Петрозаводский государственный университет
Институт математики и информационных технологий
Кафедра прикладной математики и кибернетики

Направление подготовки бакалавриата

09.03.04 Программная инженерия

Отчет о прохождении производственной практики

# Отчет

Выполнил:
студент 2 курса группы 22207
У. Е. Укпере

Руководитель:
Богоявленская О. Ю.

_____

*подпись*

Итоговая оценка:

_____

*оценка*

Петрозаводск 2023

# Содержание

# Введение

Реализация алгоритма цепей Маркова для генерации текста является мощным подходом, который использует статистические свойства и зависимости между словами для создания связного и контекстно соответствующего текста на основе заданного корпуса входного текста. В данном отчете представлено подробное объяснение каждой части программы с использованием предоставленных фрагментов кода. Будут обсуждаться цель программы, реализация алгоритма цепей Маркова, скорость работы программы и возможность ее реализации на других языках программирования.

# Цель программы

Цель программы состоит в генерации текста, который максимально соответствует стилю и структуре входного корпуса текста. Анализируя статистические закономерности и зависимости между словами в тексте, программа стремится создавать реалистичный и контекстно подходящий текст. Сгенерированный текст может использоваться в различных приложениях, включая создание контента, ответы чат-ботов и задачи обработки естественного языка. Для создания модели, улавливающей отношения между словами во входном корпусе, программа использует алгоритм цепей Маркова.

# 1 Реализация алгоритма цепей Маркова

Реализация алгоритма цепей Маркова:

Предоставленные фрагменты кода реализуют алгоритм цепей Маркова для генерации текста. Давайте рассмотрим каждую часть кода, чтобы понять его реализацию:

**Предобработка текста (preprocess_text)**: Функция preprocess_text отвечает за очистку и предобработку входного текста. Она преобразует текст в нижний регистр с использованием функции lower() и удаляет пунктуацию с помощью функции translate(). Кроме того, она заменяет символы новой строки пробелами с использованием функции replace() и удаляет лишние пробелы с помощью функций join() и split(). Эти шаги предварительной обработки обеспечивают согласованность и готовят текст для дальнейшего анализа.

**Построение статистики префиксов (build_prefix_stats)**: Функция build_prefix_stats создает словарь статистики префиксов. Она разбивает предобработанный текст на слова с помощью функции split() и выполняет итерацию по словам. Для каждого префикса длиной prefix x_length создается кортеж слов в качестве префикса и следующее слово в качестве суффикса. Если префикс отсутствует в словаре prefix_stats, создается новая запись с пустым списком в качестве значения. Затем суффикс добавляется в список суффиксов, соответствующих данному префиксу. Этот процесс позволяет учитывать частоту последовательностей слов и их возможных следующих слов.

## Генерация текста (generate_text)

Функция generate_text генерирует текст с использованием словаря статистики префиксов. В качестве входных данных она принимает начальный префикс, желаемую длину и статистику префиксов. Функция инициализирует текущий префикс с помощью начального префикса, разбитого на слова с использованием функции split(). Она также инициализирует сгенерированный текст как список, содержащий слова в текущем префиксе. Затем она выполняет итерацию для генерации оставшейся длины текста. Если текущий префикс отсутствует в словаре статистики префиксов или не имеет доступных суффиксов, процесс останавливается. В противном случае выбирается случайный суффикс с помощью функции random.choice() из списка суффиксов, соответствующих текущему префиксу. Суффикс добавляется в сгенерированный текст, а текущий префикс обновляется последними словами в сгенерированном тексте с использованием среза. Этот процесс продолжается до достижения желаемой длины текста или отсутствия подходящих суффиксов.

# 2 Скорость работы программы

Скорость работы программы зависит от размера входного корпуса текста и желаемой длины сгенерированного текста. Шаг предварительной обработки имеет линейную сложность относительно длины текста. Он включает итерацию по символам или словам в тексте и выполнение операций, таких как преобразование в нижний регистр, удаление пунктуации и манипуляции с пробелами. Построение статистики префиксов включает итерацию по словам в предварительно обработанном тексте и соответствующее обновление словаря prefix_stats. Этот шаг имеет сложность O(n), где n - количество слов в тексте. Процесс генерации текста зависит от желаемой длины текста и сложности статистики префиксов. Он включает случайный выбор и добавление слов, что имеет постоянную сложность. В целом программа работает эффективно для среднего размера входных текстов и разумной длины сгенерированного текста. Однако очень большие входные корпусы или чрезвычайно длинные сгенерированные тексты могут увеличить время выполнения.

