# Тестирование

Если написанный код всё же работает, то он может, к сожалению, работать не правильно, так что приходится искать когда он ломается. Попробуем перечислить способы тестирования.

1. Во-первых, самый простой способ – проверить на тестовых данных и разобрать вручную какие-то свои примеры и прогнать их. Способ конечно хороший и помогает понять, происходит ли то, что планировалось, или есть какие-то проблемы. Иногда, при составлении своих тестов, приходит понимание, что придумали не правильный алгоритм, и становится понятно, что нужно придумать другой. Так что делать свои тесты стоит, и не занимает много времени.

2. Второй важный тип тестов – крайние случаи. Это могут быть какие-то минимальные, максимальные входные данные, или просто данные со сложной структурой. В маленьких тестах можно разобраться самим, в больших же можно проверить, что программа не ломается и выводит что-то похожее на правду. Тут уже становится видно, что на больших тестах программу как-то сложно тестировать, а хотелось бы. Генерировать большие тесты можно с помощью скриптов, это мы обсудим дальше.

3. И третий тип тестов – когда тесты достаточно большие чтобы хорошо покрыть разные случаи, и при этом мы откуда-то знаем правильные ответы на них. Это называется стресс-тестированием и вам нужно: генератор рандомных тестов, точно правильное решение (но не оптимальное, например по времени), решение с багом (быстрое, но немного не правильное). Это ещё называется стресс-тестированием: генерируете случайные тесты и даёте их на вход двум решениям, потом сравниваете ответы. Эти части опять же хорошо делаются с помощью скриптов, которые как-то взаимодействуют друг с другом.

4. И плохой способ тестировать, но теоретически так можно заифать хоть все тесты. В общем-то суть простая: добавляете разные проверки на входные данные, выполнение которых приводит к разные результатам (проще всего получать WA и RE) – и отсылаете это в тестирующую систему. Так постепенно можно узнать что-то про тест и проверять его локально. Такое конечно могут признать «дестабилизацией тестирующей системы», так что применять метод не рекомендую.

Нам придётся взаимодействовать с консолью, так что какие-то маленькие её возможности придётся знать. Самый главный примитив – запуск программы, в консоли его можно делать так `prog_name < in.txt > out.txt` – это запустит бинарный файл `prog_name` с вводом из файлы `in.txt` и выводом в `out.txt`, какой-то из файлов можно не указывать (для Linux нужно здесь и далее нужно писать `./prog_name`). Если хочется запустить не бинарный файл `prog_name`, а например Python-скрипт, то нужно использовать `python script.py < in.txt > out.txt` (вместо `python` может быть придётся писать `py` или `python3` в зависимости от настроек системы).

Если считать, что файлы `gen.py`, `slow`, `fast` находятся в директории со скриптом, то всю логику тестирования можно довольно компактно написать на Python. Если мы считаем, что вывод должен совпасть целиком (со всеми пробелами и переносами строк), то можем просто сравнивать результаты работы программ.

```
1  import os
2  import sys
3
4
5  def read_file(filename):
6      with open(filename, 'r') as f:
7          data = f.readlines()
8      return data
9
```

```
10
11  iters = 100
12  call_gen = 'python gen.py > test.txt'
13  call_slow = 'slow < test.txt > out_slow.txt'
14  call_fast = 'fast < test.txt > out_fast.txt'
15
16  for i in range(1, iters + 1):
17      print('Test', i)
18      os.system(call_gen)
19      os.system(call_slow)
20      os.system(call_fast)
21      slow_out = read_file('out_slow.txt')
22      fast_out = read_file('out_fast.txt')
23      if slow_out != fast_out:
24          print("FAIL!\nInput:")
25          print(read_file('test.txt'))
26          print("Correct output:")
27          print(*slow_out)
28          print("Wrong output:")
29          print(*fast_out)
30          sys.exit()
31  print("All tests passed.")
```

Если же возможных ответов несколько, то проверку нужно писать каким-то более умным способом (чувствуете теперь, какого авторам задач всё это делать). Кстати, про авторство задач, если у вас есть архив со всеми фалами к задаче, то можно скомпилировать авторское решение и запускать на авторских тестах, используя чекер автора. Конечно, бывают ещё и интерактивные задачи, но они редки, так что не стоит ради них заморачиваться с тестированием, но при желании конечно можно разобраться самостоятельно.

И теперь приведём пример какого-нибудь простого генератора, чтобы у читателя было примерное понимание как они выглядят. Можно писать генераторы не на Python, а например на том же C++, просто нужно понимать, какой язык лучше использовать для каких целей. Сгенерируем случайное число, строку и массив:

```
1   from random import choice, randint
2
3
4   STR = 'qwertyuiopasdfghjklzxcvbnm_$'
5
6
7   def rnd_str(n: int) -> str:
8       return ''.join(choice(STR) for i in range(n))
9
10
11  def rnd():
12      n = randint(5, 10)
13      ln = randint(15, 20)
14      s = rnd_str(ln)
15      print(n)
16      print(ln, s, sep=' ')
17      for i in range(n):
18          ai = 10 ** 9
19          print(randint(-ai, ai), end=' ')
20      print()
21
22
23  rnd()
```

Конечно, на первый взгляд тестирование штука сложная, но по жизни полезная :)