Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Информационный поиск»

Студент: В. М. Филиппов
Преподаватель: А. А. Кухтичев
Группа: М8О-410Б

Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №3 «Токенизация, Стэмминг, Закон Ципфа»

1 Токенизация

Нужно реализовать процесс разбиения текстов документов на токены, который потом будет использоваться при индексации. Для этого потребуется выработать правила, по которым текст делится на токены. Необходимо описать их в отчёте, указать достоинства и недостатки выбранного метода. Привести примеры токенов, которые были выделены неудачно, объяснить, как можно было бы поправить правила, чтобы исправить найденные проблемы.

В результатах выполнения работы нужно указать следующие статистические данные:

- Количество токенов.
- Среднюю длину токена.

Кроме того, нужно привести время выполнения программы, указать зависимость времени от объёма входных данных. Указать скорость токенизации в расчёте на килобайт входного текста. Является ли эта скорость оптимальной? Как её можно ускорить?

2 Закон Ципфа

Для своего корпуса необходимо построить график распределения терминов по частотностям в логарифмической шкале, наложить на этот график закон Ципфа. Объяснить причины расхождения.

В качестве дополнительного задания, можно (но необязательно) подобрать константы для закона Мандельброта, наложить полученный график на график распределения терминов по частотностям. Привести выбранные константы.

3 Стемминг

Добавить в созданную поисковую систему лемматизацию. В простейшем случае, это просто поиск без учёта словоформ. В более сложном случае, можно давать бонус большего размера за точное совпадение слов.

Лемматизацию можно добавлять на этапе индексации, можно на этапе выполнения поискового запроса. В отчёте должна быть включена оценка качества поиска, после

внедрения лемматизации. Стало ли лучше? Изучите запросы, где качество ухудшилось. Объясните причину ухудшения и как можно было бы улучшить качество поиска по этим запросам, не ухудшая остальные запросы?

1 Описание

В рамках этой лабораторной работы, я реализовал токенизатор и стэммизатор на языке C++. Также построил на Python график для закона Ципфа.

1 Токенизатор

Алгоритм проходит по тексту посимвольно и решает: добавить символ в текущий токен или завершить текущий токен и начать новый.

Базовые символы, то есть буквы, цифры и символ подчёркивания включаются в состав токена.

Специальные символы алгоритм обрабатывает по-умному. Он умеет "склеивать"символы пунктуации с буквами, если они образуют логическую единицу:

- Числа с плавающей точкой (. или ,): Разрешает одну точку или запятую внутри числа (например, 3.14 или 10,5), если по обе стороны находятся цифры.
- Дефисы:
  - В начале слова: если после него идет цифра (отрицательные числа, например -5).
  - В середине слова: если он стоит между двумя буквами/цифрами (составные слова, например high-tech).
- Апострофы: Разрешает апостроф внутри слова, если за ним следует буква или цифра (например, it's или don't).

Такой токенайзер плохо справляется, например, с почтами и ссылками внутри текста. Например строку "Contact me at user@example.com today"он разберёт как {"contact" "me" "at" "user" "example" "com" "today"}. Почта разбилась на несколько отдельных токенов, вместо одного единого. Также некорректно разбираются строки вида "Version 1.2.3 costs $99.99 (50% off)". Из этой строки получится {"version" "1.2" "3" "costs" "99.99" "50" "off"}. Мы теряем знаки доллара и процентов. children3educationwits вот такие странные токены я встретил, кажется это раньше было ссылкой или что-то типо того. Можно запретить токены с символами и цифрами одновременно.

Для моего корпуса документов почты встречаются очень редко, так что не стоит заморачиваться над отдельными правилами для их разбора. Проценты и знаки денег также редкое явление, так что их можно разбирать.

