c94 se-lab]$ git add input/*
(venv) [birdium@birdium-ms7c94 se-lab]$ git add main.py
(venv) [birdium@birdium-ms7c94 se-lab]$ git status
On branch main

No commits yet
```

2. git commit:

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git commit
[main (root-commit) 1779e99] initialize
 25 files changed, 97 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 input/4A/101036360.cpp
 create mode 100644 input/4A/117364748.cpp
 create mode 100644 input/4A/127473352.cpp
 create mode 100644 input/4A/134841308.cpp
 create mode 100644 input/4A/173077807.cpp
 create mode 100644 input/4A/48762087.cpp
 create mode 100644 input/4A/84822638.cpp
 create mode 100644 input/4A/84822639.cpp
 create mode 100644 input/4A/stdin_format.txt
 create mode 100644 input/50A/138805414.cpp
```

3. git reset (--hard 去掉所有add到暂存区的文件和工作区的文件)

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git reset --hard 1779
HEAD is now at 1779e99 initialize
```

4. 更新了一些基本文件后的 git status

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        Lab4.md
        input/
        main.py
        pic/
```

5. 某次修改(添加了一行`import`)后的git diff

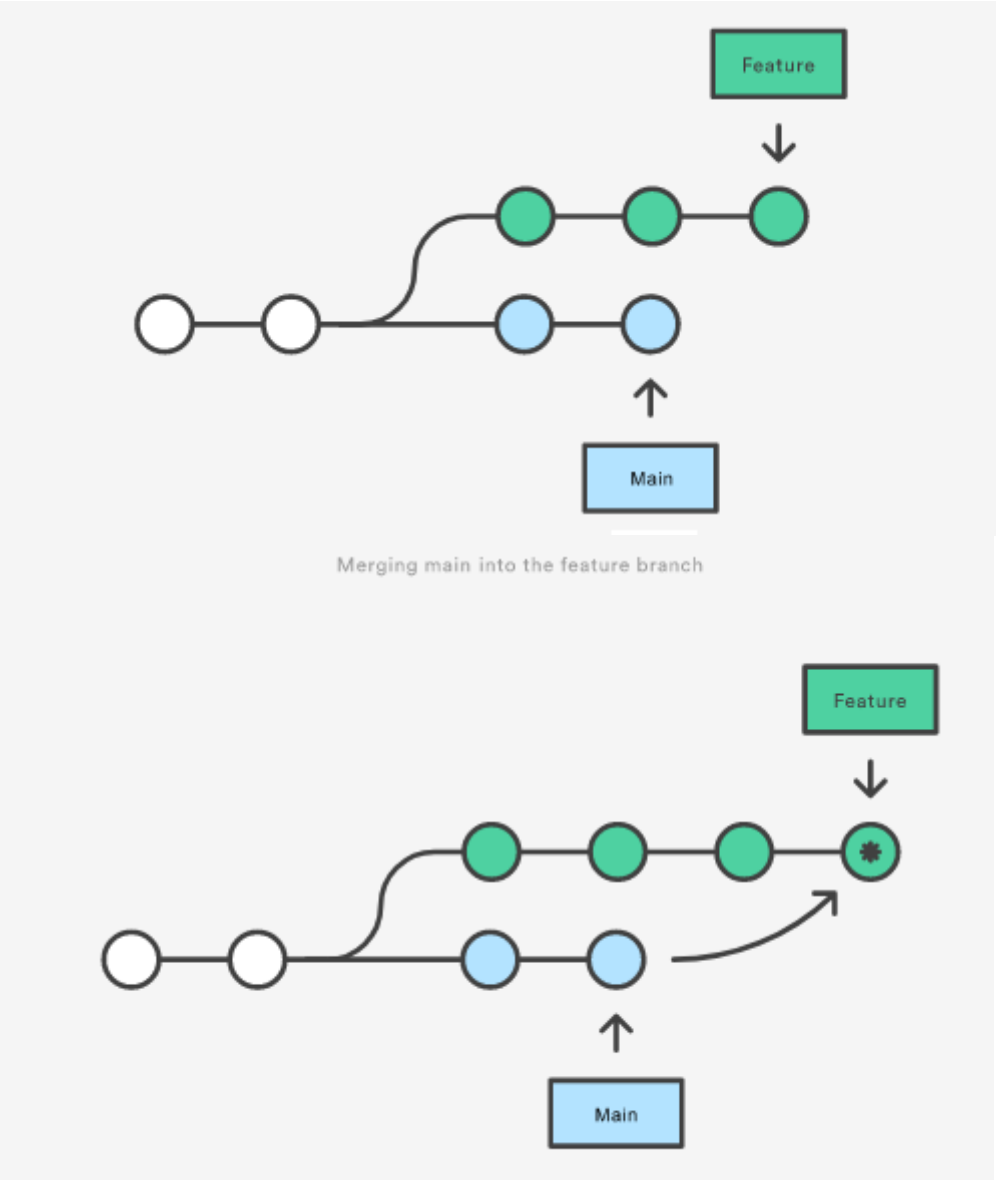
```
diff --git a/main.py b/main.py
index ffc9cd2..30d683a 100644
--- a/main.py
+++ b/main.py
@@ -1,4 +1,5 @@
 from inputter import Inputter
+from checker import Checker

 input_path = "input"
 output_path = "output"
@@ -6,8 +7,8 @@ output_path = "output"

 def main():
     inputter = Inputter(input_path)
:
```

