Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу Информационный поиск

Студент: Е. А. Коломытцева
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №1 «Токенизация»

Целью данной лабораторной работы является разработка собственной процедуры токенизации текстов документов, предназначенной для дальнейшего использования в поисковой системе.

Токенизация выполнялась по следующему правилу: токен определяется как максимальная последовательность ASCII-букв и цифр (`[A-Za-z0-9]+`). Все символы пунктуации и специальные символы используются в качестве разделителей. Все токены приводятся к нижнему регистру.

Преимущества выбранного метода токенизации:

- высокая скорость обработки за счёт линейного прохода по тексту;
- простота реализации и воспроизводимость;
- достаточность для англоязычного корпуса и булевого поиска.

Недостатки метода:

- слова с дефисами разбиваются на несколько токенов;
- не учитываются не-ASCII символы;
- числовые и формульные элементы могут создавать шумовые токены.

Статистические данные

- Объём обработанных данных: 720 077.4 KB
- Количество токенов: 89 209 599
- Средняя длина токена: 4.756 символа
- Время выполнения: 1.991 сек
- Скорость токенизации: 361 743.3 KB/s

Код

Назначение программы

Программа токенизации предназначена для последовательной обработки текстовых документов корпуса и разбиения их на элементарные единицы — токены, которые в дальнейшем используются на этапах стемминга, индексации и поиска.

Основные этапы реализации

Разработка программы токенизации включала следующие этапы:

- реализация функций классификации символов (буква, цифра, разделитель);
- реализация приведения символов к нижнему регистру;
- последовательный разбор входного текста побайтно;
- выделение токенов и подсчёт их длины;

- накопление статистики по количеству и средней длине токенов.

Алгоритм токенизации реализован как линейный проход по входному буферу без использования контейнеров стандартной библиотеки, что обеспечивает временную сложность $O(n)$.

Ключевые фрагменты кода

Функции проверки символа и приведения к нижнему регистру:

```
inline bool is_token_char(unsigned char c) {
    return (c >= '0' && c <= '9') ||
           (c >= 'A' && c <= 'Z') ||
           (c >= 'a' && c <= 'z');
}

inline unsigned char to_lower(unsigned char c) {
    if (c >= 'A' && c <= 'Z')
        return c - 'A' + 'a';
    return c;
}
```

Основной цикл разбиения текста на токены:

```
for (size_t i = 0; i < bytes_read; ++i) {
    unsigned char c = buffer[i];
    if (is_token_char(c)) {
        token[token_len++] = to_lower(c);
    } else if (token_len > 0) {
        total_tokens++;
        total_len += token_len;
        token_len = 0;
    }
}
```

Лабораторная работа №2 «Стемминг»

Целью данной лабораторной работы является нормализация словоформ с целью уменьшения размерности словаря и улучшения качества поиска. В работе использован алгоритм стемминга Портера для английского языка.

Стемминг применяется ко всем токенам после токенизации и приведения к нижнему регистру. Числовые токены сохраняются без изменений.

Использование стемминга позволяет объединить различные словоформы одного слова, однако в некоторых случаях приводит к потере точности для специализированных терминов.

Статистические данные

- Объём данных: 720 077.4 KB
- Количество токенов до стемминга: 89 209 599
- Средняя длина токена до стемминга: 4.756
- Средняя длина токена после стемминга: 4.149
- Изменённые токены: 28 552 194 (32.01%)
- Время выполнения: 22.034 сек
- Скорость: 32 679.9 KB/s

Код

Назначение программы

Программа стемминга предназначена для нормализации словоформ путём приведения слов к их основе. Это позволяет уменьшить размер словаря и повысить полноту поиска.

Основные этапы реализации

Разработка программы включала следующие этапы:

- реализация алгоритма Портера для английского языка;
- последовательное применение этапов алгоритма к каждому токену;
- интеграция стемминга в общий конвейер обработки корпуса;
- подсчёт статистики изменения токенов.

Стемминг применяется к каждому токену после токенизации и приведения к нижнему регистру.

Ключевые фрагменты кода

Основная функция стемминга:

```
int stem_word_en(char* word, int len) {
    if (len < 3) return len;
    len = step1a(word, len);
    len = step1b(word, len);
    len = step1c(word, len);
    len = step2(word, len);
    len = step3(word, len);
    len = step4(word, len);
    len = step5(word, len);
    return len;
}
```

Применение стемминга и подсчёт изменённых токенов:

```
int new_len = stem_word_en(token, token_len);
if (new_len != token_len) {
    changed_tokens++;
}
token_len = new_len;
```

Лабораторная работа №3 «Закон Ципфа»

В данной лабораторной работе исследуется распределение частот термов в корпусе документов. Перед анализом выполнялась токенизация, приведение к нижнему регистру и стемминг.

Для каждого терма была подсчитана его частота, после чего термы были отсортированы по убыванию частоты и каждому был присвоен ранг. Построен график зависимости частоты терма от его ранга в логарифмической шкале.

