Петрозаводский государственный университет
Институт математики и информационных технологий
Кафедра Информатики и математического обеспечения

Направление подготовки бакалавриата

09.03.04 Программная инженерия

Профиль направления подготовки бакалавриата

“Системное и прикладное программное обеспечение”

Отчёт о прохождении производственной практики

Выполнила:
студентка группы 22207

Юлия Александровна Тикки _____
подпись

Место прохождения практики:
Кафедра Информатики и Математического Обеспечения

Период прохождения практики:
29.05.2023-11.06.2023

Руководитель:
О.Ю. Богоявленская, к.т.н, доцент

_____
подпись

Итоговая оценка

_____
оценка

Петрозаводск — 2023

Содержание

Введение

Цель практики: тестирование и исследование производительности модуля разбора текста алгоритма случайной генерации текста с использованием цепей Маркова.

Задачи производственной практики:

- разбор метода генерации случайной генерации текста с использованием цепей Маркова;
- написание тестов для модуля разбора текста;
- анализ производительности модуля разбора текста.

1 Генерация случайного текста с использованием цепей Маркова

Задача производственной практики - реализация алгоритма генерации случайного текста - изящно решается с помощью алгоритма, описанного в книге «Практика программирования» [1]. В нём используется цепь Маркова.

Входные данные представляются в виде цепочки перекрывающихся словосочетаний. Алгоритм делит каждую фразу на две части: префикс из нескольких слов и суффикс из одного слова, следующий за префиксом. Рассматриваемый алгоритм генерирует фразы на выходе, случайным образом выбирая для определенного префикса следующий за ним суффикс. Выбор делается на основе статистики исходного текста. Для этой цели вполне подходят сочетания из трех слов, т.е. по префиксу из двух слов выбирается третье слово-суффикс:

```
поместить в w1 и w2 первые два слова текста;
вывести w1 и w2;
цикл:
   случайно выбрать w3 из набора суффиксов к префиксу w1 и w2;
   заменить w1 и w2 на w2 и w3;
   повторить цикл;
```

Программа будет считывать фрагмент текста и использовать марковский алгоритм для генерирования нового текста на основе частот появления словосочетаний фиксированной длины. Количество слов в префиксе, а в нашем примере их два, будет являться параметром задачи. Чем короче префикс, тем более бессвязным оказывается новый текст; чем этот префикс длиннее, тем больше программа склонна повторять текст практически дословно.

Желательно оставить в тексте его пунктуацию, поэтому следует считать слова «word» и «word.» различными. Таким образом улучшается качество сгенерированного текста, поскольку в нем сохраняются знаки препинания, а значит, и до некоторой степени грамматика.

2 Тестирование модуля разбора текста

2.1 Описание ожидаемой работы модуля разбора текста

Модуль разбора текста - это модуль, с которого начинается выполнение всей программы алгоритма. Программа запускается с двумя параметрами:

1. Имя файла. Является путём к файлу (абсолютным или относительным). Строковый тип данных.

2. Количество слов в префиксе. Целое положительное число.

Программа открывает файл и разбивает текст в нём на слова: каждое очередное слово передаётся в функцию, которая производит его дальнейшую обработку. Для простоты тестирования эта функция будет выводить слово в стадартный поток вывода.

При возникновении ошибок при обработке переданных аргументов или чтении файла программа должна завершаться и выводить описание ошибки в стандартный поток вывода ошибок.

2.2 Написание тестов

Так как модуль разбора текста написан на языке программирования Python, тесты было решено реализовывать с помощью среды тестирования Pytest.

Объявление фикстуры с временным файлом.

```
@pytest.fixture(scope="module")
def file_path():
    with tempfile.NamedTemporaryFile(mode="w", delete=False) as tmp_file:
        tmp_file.write("hello world\nfoo, bar\nbaz qux.")
        return tmp_file.name
```

Проверка того, что функция split_generator правильно разбивает строку на слова.

```
def test_split_generator():
    input_str = "hello   world foo  bar\nbaz\tqux"
    expected_output = ["hello", "world", "foo", "bar", "baz", "qux"]
    assert list(split_generator(input_str)) == expected_output
```

Проверка того, что программа корректно обрабатывает ошибки и код возврата программы соответствует ожидаемому.

```
def test_build_error_handling():
    with pytest.raises(SystemExit) as e:
```

```
3           sys.argv = ["program.py", "missing_file.txt", "10"]
4           main()
5       assert e.type == SystemExit
6       assert e.value.code == 4
7
8       with pytest.raises(SystemExit) as e:
9           sys.argv = ["program.py", file_path, "-1"]
10          main()
11      assert e.type == SystemExit
12      assert e.value.code == 3
13      with pytest.raises(SystemExit) as e:
14          sys.argv = ["program.py", file_path, "abc"]
15          main()
16      assert e.type == SystemExit
17      assert e.value.code == 2
```