Токенизация для корпуса размером в 305135 документов или 2661.05 МБ заняла 17.102 секунды, со скоростью 158776 КБ/с. Средний размер токена вышел 8.98245

символа, общее количество токенов составило 3696905. В среднем длина слова на английском составляет 4.7 символа, у мой токенизатор выдал чуть больше. Я связываю это, во-первых, с тем, что корпус документов содержит в себе спортивные термины, фамилии, названия клубов, которые по длине в среднем больше, чем слова из обихода. Во-вторых, вероятно, это связано с проблемами в выделении текста из HTML страницы: из-за использования специфических тэгов, которые не распознаёт GUMBO слова могут склеиваться.

2 Стеммер

Для стемминга используются алгоритм Портера. Цель стемминга - привести разные формы одного слова (например, connections, connected, connecting) к единой основе (connect). Центральное понятие в этом алгоритме – мера. Любое слово английского языка можно представить в виде $[C](VC)^m[V]$, где

- C — последовательность из одной или более согласных.
- V — последовательность из одной или более гласных.
- m — мера слова.

Алгоритм разрешает отрезать суффикс только в том случае, если оставшаяся часть слова достаточно «длинная» (имеет достаточную меру m). Это нужно, чтобы не превратить короткие слова типа sky или be в бессмысленные обрубки.

Основная часть алгоритма состоит из пяти этапов.

1. Удаление простых окончаний: Сначала убираются множественное число (-s, -es) и простые глагольные формы (-ed, -ing).
2. Сжатие длинных суффиксов: На втором и третьем этапах сложные латинские и технические суффиксы заменяются на более простые.
3. Финальная обрезка: На четвертом этапе удаляются оставшиеся суффиксы типа -ance, -able, -ion, если слово остается узнаваемым.
4. Чистка: В самом конце алгоритм убирает лишние буквы (например, немую -e в конце слова или двойные согласные).

Поскольку алгоритму не нужен огромный словарь всех слов языка, он работает невероятно быстро и занимает минимум памяти.

Однако, поскольку это просто набор правил, то случаются и промахи

- Перебор: Может превратить universal, university и universe в одну основу univers, хотя это разные понятия.
- Недобор: Может не понять, что knew и know — это одно и то же, потому что это неправильный глагол, а алгоритм знает только правила.

Токенизация вместе со стэммингом для корпуса размером в 305135 документов или 2661.05 МБ заняла 191.491 секунды, со скоростью 14230 КБ/с. Средний размер токена вышел 8.71398 символа, общее количество токенов составило 3361882. Средний размер токена упал на 3%, а количество токенов упало на 9%. На самом деле, очень небольшое падение. Вероятно, это связано с тем, что около 900000 токенов - это числа, а еще в корпусе много имён собственных, которые плохо сокращаются стеммером.

3 Закон Ципфа

Закон Ципфа — это эмпирическая закономерность, которая описывает распределение частоты слов в естественном языке. Если говорить просто: в любом достаточно большом тексте (или корпусе текстов) самое частое слово встречается примерно в 2 раза чаще, чем второе по частоте, в 3 раза чаще, чем третье, и так далее.

$$f(r) = \frac{C}{r^s}$$

, где

- $f(r)$ — частота встречаемости слова.
- C — константа (соответствует частоте самого популярного слова, если s=1).

или в развёрнутом виде

$$P_r = \frac{\frac{1}{r^s}}{\sum_{n=1}^{N} \frac{1}{n^s}}$$

, где

- f — частота (вероятность появления) слова с рангом r.
- r — ранг слова (позиция в списке частотности по убыванию).
- N — общее количество слов в словаре (размер лексикона).
- s — показатель степени, характеризующий распределение (в классическом законе Ципфа для естественных языков $s = 1$).

Закон Ципфа для токенов со стеммингом имеет более ярко выраженный горб, который оказывается прижат к линии, если стемминг не использовать. Это легко объяснимо: мы искусственно увеличиваем частоту встречи символа в тексте, поэтому график отклоняется от теоретических значений.