На график наложена теоретическая зависимость закона Ципфа:

$$f(r) = \frac{C}{r},$$

где C — частота самого частотного терма.

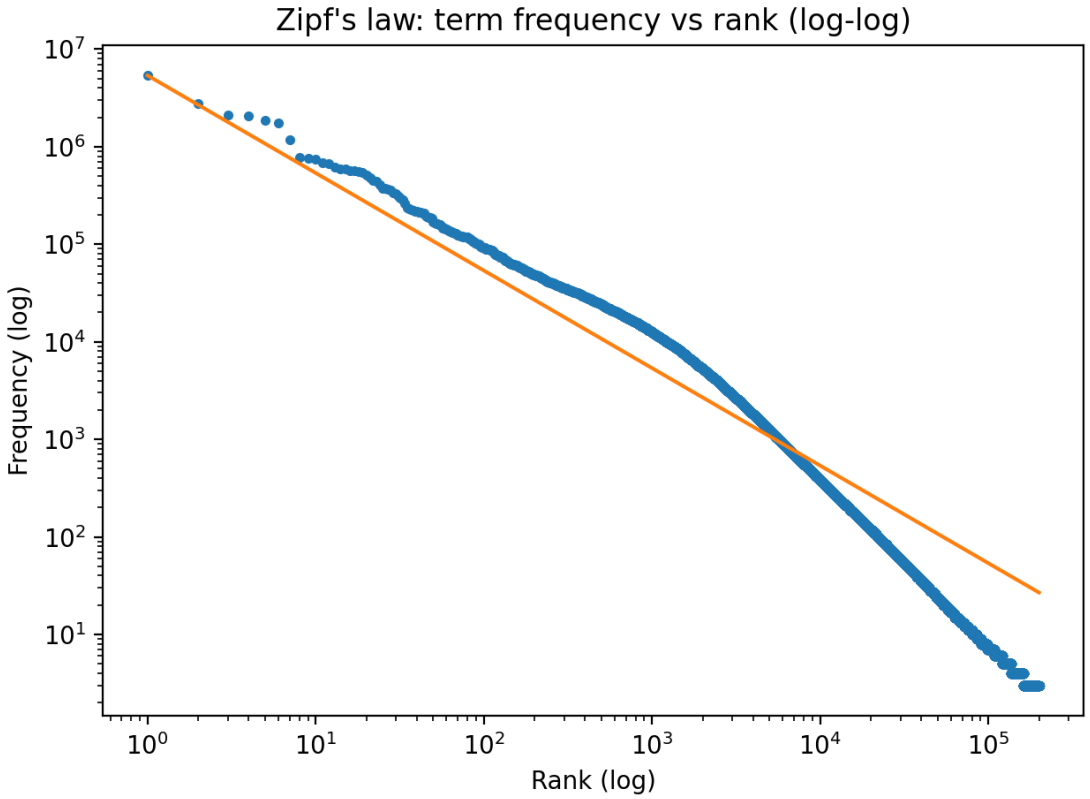

Рис. 1: Распределение частот термов и линия закона Ципфа

Анализ графика показывает хорошее соответствие закону Ципфа в среднем диапазоне рангов. Отклонения в начале распределения связаны с высокой частотой служебных слов, а отклонения в хвосте — с редкими термами и шумом токенизации.

Наиболее частотные термы

В таблице 1 приведены наиболее частотные термы корпуса после токенизации, приведения к нижнему регистру и применения стемминга. Как видно, на первых позициях находятся служебные слова английского языка, что является типичным для текстовых корпусов большого объёма и соответствует ожиданиям закона Ципфа.

Таблица 1: Наиболее частотные термы корпуса

Ранг	Терм	Частота
1	the	5 374 850
2	of	2 770 307
3	a	2 129 426
4	and	2 087 502
5	in	1 846 756
6	to	1 749 130
7	is	1 167 734
8	for	774 317
9	on	748 926
10	that	682 037

Код

Назначение программы

Программа анализа закона Ципфа предназначена для исследования статистических свойств корпуса документов путём подсчёта частот термов и построения зависимости частоты от ранга.

Основные этапы реализации

Процесс реализации включал:

- повторное использование конвейера токенизации и стемминга;
- подсчёт частоты каждого терма во всём корпусе;
- сортировку термов по убыванию частоты;
- сохранение результатов в формате CSV;
- построение графика с использованием внешнего скрипта.

Ключевые фрагменты кода

Подсчёт частоты термов:

```
uint32_t h = hash_term(token, token_len) % HASH_SIZE;
while (table[h].used && strcmp(table[h].term, token) != 0)
    h = (h + 1) % HASH_SIZE;

if (!table[h].used) {
    strcpy(table[h].term, token);
    table[h].freq = 0;
    table[h].used = true;
}
table[h].freq++;
```

Сортировка термов по частоте:

```
std::sort(terms, terms + term_count,
    [](const Term& a, const Term& b) {
        return a.freq > b.freq;
    });
```

Лабораторная работа №4 «Булев индекс»

Целью данной лабораторной работы является построение булева инвертированного индекса для корпуса документов.

