Курсовой проект по курсу дискретного анализа: Текстовый поиск

Выполнил студент группы 08-307 МАИ *Дегтярев Денис Андреевич.*

Условие

Реализуйте систему для поиска статей по заданным словам. **Формат запуска в режиме индексации**

```
./prog index --input <input file> \
             --output <index file>
```

- `-input`: входной файл со статьями
- `-output`: выходной файл с индексом

Формат запуска в режиме поиска

```
./prog search --index <index file> \
              --input <input file> \
              --output <output file> \
              [--full-output]
```

- `-index`: входной файл с индексом
- `-input`: входной файл с запросами
- `-output`: выходной файл с ответами на запросы
- `-full-output`: переключение формата выходного файла на подробный

Формат ввода

Формат файлов для индексации:

```
<doc id="12" url="https://en.wikipedia.org/wiki?curid=12" title="Anarchism">
<текст статьи>
</doc>
<doc id="25" url="https://en.wikipedia.org/wiki?curid=25" title="Autism">
<текст статьи>
</doc>
```

Формат файла с запросами:

```
<word 1>
<word 1> & <word 2>
<word 1> | <word 2>
~<word 1>
~(<word 1> & <word 2> & <word 3>) | (<word 4> & (<word 5> | ~<word 6>))
```

Формат вывода

Формат выходного файла в режиме индеексации:

Бинарный файл, в котором хранится unordered_map и вектор названий статей

Формат выходного файла в режиме поиска:

Если опция --full-output не указана: на каждый запрос в отдельной строке выводится колич
документов подпадающих под запрос.
Если опция --full-output указана: на каждый запрос выводится отдельная строка, с количес
документов подпадающих под запрос, а затем названия всех документов подпадающих под запр
одному названию в строке.

Метод решения

Индексация:

- Создаем хэш таблицу:
  - ключ: слово
  - значение: название файла, где хранится очередь из номеров на каждое слово
- Заполняем файлы, где хранится очередь из номеров на каждое слово
- Создаем массив:
  - индекс: номер текста
  - значение: название текста
- Сохраняем хэш таблицу
- Сохраняем массив

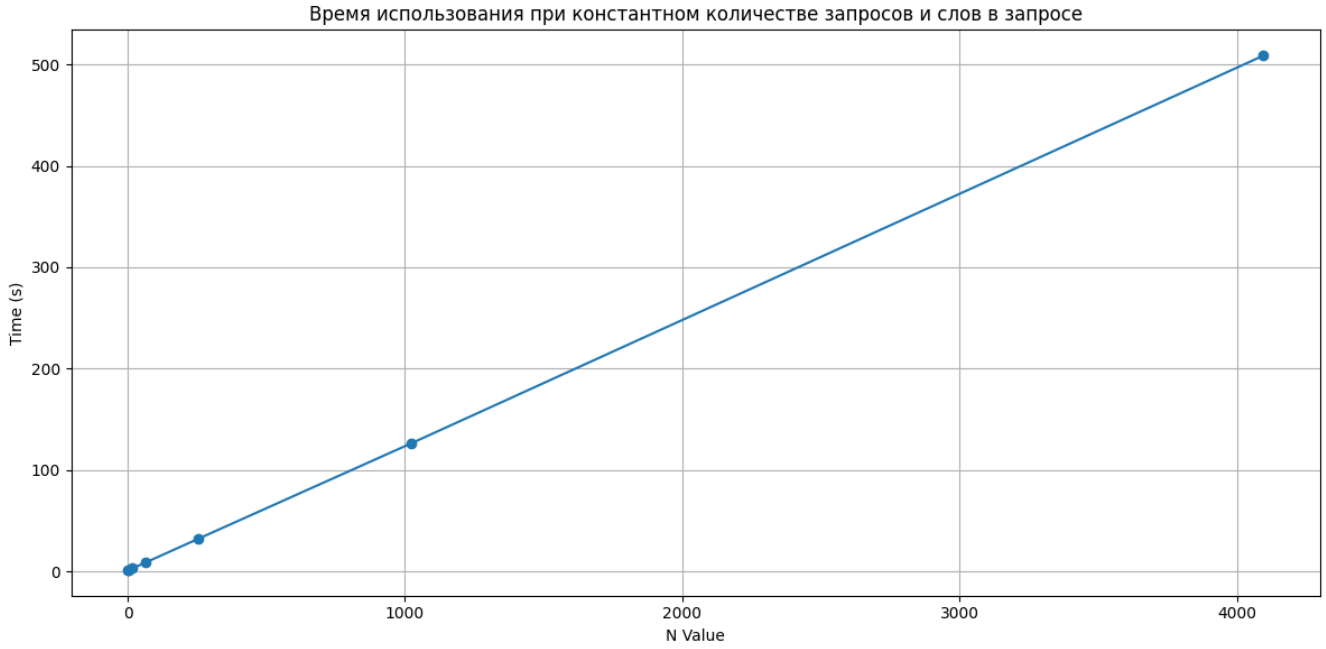
Поиск:

- Загружаем хэш таблицу и в случае флага –full-output - массив
- Преобразуем операции над множествами в польскую запись
- Выполняем операции над статьями, в которых содержатся слова в запросах:
  - преобразуем файл соответствующего слова, где хранятся индексы статей, в очередь из этих индексов
  - выполняем операции над очередями по ходу поступления в польской записи
- Выводим результат:
  - без флага –full-output: выводим размер очереди
  - с флагом –full-output: выводим размер очереди и в каждой строке название соответствующего индексу, находящемуся в очереди, статьи

Описание программы

```
src
|- main.cpp: получает на вход все входные данные и обрабатывает их
lib
|- function.hpp: содержит вспомогательные функции
|- index.hpp: содержит функции, превращающие текст в формат, удобный для хранения
   и дальнейшего использования для поиска
|- library.hpp: содержит нужные для программы библиотеки
|- load.hpp: содержит функции, превращающиеся файлы в нужный формат для дальнейших
   вычислений
|- operator.hpp: содержит функции для операций пересечения, объединения,
   логического нет
|- parser.hpp: содержит парсеры, обрабатывающие аргументы при запуске программы
   и запросы при поиске
|- save.hpp: содержит функции, преобразующие слова и статьи, в которых они содержатся
   в удобный для хранения и запуска формат
|- search.hpp: содержит функции, преобразующие запросы в постфиксный вид и производящие
   вычисления операций над множествами
```

Тест производительности



Особенности программы

- Все операции вставки выполняются за $O(1)$.
- Операции пересечения и объединения в режиме поиска суммарно выполняются за $O\left(\sum_{i=1}^{k} queue_i.size()\right)$, где k - количество знаков или | в запросе. То есть, при одном слове в строке поиска, поиск будет выполняться за $O(1)$.
- Операция логического "нет"выполняется за $O(n)$, где n - количество всевозможных статей.
- Никакой нагрузки на оперативную память, даже при огромном числе слов за счет хранения нужной информации на жестком диске.
- Все операции при поиске линейны, что обеспечивает высокую скорость работы.

Выводы

В процессе реализации инвертированного индекса и алгоритма поиска по текстам для написания курсового проекта по предмету «Дискретный анализ» я достиг нескольких важных целей и узнал много нового:

1. **Понимание инвертированных индексов**: Я получил глубокое понимание того, как работают инвертированные индексы, которые являются основой многих систем полнотекстового поиска. Это знание может быть применено в разработке поисковых систем и в обработке больших объёмов текстовых данных.
2. **Работа с хэш-таблицами**: Я использовал `unordered_map` для создания индекса, что помогло мне практически применить знания о хэш-таблицах, включая их эффективность и способы применения в реальных задачах.
3. **Алгоритмическое мышление**: Решение задачи требовало алгоритмического подхода, в частности, для определения пересечения наборов текстов. Это укрепило мои навыки в области алгоритмов и структур данных.
4. **Работа со строками и текстовыми данными**: Я практиковался в обработке и анализе строк, что полезно во многих областях программирования, от разработки программного обеспечения до анализа данных.
5. **Проблемы масштабируемости и оптимизации**: Я столкнулся с вопросами масштабируемости и оптимизации при обработке большого количества текстов и запросов, что представляет собой важный аспект в разработке производительных приложений.
6. **Умение решать практические задачи**: Я успешно применил теоретические знания для решения практической задачи, что является важным навыком в любой инженерной дисциплине.

Таким образом, выполнение этой работы не только помогло мне развить технические навыки в области компьютерных наук, но и дало ценный опыт в решении реальных задач, который может быть применен в будущих проектах.