Резкий уход вниз от теоретического закона связан с тем, что в токенах оказывается очень большое количество чисел, зачастую уникальных, поэтому вот такое "падение"графика.

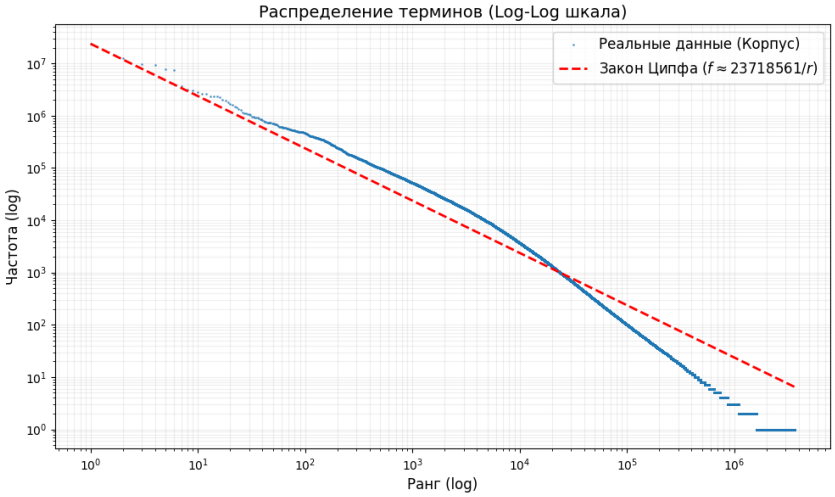

Рис. 1: Распределение Ципфа для токенов без стемминга

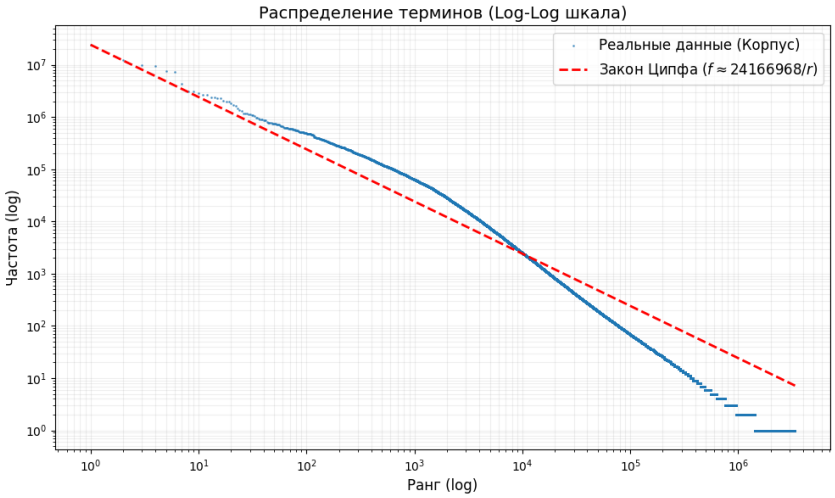

Рис. 2: Распределение Ципфа для токенов со стеммингом

2 Исходный код

Для реализации токенизатора используется класс Tokenizer, внутри которого реализуется метод tokenize, который делает основную токенизацию. Также есть метод getRawTokens, который будет необходим позже для сбора всяких служебных символов из запроса: отрицание, пересечение и объединение.

