Testcase-Tools

中文帮助指南

A testcase toolset repository powered by Cyaron

Preliminary preparation:

1. Run `pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple`
2. Create a folder to save all the files required to generate the problem testcase. For example, I created a folder `generator_Example\` You need to place two files in this directory: `Config.py` and `std.cpp` You need to place them like this:

   ```
   generator_example\
     |- Config.py
     |- std.cpp
   ```

   And copy the contents of `Config_template.py` and `std_template.cpp` to `Config.py` and `std.cpp`. Or you can use the command `python FastInit.py {pathname}` to initialize a problem directory quickly, for example: `python FastInit.py fastinit_example`
3. Customize Config.py You need to customize the method `Gen.generator` in `Config.py`, which should do one of the following two things:

   1. Return the string content entered by the testcase, for example:

   ```
   from cyaron import *
   class Gen:
   ```

```
    @staticmethod
    def generator(data_group: int, io: IO) -> str:
        return '114514' # Output '114514' to {data_group}. in
```

2. Write testcase data into `io` (which is recommended), for example:

```
from cyaron import *
from libs.glib import ToolSet

class Gen:
    @staticmethod
    def generator(data_group: int, io: IO) -> None:
        a = randint(ToolSet.INT_MIN, ToolSet.INT_MAX)
        b = randint(ToolSet.INT_MIN, ToolSet.INT_MAX)
        io.input_writeln(a, b)
        # Output a and b to {data_group}.in, within a line and
separated by a space
```

There are also some adjustable parameters in `Config.py`, as shown below:

```
Basic functions:
std_name = 'std.cpp'  The name of source codes
data_set = 20         Number of data groups to generate

Experimental functions(not recommended to adjust):
no_gen = False        Disable the generation of input files
genOut = False        Generate the output file in Gen.generator instead
of using {std_name}.cpp
```

Specific IO operations and tool functions and classes that Cyaron provides, please refer to Cyaron Wiki

4. Customize std.cpp In `std.cpp`, you only need to write a c++ standard program that can solve the problem **You do not need to redirect stdin or stdout** It is optional to have no `std.cpp` in the problem directory. You can set `genOut` to `True` in `Config.py` and write output in `Gen.generator`, but this is still an experimental function

1. Testcase generator

Usage: `python general.py {folder_name}`

Use Case: `python general.py generator_example`

The testcase will be automatically generated and compressed into a zip file, and the folder will be as follows:

```
generator_example\
  |- Config.py
  |- std.cpp
  |- std.exe
  |- 1.in
  |- 1.out
  |- 2.in
  |- 2.out
  | ....
  |- generator_example.zip
```

2. Comparator

Using the comparator, you can compare the output of two cpp codes with the same data

Usage: `python compare.py {Data Generator} {program1} {program2} {(Optional) Number of runs}`

Use case 1: `python compare.py generator_example compare_example/1.cpp compare_example/2.cpp` This use case is to counterplot the two programs until an error occurs or the answer is inconsistent

Use case 2: `python compare.py generator_example compare_example/1.cpp compare_example/2.cpp 100` This use case is to counterplot the two programs until an error occurs or the answer is inconsistent or the running times limit is exceeded (100 times)

Note: The program path here is relative

3. SPJ(Special Judge) Inspector

Using the spj inspector, you can produce input data and batch spj your program

Usage: `python spj_checker.py {Data Generator} {spj_program} {tested_program} {(Optional) Number of runs}`

Use case 1: `python compare.py generator_example spj_example/spj.cpp spj_example/to_test_1.cpp` This use case is to spj a program until an error occurs or the answer is incorrect

Use case 2: `python compare.py generator_example spj_example/spj.cpp spj_example/to_test_1.cpp 100` This use case is to spj a program until an error occurs or the answer is incorrect or the running times limit is exceeded (100 times)

Note: The program path here is relative