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”

Отчет о прохождении производственной практики

Выполнил:
студент группы 22207

Илья Игоревич Иванов _____
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: разработать оптимизированный модуль разбора текста для алгоритма генерации текста на основе цепей Маркова.

Задачи производственной практики:

- выбрать способы оптимизации модуля разбора текста;
- реализовать модуль разбора текста.

В данной производственной практике решается задача генерации текста на основе алгоритма, построенного на цепях Маркова. В работе рассматривается модуль разбора текста.

Алгоритм, использующий цепи Маркова, 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 - в данном случае статистики исходного текста. Для этой цели вполне подходят сочетания из трех слов, т.е. по префиксу из двух слов выбирается третье слово-суффикс

В результате работе будет написан оптимальный код на Python для разбора файла на слова.

1 Способы оптимизации

1.1 Последовательное чтение файла

Оптимальный способ последовательного чтения файла – это последовательное чтение строк этого файла с помощью конструкции `with open(...) as file`.

Конструкция `with open(...) as file` является более оптимизированным способом чтения содержимого файла в Python по сравнению с методами `read` и `readlines` поскольку считывает файл построчно, а не целиком. Этот подход также гораздо более эффективен с точки зрения использования памяти, поскольку не требует от системы выделения памяти для всего файла сразу.

Более того, при чтении большого файла загрузка всего содержимого в память может привести к серьёзным проблемам с производительностью и вызвать аварийное завершение работы программы в случае, если у программы закончится свободная память.

Оператор `with` также гарантирует, что файл будет правильно закрыт после выполнения блока кода. Когда как при использовании методов `read`, `readlines` ответственность за правильное закрытие файла после чтения лежит на программисте. В противном случае ресурсы могут остаться открытыми, что может вызвать проблемы с производительностью или аварийное завершение программы.

1.2 Последовательное разбиение строки

Оптимальный способ последовательного разбиения строки – это последовательная обработка строки с использованием генератора.

Встроенная функция `split` разбивает строку на список подстрок и возвращает этот список в качестве результата. Это означает, что весь результат должен храниться в памяти, что может быть проблемой для больших строк.

Использование генератора же позволяет последовательно возвращать очередное слово. Это позволяет не выделять список строк, а обрабатывать слова одно за другим, что должно улучшить производительность и значительно уменьшить использование памяти.

Однако стоит отметить, что функция `split` внутреннее реализована на языке Си, что даёт более высокую производительность, чем генератор написанный на Python. Тем не менее даже в таком случае генератор значительно уменьшит использование памяти.

Результат оптимизаций также сильно зависит от содержимого исходного файла: количества строк и длины строк.

2 Число слов – параметр

Количество слов в префиксе 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. Для англоязычного текста выбор третьего слова по двум предыдущим является хорошим компромиссом. Это позволяет сохранить колорит исходного текста, добавив к нему новые причудливые штрихи.

В программе будет считываться и проверяться число слов в глобальную переменную для дальнейшего использования с целью поиска префиксов.

3 Реализация алгоритма

Итоговый модуль на Python состоит из следующих частей:

```
def main():
    global word_count

    if len(sys.argv) != 3:
        print_error(f"Usage: {sys.argv[0]} <name> <count>.")
        exit(1)

    path = sys.argv[1]
    word_count = 0

    try:
        word_count = int(sys.argv[2])
    except Exception as exception:
        print_error(f"Count not a number: {exception}")
        exit(2)

    if word_count <= 0:
        print_error(f"Count negative.")
        exit(3)

    try:
```

```
        with open(path, encoding="ascii") as file:
            build(file)
    except Exception as exception:
        print_error(f"Error {exception}")
        exit(4)
```

Листинг 1 – Функция main

Данный фрагмент отвечает за начало работы программы. Считываются и проверяются два аргумента командной строки: путь к файлу и число слов в префиксе. При успешном открытии файла вызывается функция `build`, отвечающая за разбор текста.

```
def split_generator(s, sep=string.whitespace):
    word = []
    for c in s:
        if c in sep:
            if word:
                yield "".join(word)
                word = []
        else:
            word.append(c)
    if word:
        yield "".join(word)


def build(file):
    for line in file:
        for word in split_generator(line):
            add(word)
```

Листинг 2 – Функция build и split generator

Данный фрагмент отвечает за разбиение текста. При вызове функции `build` начинается последовательное чтение файла, о котором говорилось ранее. Для очередной строки вызывается функция-генератор `split_generator`, о которой также говорилось ранее. Функция поочередно возвращает очередное слово. В результате для очередного слова вызывается функция `add`, которая являются частью модуля поиска префиксов.

Заключение

В процессе прохождения производственной практики мною были рассмотрены способы оптимизации алгоритма разбора текста а также реализован алгоритм разбора текста с выбранными способами оптимизации.

Список литературы

1. Работа с файлами [Электронный ресурс]. URL: http://cs.mipt.ru/advanced_python/lessons/lab03.html.

2. Генераторы в Python. Оператор yield. Генераторные выражения [Электронный ресурс]. URL: https://younglinux.info/oopython/generator.

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