Для реализации стеммера используется интерфейс IStemmer вместе с реализациями DummyStemmer и PorterStemmer. Интерфейс необходим для того, чтобы легко подменять для тестирования отсутствия стеммера в лице DummyStemmer, который возвращает просто тоже слово при стемминге. PorterStemmer реализует алгоритм Портера, включая методы шагов, измерения меры, проверки CVC последовательностей и прочее.

```
1  #pragma once
2
3  #include <string>
4  #include <string_view>
5  #include <vector>
6
7  class IStemmer {
8  public:
9      virtual ~IStemmer() = default;
10     virtual std::string stem(std::string word) = 0;
11 };
12
13 class PorterStemmer : public IStemmer {
14 private:
15     std::string word;
16     int end;
17     int j;
18     bool isVowel(int i) const;
19     int getMeasure(int limit) const;
20     bool m_condition(int minM) const;
21     bool hasVowel() const;
22     bool isDoubleConsonant(int i) const;
23     bool isCVC(int i) const;
24     bool endsWith(const std::string& w, const std::string& suffix) const;
25     bool replaceSuffixIfMeasure(std::string& w, const std::string& suffix, const std::
           string& replacement, int minM);
26     bool replaceSuffixIfVowel(std::string& w, const std::string& suffix, const std::
           string& replacement);
27     void step1();
28     void step2();
29     void step3();
30     void step4();
31     void step5();
32 public:
```

```
    std::string stem(std::string input_word) override;
    PorterStemmer() = default;
};

class DummyStemmer : public IStemmer {
public:
    std::string stem(std::string word) override;
    ~DummyStemmer() = default;
};

class Tokenizer {
private:
    std::vector<std::string> tokens;
    uint64_t total_len;
    std::unique_ptr<IStemmer> stemmer;

public:
    Tokenizer(std::unique_ptr<IStemmer> stemmer);
    Tokenizer();
    virtual void tokenize(const std::string_view& text);
    virtual std::vector<std::string> getRawTokens(const std::string_view& text) const;

    std::vector<std::string> getTokens() const;
    size_t tokensAmount() const;
    double avgTokenLen() const;
    char* nextToken();
};
```

```
#include "tokenizer.h"

#include <cctype>
#include <memory>
#include <string>
#include <vector>

Tokenizer::Tokenizer(std::unique_ptr<IStemmer> s) : stemmer(std::move(s)) {}

Tokenizer::Tokenizer() : stemmer(std::make_unique<DummyStemmer>()) {}

std::vector<std::string> Tokenizer::getRawTokens(const std::string_view& query) const
    {
    std::vector<std::string> rawTokens;
    std::string currentToken;
    for (size_t i = 0; i < query.length(); ++i) {
        char c = query[i];
        bool is_op_char = (c == '(' || c == ')' || c == '!' || c == '&' || c == '|');
        if (c == ' ' || is_op_char) {
            if (!currentToken.empty()) {
                rawTokens.push_back(currentToken);
                currentToken.clear();
```

```
            }
            if (is_op_char) {
                rawTokens.emplace_back(1, c);
            }
        } else {
            currentToken += c;
        }
    }
    if (!currentToken.empty()) rawTokens.push_back(currentToken);

    return rawTokens;
}

void Tokenizer::tokenize(const std::string_view& text) {
    tokens.clear();
    tokens.reserve(text.size() / 6);
    total_len = 0;

    std::string current_token;
    current_token.reserve(32);

    int dots_in_token = 0;

    auto flush_token = [&]() {
        if (!current_token.empty()) {
            total_len += current_token.size();
            std::string stemmed_token = stemmer->stem(current_token);
            tokens.push_back(std::move(stemmed_token));
            current_token.clear();
            dots_in_token = 0;
        }
    };