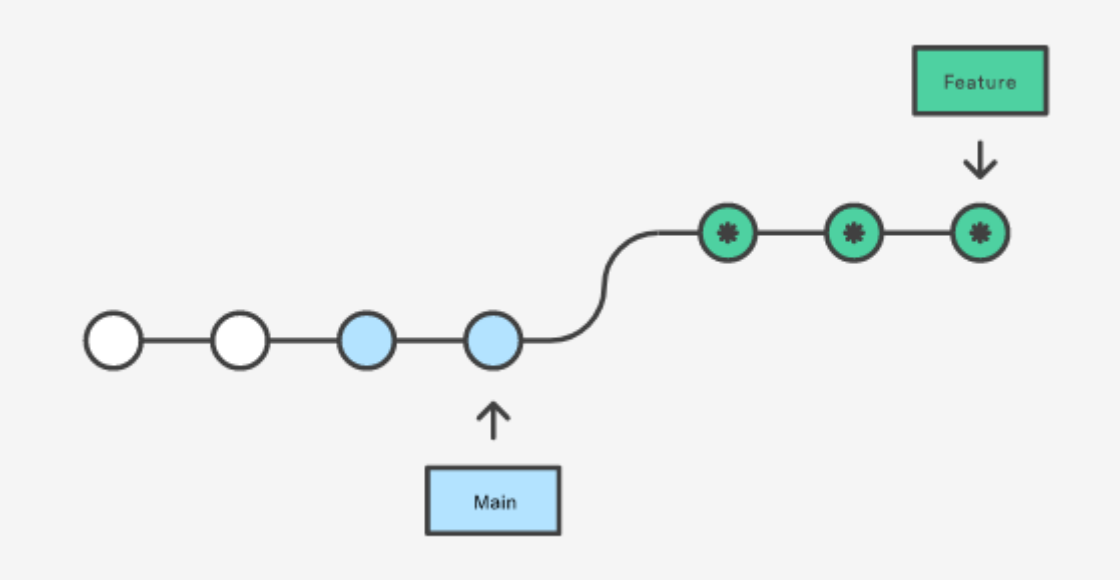
6. （扩展）git merge

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git merge input
Updating 1779e99..a8891b6
Fast-forward
 README.md    |  2 +-
 inputter.py | 39 ++++++++++++++++++++++++++++++++++++++++
 2 files changed, 40 insertions(+), 1 deletion(-)
 create mode 100644 inputter.py
(venv) [birdium@birdium-ms7c94 se-lab]$ git branch
  checker
  element
  generator
  input
* main
  program
(venv) [birdium@birdium-ms7c94 se-lab]$
```

可以看到, 在main分支下使用git merge input, 会在main中进行一个新的commit, 内容为input分支中修改的内容。除此之外的commit记录不会被改变。

而git rebase相比较之下，会将之前的commit全部删除，添加在底端，具体区别可见下面三张图(来源https://www.atlassian.com/git/tutorials/merging-vs-rebasing):





其中第一张图为操作前，第二张图为将main merge进feature, 第三张为将main rebase到feature

7. git stash
   当我写了一半，突然因为种种原因需要中断去写别的，这时候可以使用git stash将工作代码放到暂存区：