## 2.1 тестирование

```
import random
import string
import time
from text_generator import generate_text

def test_generate_text():
    prefix_stats = {}  # Provide the prefix_stats dictionary you want to test
    initial_prefix = "because"  # Provide the initial_prefix you want to test
    text_length = 1000  # Provide the
    desired length of the generated text for testing

    start_time = time.time()
    generated_text = generate_text(prefix_stats, initial_prefix, text_length)
    end_time = time.time()

    execution_time = end_time - start_time
    print(f"Generated␣text:\n{generated_text}")
    print(f"Execution␣time:␣{execution_time}␣seconds")

test_generate_text()
```

```
MacBook-Air-user:practise user$ python3 test.py
Generated text:
because
Execution time: 1.1920928955078125e-05 seconds
MacBook-Air-user:practise user$
```

# 3 Реализация на других языках программирования

Реализация алгоритма цепей Маркова для генерации текста не ограничена конкретным языком программирования. Хотя предоставленный фрагмент кода демонстрирует реализацию на Python, программу можно реализовать на других языках программирования, чтобы достичь той же функциональности. Хотя логика остается прежней, синтаксис и специфические возможности языка могут отличаться. Вот несколько примеров реализации программы на различных языках:

**JavaScript:** JavaScript предоставляет мощные функции обработки строк и структуры данных, подходящие для генерации текста веб-приложений. Программу можно реализовать с использованием встроенных методов JavaScript для обработки строк и массивов.

**Java:** Java предлагает набор надежных инструментов для обработки текста. Программу можно реализовать с использованием функций обработки строк и структур данных Java, таких как HashMap или LinkedHashMap для хранения статистики префиксов.

**C++:** C++ предоставляет высокопроизводительные возможности для генерации текста. Программу можно реализовать с использованием функций обработки строк и структур данных C++, таких как std::unordered_map или std::map.

**Ruby:** Ruby - это динамический, рефлексивный и объектноориентированный язык, известный своей простотой. Программу можно реализовать с использованием методов обработки строк Ruby и хеш-структур данных.

Каждый язык программирования имеет свои собственные преимущества и особенности, и выбор языка зависит от конкретных требований и предпочтений разработчика. Однако концепции и логика алгоритма цепей Маркова остаются неизменными, независимо от выбранного языка программирования.

## 3.1 Реализация генерации текста с использованием префиксной статистики в python

```
import random
import string

def preprocess_text(text):
    % Clean and preprocess the text
    cleaned_text = text.lower()  % Convert text to lowercase
    cleaned_text =
    cleaned_text.translate(str.maketrans("", "", string.punctuation))
    % Remove punctuation
    cleaned_text = cleaned_text.replace("\n", "␣")  % Remove newline characters
    cleaned_text = "␣".join(cleaned_text.split())  % Remove extra whitespaces
    return cleaned_text

def build_prefix_stats(text, prefix_length):
    prefix_stats = {}
    words = text.split()

    for i in range(len(words) - prefix_length):
        prefix = tuple(words[i:i + prefix_length])
        suffix = words[i + prefix_length]
        if prefix not in prefix_stats:
            prefix_stats[prefix] = []
        prefix_stats[prefix].append(suffix)

    return prefix_stats

def generate_text(prefix_stats, initial_prefix, length):
    current_prefix = tuple(initial_prefix.split())
    generated_text = list(current_prefix)

    for _ in range(length - len(generated_text)):
        if current_prefix not in prefix_stats
        len(prefix_stats[current_prefix]) == 0:
            break
    % Handle cases where the prefix
    is not in the statistics or has no available suffixes
```

```
        next_word = random.choice(prefix_stats[current_prefix])
        generated_text.append(next_word)
        current_prefix = tuple(generated_text[-len(current_prefix):])

    return "␣".join(generated_text)

% Step 3: Setup and Usage Instructions

% Set the desired prefix length
prefix_length = 1

% Set an initial prefix to start the text generation
initial_prefix = "because"

% Set the desired length of the generated text
text_length = 1000

with open("source.txt") as f:
    corpus = f.read()

% Preprocess the text
cleaned_text = preprocess_text(corpus)

% Build the prefix statistics
prefix_stats = build_prefix_stats(cleaned_text, prefix_length)

% Generate the text
initial_prefix = initial_prefix.lower()
generated_text = generate_text(prefix_stats, initial_prefix, text_length)

% Write generated text to answer.txt file
with open("answer.txt", "w") as f:
    f.write(generated_text)
```

# 4 Заключение

Я создал базовую программу для генерации текста с использованием статистики префиксов и реализовал алгоритм цепей Маркова. Реализация алгоритма цепей Маркова для генерации текста позволяет создавать связный и контекстно соответствующий текст на основе заданного корпуса исходного текста. Цель программы - генерировать текст, который напоминает стиль и структуру исходного корпуса. Путем использования статистических свойств и зависимостей между словами программа способна создавать реалистичный и содержательный текст. Скорость работы программы зависит от размера входного корпуса и желаемой длины текста. Кроме того, программу можно реализовать на других языках программирования, адаптировав код под синтаксис и доступные библиотеки или функции конкретного языка. Алгоритм цепей Маркова для генерации текста предоставляет гибкое решение для различных задач, требующих возможности генерации текста.

# Список литературы


1. Котельников, И. А. LaTeX по_русски [Электронный ресурс]. / И. А. Котельников, П. З. Чеботаев. — 3_е изд., перераб. и доп. — Новосибирск: Сибирский хронограф, 2004. — 496 с. — URL: https://www.researchgate.net/publication/235255954_LaTeX_-po_russki.

2. Brian W. Kernighan Rob Pike, The Practise of programming 1999. .