Проверка правильного результата при работе с файлом.

```
1 def test_main_output(capsys):
2     sys.argv = ["program.py", file_path, "4"]
3     main()
4     captured = capsys.readouterr()
5     expected_output = "hello\nworld\nfoo,\nbar\nbaz\nqux.\n"
6     assert captured.out == expected_output
```

Проверка корректной обработки ошибок и вывода сообщений о них в стандартный поток вывода ошибок.

```
1 def test_main_error_handling(capsys):
2     sys.argv = ["program.py", "missing_file.txt", "10"]
3     main()
4     captured = capsys.readouterr()
5     assert captured.err.startswithОшибка(" работы с файлом:")
6     sys.argv = ["program.py", file_path, "-1"]
7     main()
8     captured = capsys.readouterr()
9     assert captured.err == Неверный" аргумент числа слов, должен быть положительным
    числом.\n"
10    sys.argv = ["program.py", file_path, "abc"]
11    main()
12    captured = capsys.readouterr()
13    assert captured.err.startswithНеверный(" аргумент числа слов")
```

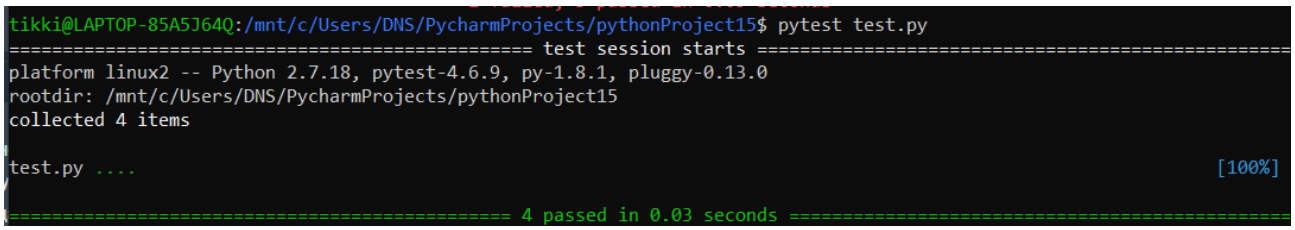

Рис. 1 – Результат запуска тестов

3 Анализ производительности модуля разбора текста

Тестирование производительности - это тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой [2].

Было принято решение запускать программу, подавая на вход файлы разного размера, и записывать результат времени выполнения программы. Для этого использовалась unix-утилита time. Эта утилита позволяет узнать время сразу в трех форматах:

- real - общее время от начала выполнения процесса и до его завершения;
- user - время, в течение которого процесс был задействован в режиме пользователя;
- sys - время, в течение которого процесс был задействован в режиме супервизора.

Таблица 1 – Результаты замеров времени выполнения реализованного алгоритма

Размер файла/Время	real	user	sys
1 килобайт	0m0.032s	0m0.016s	0m0.000s
1 мегабайт	0m0.104s	0m0.078s	0m0.016s
10 мегабайт	0m0.651s	0m0.625s	0m0.016s
100 мегабайт	0m6.259s	0m6.188s	0m0.063s
500 мегабайт	0m32.068s	0m31.703s	0m0.063s

Рис. 2 – График зависимости времени выполнения программы от размера файла

Для наглядности роста времени был построен график 2. Можно заметить, что для общего и пользовательского времени характерен экспоненциальный рост, тогда как для системного рост является очень медленным.

Можно предположить, что экспоненциальное увеличение времени выполнения программы происходит потому, что каждый дополнительный элемент входных данных приводит к значительному увеличению количества операций, которые должны быть выполнены. В целом, экспоненциальный рост времени выполнения программы может быть вызван не только экспоненциальной сложностью алгоритма, но и другими факторами, такими как выбор неэффективных структур данных и т.д.

Заключение

В процессе производственной практики был изучен метод случайной генерации текста с использованием цепей Маркова, написаны тесты для модуля разбора текста, исследована и проанализирована его производительность.

Список литературы

1. Керниган, Брайан У., Пайк, Роб. Практика программирования. : Пер. с англ. - М. : ООО «И.Д. Вильямс», -288 с.

2. Тестирование производительности - Википедия [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Тестирование_производительности.

3. Pytest documentation [Электронный ресурс] URL: https://docs.pytest.org/en/latest/.