```
(venv) [birdium@birdium-ms7c94 se-lab]$ git stash
Saved working directory and index state WIP on checker: 1779e99 initialize
```

   需要回复的时候，只需要使用git stash pop 指定 index，就可以获得暂存的工作代码
8. git reset/revert
   区别非常简单，git reset是回退到之前的commit, 而git revert是用一次新的commit来回滚之前的commit

四、模块设计

1. 首先设计输入模块，我们考虑输入模块的需求：给定一个文件夹的路径，能返回一组程序和一个生成数据的格式，我的处理方式是在构造函数里传入文件夹的路径，随后对每个程序文件，调用Program类类的构造函数生成并放入列表作为`get_programs()`返回值。对`stdin_format.txt`，调用解析函数`parse_format()`进行构造，返回一个list, 元素是每行的输入类型，每行的输入类型中有若干个输入类型，被封装为Element类。

```
def parse_format(path):
    result = []
    with open(path, 'r', encoding='utf-8') as f:
        for line in f.readlines():
            line_result = []
            for item in line.split(" "): # 在识别类型时使用了python的正则表达式
库，可以直接从每一个元素里获取三元组
                line_result.extend(re.findall(r"(int)\((\d+),(\d+)\)", item))
                line_result.extend(re.findall(r"(string)\((\d+),(\d+)\)",
item))
                line_result.extend(re.findall(r"(char)", item))
            line_result = map(Element, line_result) # 使用map将Element()作用于
每一个line_result元素中
            result.append(line_result)
    return result

def __init__(self, path):
    self.__folder_path__ = path
    self.__programs__ = []
    for file in os.listdir(path):
```

```
        file_path = os.path.join(path, file)
        if file == "stdin_format.txt":
            self.__stdin_format__ = self.parse_format(file_path)
        else:
            self.__programs__.append(Program(file_path))
```

2. 然后是设计数据表示模块，需要表示的数据有输入类型Element和程序Program, 我使用之前`parse_format`中生成的类型来初始化Element.
   对于Program稍显复杂一些，我让源代码路径作为初始化参数，在构造时就开始编译，同时使用了python的`subprocess`库来实现程序的运行，返回程序运行的结果

```
class Program:
    def __init__(self, src_dir):
        self.__src_name__ = src_dir
        self.__src_dir__ = os.path.abspath(src_dir)
        self.__bin_dir__ = self.get_bin_dir(src_dir)
        args = ["g++", self.__src_dir__, "-w", "-o", self.__bin_dir__] # g++
[src] -w -o [out]
        proc = subprocess.run(args)

    def __del__(self): # 析构函数，消亡时清理生成的.out文件
        os.remove(self.__bin_dir__)

    def run(self, str_in):
        args = [self.__bin_dir__]
        return subprocess.run(args, input=str_in.encode(),
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

```
class Element:
    def __init__(self, item : tuple):
        self.__type__ = item[0]
        if item[0] == "int" or item[0] == "string":
            self.__lower__ = int(item[1])
            self.__upper__ = int(item[2])
```

3. 接下来设计数据生成模块，数据生成模块由`stdin_format`初始化，需要能生成满足要求的数据，由于前面已经对类型封装的相当好了，这里的设计就显得很简单

```
@staticmethod
def random_int(lower, upper):
    return random.randint(lower, upper + 1)

def random_char(self):
    return random.choice(self.__alphabet__)

def random_str(self, lower, upper):
    rand_str = ""
    rand_len = self.random_int(self, lower, upper)
```

```
    for i in rand_len:
        rand_str.append(self.random_char())
    return rand_str

def __init__(self, stdin_format):
    self.__alphabet__ = "qwertyuiopasdfghjklzxcvbnm"
    self.__stdin_format__ = stdin_format

def gen_test(self):
    test_str = ""
    for line in self.__stdin_format__:
        for elem in line:
            if elem.__type__ == "char":
                test_str = test_str + self.random_char()
            if elem.__type__ == "int":
                test_str = test_str + str(self.random_int(elem.get_lower(),
elem.get_upper()))
            if elem.__type__ == "string":
                test_str = test_str + self.random_str(elem.get_lower(),
elem.get_upper())
            test_str = test_str + ' '
        test_str = test_str + '\n'
    return test_str
```