Реализованы следующие структуры данных:

- прямой индекс, содержащий идентификатор документа, его заголовок и URL;
- обратный индекс, сопоставляющий каждому терму список документов, в которых он встречается.

Индекс хранится в бинарном формате и состоит из файлов: `lexicon.bin`, `postings.bin` и `docs.bin`. Формат индекса предусматривает возможность расширения.

Статистические данные

- Количество документов: 30 000
- Количество термов: 671 038
- Средняя длина терма: 7.585
- Среднее число уникальных термов на документ: 828.6
- Время индексации: 217.39 сек
- Скорость индексации: 3 312.3 KB/s

Код

Назначение программы

Программа индексатора предназначена для построения булева инвертированного индекса, позволяющего быстро выполнять логические поисковые запросы по корпусу документов.

Основные этапы реализации

Разработка индексатора включала:

- чтение документов корпуса и их метаданных;
- выделение множества уникальных термов для каждого документа;
- формирование временных блоков инвертированного индекса;
- сортировку и слияние блоков;
- сохранение индекса в бинарном формате.

Ключевые фрагменты кода

Добавление документа в постинг-листы:

```
for (uint32_t i = 0; i < unique_terms; ++i) {
    uint32_t term_id = get_term_id(terms[i]);
    postings[term_id].push_back(doc_id);
}
```

Сохранение словаря в бинарный файл:

```
fwrite(&lexicon_header, sizeof(lexicon_header), 1, f);
fwrite(lexicon, sizeof(LexRec), term_count, f);
fwrite(string_pool, 1, pool_size, f);
```

Лабораторная работа №5 «Булев поиск»

В данной лабораторной работе реализован булев поиск по построенному индексу. Поддерживаются следующие логические операции:

- пробел или `&&` — логическое И;
- `||` — логическое ИЛИ;
- `!` — логическое НЕ;
- круглые скобки для задания приоритета.

Поиск реализован в виде утилиты командной строки, принимающей запросы из стандартного ввода и выводящей результаты в стандартный вывод.

Пример работы

```
echo 'function' | ./search_cli --index ./out --limit 5
```

```
echo '(!network) (algorithm || proof)' | ./search_cli --index ./out
   --limit 5
```

Среднее время выполнения запроса составляет доли миллисекунды. Замедление наблюдается для сложных выражений с частыми термами и операцией OR.

Код

Назначение программы

Программа булевого поиска предназначена для выполнения логических запросов над построенным индексом и вывода списка документов, удовлетворяющих условиям запроса.

Основные этапы реализации

Реализация поиска включала:

- разбор поискового запроса и обработку операторов;
- преобразование выражения в обратную польскую нотацию;
- выполнение логических операций над постинг-листами;
- вывод результатов и статистики выполнения запроса.

Ключевые фрагменты кода

Преобразование выражения в обратную польскую нотацию:

```
while (!op_stack.empty()) {
    output.push_back(op_stack.top());
    op_stack.pop();
}
```

Реализация операции логического И:

```
while (i < na && j < nb) {
    if (a[i] == b[j]) {
        out.push_back(a[i]);
        i++; j++;
    } else if (a[i] < b[j]) {
        i++;
    } else {
        j++;
    }
}
```

Выводы

В ходе выполнения лабораторных работ по курсу «Информационный поиск» был реализован полный цикл построения простой поисковой системы, начиная со сбора корпуса документов и заканчивая выполнением булевых поисковых запросов по собственному индексу.

В процессе работы я на практике познакомилась с задачами предварительной обработки текстовых данных. Были реализованы токенизация и стемминг, что позволило лучше понять, каким образом текст приводится к нормализованному виду и как выбор правил обработки влияет на размер словаря и качество поиска. Отдельное внимание было уделено анализу производительности, что дало представление о том, какие этапы обработки являются наиболее ресурсоёмкими.

Построение распределения частот термов и анализ закона Ципфа позволили убедиться, что реальные текстовые корпуса обладают устойчивыми статистическими свойствами. Наблюдаемые отклонения от теоретической модели показали влияние служебных слов, редких термов и особенностей токенизации, что важно учитывать при проектировании поисковых систем.

Наиболее значимой частью работы стало построение булевого инвертированного индекса и реализация булевого поиска. В ходе выполнения этой части были получены практические навыки проектирования собственных бинарных форматов данных, работы с отсортированными списками и реализации логических операций над большими массивами данных. Было показано, что даже относительно простой индекс позволяет выполнять сложные поисковые запросы за доли миллисекунды.

Полученные знания и навыки могут быть использованы при разработке поисковых и аналитических систем, а также при работе с большими текстовыми данными, где требуется эффективная индексация и быстрый поиск информации.