    for (size_t i = 0; i < text.size(); ++i) {
        unsigned char raw_c = static_cast<unsigned char>(text[i]);
        char c = std::tolower(raw_c);

        bool is_basic_char = std::isalnum(raw_c);
        bool should_include = false;

        if (c == '.' || c == ',') {
            if (dots_in_token == 0 && !current_token.empty() && std::isdigit(
                static_cast<unsigned char>(current_token.back()))) {
                if (i + 1 < text.size() && std::isdigit(static_cast<unsigned char>(text[
                    i + 1]))) {
                    should_include = true;
                    dots_in_token++;
                }
            }
```

```
69          }
70
71          if (c == '\'') {
72              if (!current_token.empty() && i + 1 < text.size() && std::isalnum(
                    static_cast<unsigned char>(text[i + 1]))) {
73                  should_include = true;
74              }
75          }
76
77          if (is_basic_char || should_include) {
78              if (!current_token.empty() && is_basic_char && !should_include) {
79                  unsigned char last_raw = static_cast<unsigned char>(current_token.back()
                        );
80                  if (std::isalnum(last_raw)) {
81                      bool last_is_digit = std::isdigit(last_raw);
82                      bool curr_is_digit = std::isdigit(raw_c);
83                      if (last_is_digit != curr_is_digit) {
84                          flush_token();
85                      }
86                  }
87              }
88              current_token += c;
89          } else {
90              flush_token();
91          }
92      }
93
94      flush_token();
95  }
96
97  std::vector<std::string> Tokenizer::getTokens() const { return tokens; }
98
99  size_t Tokenizer::tokensAmount() const { return tokens.size(); }
100
101 double Tokenizer::avgTokenLen() const { return tokens.empty() ? 0 : double(total_len)
        / tokens.size(); }
102
103 std::string DummyStemmer::stem(std::string word) { return word; }
104
105 bool PorterStemmer::isVowel(int i) const {
106     if (i < 0 || i > end) return false;
107     char c = word[i];
108     if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') return true;
109     if (c == 'y') {
110         return (i > 0) && !isVowel(i - 1);
111     }
112     return false;
113 }
114
```

```cpp
int PorterStemmer::getMeasure(int limit) const {
    int measure = 0;
    bool vc = false;

    for (int i = 0; i <= limit; ++i) {
        if (isVowel(i)) {
            vc = false;
        } else {
            if (!vc) {
                ++measure;
                vc = true;
            }
        }
    }
    return measure;
}

bool PorterStemmer::m_condition(int minM) const { return getMeasure(j) > minM; }

bool PorterStemmer::hasVowel() const {
    for (int i = 0; i <= j; ++i) {
        if (isVowel(i)) return true;
    }
    return false;
}

bool PorterStemmer::isDoubleConsonant(int i) const {
    if (i < 1 || isVowel(i) || isVowel(i - 1)) return false;
    return (word[i] == word[i - 1]);
}

bool PorterStemmer::isCVC(int i) const {
    if (i < 2 || isVowel(i) || !isVowel(i - 1) || isVowel(i - 2)) return false;
    char c = word[i];
    return !(c == 'w' || c == 'x' || c == 'y');
}

bool PorterStemmer::endsWith(const std::string& w, const std::string& suffix) const {
    if (w.size() < suffix.size()) return false;
    return w.compare(w.size() - suffix.size(), suffix.size(), suffix) == 0;
}

bool PorterStemmer::replaceSuffixIfMeasure(std::string& w, const std::string& suffix,
    const std::string& replacement, int minM) {
    if (!endsWith(w, suffix)) return false;

    size_t suffix_len = suffix.size();
    size_t current_size = w.size();