4. 对于等价判断模块，我将其设计为了一个只有静态方法的类，函数`check_list(p_list, generator)`用于判断一组程序之间每一对在给定生成下运行结果是否相等，并返回两个列表，分别是等价程序对和不等价程序对，函数`check_pair(p1, p2, generator)`用于判断一对程序是否等价，`equiv(ret1, ret2)`用于判断两个程序的返回结果是否符合我们对程序等价性的定义

```
class Checker:
    TIMES = 10

    @staticmethod
    def equiv(ret1, ret2):
        if ret1.returncode == ret2.returncode:
            return ret1.returncode != 0 or ret1.stdout == ret2.stdout

    @staticmethod
    def check_pair(p1, p2, generator):
        for _ in range(Checker.TIMES):
            str_in = generator.gen_test()
            ret1 = p1.run(str_in)
            ret2 = p2.run(str_in)
            if not Checker.equiv(ret1, ret2):
                return False
        return True

    @staticmethod
    def check_list(p_list, generator):
        eq_pairs = []
        neq_pairs = []
        for i1 in range(len(p_list)):
            for i2 in range(i1):
                p1 = p_list[i1]
```

```
                p2 = p_list[i2]
                if Checker.check_pair(p1, p2, generator):
                    eq_pairs.append([p1, p2])
                else:
                    neq_pairs.append([p1, p2])
        return eq_pairs, neq_pairs
```

5. 输出模块获取等价程序对和不等价程序对，能够将符合格式的数据输出到.csv文件中。这一部分主要调用了csv库来输出到目标位置

```
class Outputter:

    def __init__(self, eq_pairs, neq_pairs, output_dir):
        self.__output_dir__ = output_dir
        self.__eq_pairs__ = [[p.get_dir() for p in pair] for pair in
eq_pairs]
        self.__neq_pairs__ = [[p.get_dir() for p in pair] for pair in
neq_pairs]

    def write_csv(self):
        eq_csv_path = os.path.join(self.__output_dir__, "equal.csv")
        neq_csv_path = os.path.join(self.__output_dir__, "inequal.csv")
        header = ['file1', 'file2']
        with open(eq_csv_path, "w", encoding='utf-8', newline='') as eq_csv:
            writer = csv.writer(eq_csv)
            writer.writerow(header)
            writer.writerows(self.__eq_pairs__)
        with open(neq_csv_path, "w", encoding='utf-8', newline='') as
neq_csv:
            writer = csv.writer(neq_csv)
            writer.writerow(header)
            writer.writerows(self.__neq_pairs__)
```

6. 在main.py中，我们需要组织整个程序的运行流程：
    1. 对每个给出的文件夹，做如下操作：
        1. 用路径调用输入模块获取生成格式和程序列表
        2. 用生成格式生成一个生成工具
        3. 将程序列表和生成工具传给判断工具，进行判断
        4. 将结果添加进列表
    2. 最后将两个列表和路径交给输出模块进行输出

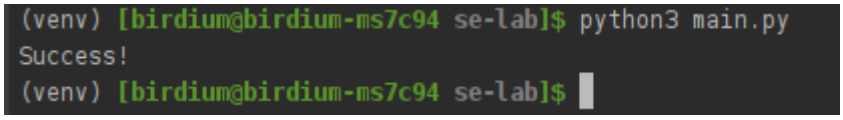
```
import os

from inputter import Inputter
from checker import Checker
from generator import Generator
from outputter import Outputter

input_path = "input"
output_path = "output"
```

```
def main():
    eq_pairs = []
    neq_pairs = []
    for folder in os.listdir(input_path):
        folder_path = os.path.join(input_path, folder)
        inputter = Inputter(folder_path)
        gene_format = inputter.get_format()
        generator = Generator(gene_format)
        p_list = inputter.get_programs()
        new_eq_pairs, new_neq_pairs = Checker.check_list(p_list, generator)
        eq_pairs.extend(new_eq_pairs)
        neq_pairs.extend(new_neq_pairs)
    outputter = Outputter(eq_pairs, neq_pairs, output_path)
    outputter.write_csv()


if __name__ == "__main__":
    main()
```

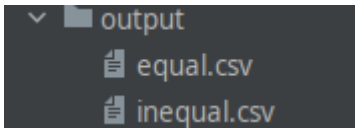
7. 完成了这些设计，并进行调试之后，将分支推送到github上，这样我们的等价判断工具就实现完成了。

五、程序demo

1. 程序运行的流程：
    1. 安装环境: git, python
    2. 运行`git clone https://github.com/Birdium/SE-lab.git`, 将代码clone到本地
    3. 向`input`文件中添加若干待测试文件夹
    4. 运行`python3 main.py`
    5. csv文件会被生成到`output.py`文件里
2. 截图展示



可以看到，成功完成判断时，程序会返回一个Success!



同时成功生成了两个目标文件

*.csv files are supported in other JetBrains IDEs

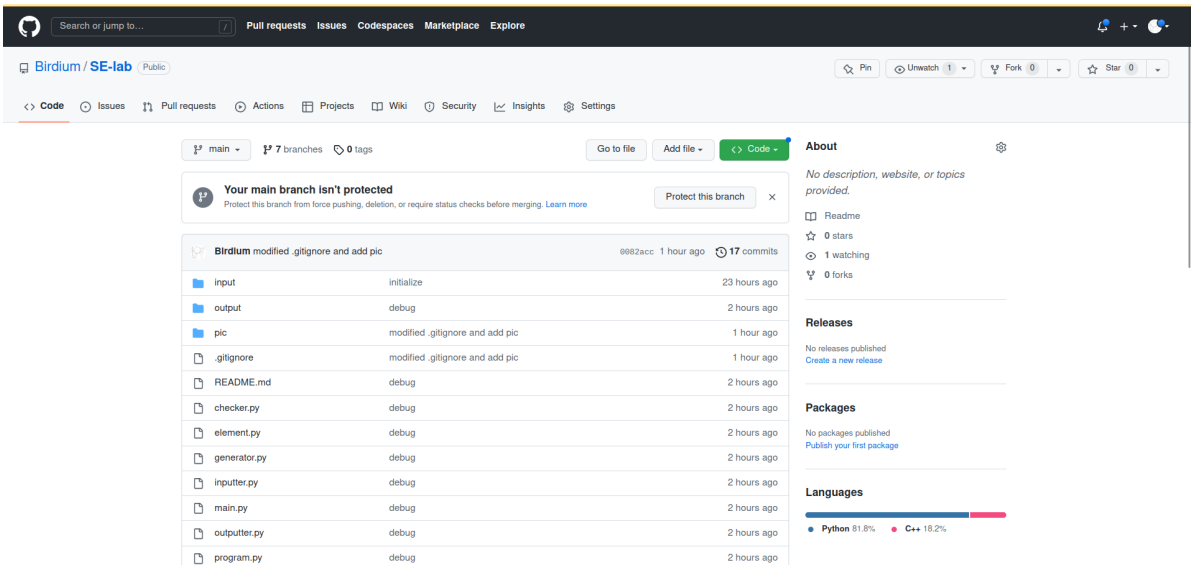
```
file1,file2
input/50A/21508898.cpp,input/50A/30534178.cpp
input/50A/164831265.cpp,input/50A/45851050.cpp
input/50A/21508887.cpp,input/50A/30534178.cpp
input/50A/21508887.cpp,input/50A/21508898.cpp
input/50A/138805414.cpp,input/50A/30534178.cpp
input/50A/138805414.cpp,input/50A/21508898.cpp
input/50A/138805414.cpp,input/50A/21508887.cpp
input/50A/142890373.cpp,input/50A/30534178.cpp
input/50A/142890373.cpp,input/50A/21508898.cpp
input/50A/142890373.cpp,input/50A/21508887.cpp
input/50A/142890373.cpp,input/50A/138805414.cpp
input/4A/173077807.cpp,input/4A/84822639.cpp
input/4A/134841308.cpp,input/4A/127473352.cpp
input/4A/84822638.cpp,input/4A/84822639.cpp
input/4A/84822638.cpp,input/4A/173077807.cpp