```

```
    j = static_cast<int>(current_size - suffix_len - 1);

    if (m_condition(minM)) {
        w.resize(current_size - suffix_len);
        w += replacement;
        end = static_cast<int>(w.size() - 1);
        return true;
    }
    return false;
}

bool PorterStemmer::replaceSuffixIfVowel(std::string& w, const std::string& suffix,
    const std::string& replacement) {
    if (!endsWith(w, suffix)) return false;

    size_t suffix_len = suffix.size();
    size_t current_size = w.size();

    j = static_cast<int>(current_size - suffix_len - 1);

    if (hasVowel()) {
        w.resize(current_size - suffix_len);
        w += replacement;
        end = static_cast<int>(w.size() - 1);
        return true;
    }
    return false;
}

void PorterStemmer::step1() {
    if (end < 0) return;

    if (endsWith(word, "sses")) {
        word.resize(word.size() - 2);
        end -= 2;
    } else if (endsWith(word, "ies")) {
        word.resize(word.size() - 3);
        word += 'i';
        end -= 2;
    } else if (endsWith(word, "ss")) {
    } else if (endsWith(word, "s")) {
        if (end > 0 && word.at(end - 1) != 's') {
            word.pop_back();
            end--;
        }
    }

    bool rule1b_applied = false;

```

```
211     if (endsWith(word, "eed")) {
212         replaceSuffixIfMeasure(word, "eed", "ee", 0);
213     } else {
214         if (endsWith(word, "ed")) {
215             if (replaceSuffixIfVowel(word, "ed", "")) rule1b_applied = true;
216         } else if (endsWith(word, "ing")) {
217             if (replaceSuffixIfVowel(word, "ing", "")) rule1b_applied = true;
218         }
219     }
220 
221     if (rule1b_applied) {
222         char last = word.back();
223         if (endsWith(word, "at") || endsWith(word, "bl") || endsWith(word, "iz")) {
224             word += 'e';
225             end++;
226         } else if (isDoubleConsonant(end) && !(last == 'l' || last == 's' || last == 'z
                'i)) {
227             word.pop_back();
228             end--;
229         } else if (getMeasure(end) == 1 && isCVC(end)) {
230             word += 'e';
231             end++;
232         }
233     }
234 
235     if (end > 0 && word.back() == 'y') {
236         j = end - 1;
237         if (hasVowel()) {
238             word.back() = 'i';
239         }
240     }
241 }
242 
243 void PorterStemmer::step2() {
244     if (word.size() <= 2) return;
245 
246     if (replaceSuffixIfMeasure(word, "ational", "ate", 0)) return;
247     if (replaceSuffixIfMeasure(word, "tional", "tion", 0)) return;
248     if (replaceSuffixIfMeasure(word, "enci", "ence", 0)) return;
249     if (replaceSuffixIfMeasure(word, "anci", "ance", 0)) return;
250     if (replaceSuffixIfMeasure(word, "izer", "ize", 0)) return;
251     if (replaceSuffixIfMeasure(word, "abli", "able", 0)) return;
252     if (replaceSuffixIfMeasure(word, "alli", "al", 0)) return;
253     if (replaceSuffixIfMeasure(word, "entli", "ent", 0)) return;
254     if (replaceSuffixIfMeasure(word, "eli", "e", 0)) return;
255     if (replaceSuffixIfMeasure(word, "ousli", "ous", 0)) return;
256     if (replaceSuffixIfMeasure(word, "ization", "ize", 0)) return;
257     if (replaceSuffixIfMeasure(word, "ation", "ate", 0)) return;
258     if (replaceSuffixIfMeasure(word, "ator", "ate", 0)) return;
```

```
259     if (replaceSuffixIfMeasure(word, "alism", "al", 0)) return;
260     if (replaceSuffixIfMeasure(word, "iveness", "ive", 0)) return;
261     if (replaceSuffixIfMeasure(word, "fulness", "ful", 0)) return;
262     if (replaceSuffixIfMeasure(word, "ousness", "ous", 0)) return;
263     if (replaceSuffixIfMeasure(word, "aliti", "al", 0)) return;
264     if (replaceSuffixIfMeasure(word, "iviti", "ive", 0)) return;
265     if (replaceSuffixIfMeasure(word, "biliti", "ble", 0)) return;
266     if (replaceSuffixIfMeasure(word, "logi", "log", 0)) return;
267 }
268 
269 void PorterStemmer::step3() {
270     if (word.size() <= 2) return;
271 
272     if (replaceSuffixIfMeasure(word, "icate", "ic", 0)) return;
273     if (replaceSuffixIfMeasure(word, "ative", "", 0)) return;
274     if (replaceSuffixIfMeasure(word, "alize", "al", 0)) return;
275     if (replaceSuffixIfMeasure(word, "iciti", "ic", 0)) return;
276     if (replaceSuffixIfMeasure(word, "ical", "ic", 0)) return;
277     if (replaceSuffixIfMeasure(word, "ful", "", 0)) return;
278     if (replaceSuffixIfMeasure(word, "ness", "", 0)) return;
279 }
280 
281 void PorterStemmer::step4() {
282     if (word.size() <= 2) return;
283 
284     if (replaceSuffixIfMeasure(word, "al", "", 1)) return;
285     if (replaceSuffixIfMeasure(word, "ance", "", 1)) return;
286     if (replaceSuffixIfMeasure(word, "ence", "", 1)) return;
287     if (replaceSuffixIfMeasure(word, "er", "", 1)) return;
288     if (replaceSuffixIfMeasure(word, "ic", "", 1)) return;
289     if (replaceSuffixIfMeasure(word, "able", "", 1)) return;
290     if (replaceSuffixIfMeasure(word, "ible", "", 1)) return;
291     if (replaceSuffixIfMeasure(word, "ant", "", 1)) return;
292     if (replaceSuffixIfMeasure(word, "ement", "", 1)) return;
293     if (replaceSuffixIfMeasure(word, "ment", "", 1)) return;
294     if (replaceSuffixIfMeasure(word, "ent", "", 1)) return;
295 
296     if (endsWith(word, "ion")) {
297         j = (int)word.size() - 4;
298         if (j >= 0 && (word[j] == 's' || word[j] == 't')) {
299             if (m_condition(1)) {
300                 word.resize(word.size() - 3);
301                 end -= 3;
302                 return;
303             }
304         }
305     }
306 
307     if (replaceSuffixIfMeasure(word, "ou", "", 1)) return;
```

```
    if (replaceSuffixIfMeasure(word, "ism", "", 1)) return;
    if (replaceSuffixIfMeasure(word, "ate", "", 1)) return;
    if (replaceSuffixIfMeasure(word, "iti", "", 1)) return;
    if (replaceSuffixIfMeasure(word, "ous", "", 1)) return;
    if (replaceSuffixIfMeasure(word, "ive", "", 1)) return;
    if (replaceSuffixIfMeasure(word, "ize", "", 1)) return;
}

void PorterStemmer::step5() {
    if (end < 0) return;

    if (word.back() == 'e') {
        j = end - 1;
        int m = getMeasure(j);

        if (m > 1 || (m == 1 && !isCVC(end - 1))) {
            word.pop_back();
            end--;
        }
    }

    if (word.size() > 1 && word.back() == 'l' && word[word.size() - 2] == 'l') {
        j = end - 1;
        if (getMeasure(j) > 1) {
            word.pop_back();
            end--;
        }
    }
}

std::string PorterStemmer::stem(std::string input_word) {
    word = std::move(input_word);

    end = static_cast<int>(word.length() - 1);

    if (word.size() <= 2) return word;

    step1();
    step2();
    step3();
    step4();
    step5();

    return word;
}
```

3 Выводы

При выполнении данной лабораторной работы я узнал, каким образом реализуется стемминг слов в естественном языке. Также я познакомился с законом Ципфа, который меня сильно удивил. Это очень красивая закономерность. С токенизацией я так или иначе сталкивался и до этого, но было полезным еще раз потренироваться в ёё написании.

Список литературы

[1] Маннинг К., Рагхаван П., Шютце Х. Введение в информационный поиск / Пер. с англ. Д. А. Клюшина. — М.: Вильямс, 2011. — 528 с. — ISBN 978-5-8459-1623-4.

[2] The Porter Stemming Algorithm [Электронный ресурс] // Tartarus. URL: https://tartarus.org/martin/PorterStemmer/ (дата обращения: 15.12.2025).

[3] Как поисковые системы оценивают тексты: закон Ципфа, индекс туманности и прочее [Электронный ресурс] // Pitcher. URL: https://pitcher.agency/blog/zakon-cipfa (дата обращения: 16.12.2025).