```

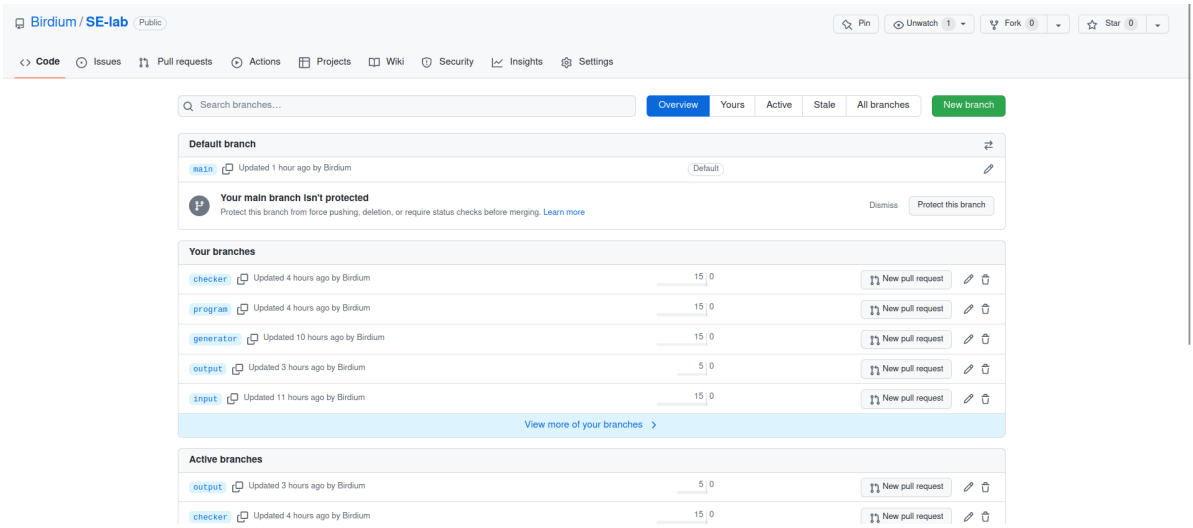
*.csv files are supported in other JetBrains IDEs

```
file1,file2
input/50A/45851050.cpp,input/50A/36641065.cpp
input/50A/30534178.cpp,input/50A/36641065.cpp
input/50A/30534178.cpp,input/50A/45851050.cpp
input/50A/21715601.cpp,input/50A/36641065.cpp
input/50A/21715601.cpp,input/50A/45851050.cpp
input/50A/21715601.cpp,input/50A/30534178.cpp
input/50A/21508898.cpp,input/50A/36641065.cpp
input/50A/21508898.cpp,input/50A/45851050.cpp
input/50A/21508898.cpp,input/50A/21715601.cpp
input/50A/164831265.cpp,input/50A/36641065.cpp
input/50A/164831265.cpp,input/50A/30534178.cpp
input/50A/164831265.cpp,input/50A/21715601.cpp
input/50A/164831265.cpp,input/50A/21508898.cpp
input/50A/21508887.cpp,input/50A/36641065.cpp
input/50A/21508887.cpp,input/50A/45851050.cpp
input/50A/21508887.cpp,input/50A/21715601.cpp
input/50A/21508887.cpp,input/50A/164831265.cpp
input/50A/138805414.cpp,input/50A/36641065.cpp
input/50A/138805414.cpp,input/50A/45851050.cpp
input/50A/138805414.cpp,input/50A/21715601.cpp
input/50A/138805414.cpp,input/50A/164831265.cpp
input/50A/142890373.cpp,input/50A/36641065.cpp
input/50A/142890373.cpp,input/50A/45851050.cpp
input/50A/142890373.cpp,input/50A/21715601.cpp
input/50A/142890373.cpp,input/50A/164831265.cpp
input/50A/29019948.cpp,input/50A/36641065.cpp
input/50A/29019948.cpp,input/50A/45851050.cpp
input/50A/29019948.cpp,input/50A/30534178.cpp
input/50A/29019948.cpp,input/50A/21715601.cpp
input/50A/29019948.cpp,input/50A/21508898.cpp
input/50A/29019948.cpp,input/50A/164831265.cpp
input/50A/29019948.cpp,input/50A/21508887.cpp
input/50A/29019948.cpp,input/50A/138805414.cpp
input/50A/29019948.cpp,input/50A/142890373.cpp
```

生成的数据，经过粗略检查发现判断大致正确

六、Github展示





可以看到，所有的分支都被推送到了remote上