МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчет по лабораторной работе №2
«Булев индекс»
по курсу
«Информационный поиск»

Группа: М80-108М-17
Выполнил: Забарин Н.И.
Преподаватель: А.Л. Калинин

Москва, 2018

Задание

Требуется построить поисковый индекс, пригодный для булева поиска, по подготовленному в ЛР1 корпусу документов.

Требования к индексу:

- Самостоятельно разработанный, бинарный формат представления данных. Формат необходимо описать в отчёте, в побайтовом представлении.
- Формат должен предполагать расширение, т.к. в следующих работах он будет меняться под требования новых лабораторных работ.
- Использование текстового представления или готовых баз данных не допускается.
- Кроме обратного индекса, должен быть создан «прямой» индекс, содержащий в себе как минимум заголовки документов и ссылки на них (понадобятся для выполнения ЛР2, при генерации страницы поисковой выдачи).
- Для термов должна быть как минимум понижена капитализация.

Метод решения

Индексация происходит за один проход.

Поскольку словарь токенов не помещается в оперативную память (было выставлено искусственное ограничение на 1ГБ) процесс индексации состоит из 3 частей.

На первом этапе собирается словарь в котором ключом является токен, а значением список идентификаторов статей. После обработки каждой статьи считается приближенный размер индекса содержащегося в словаре, как только размер превышает пороговое значение в 200 МБ происходит запись индекса на диск и обнуление словаря, это второй этап. После записи возвращаемся к построению индекса.

Последний этап это слияние всех полученных индексов в один.

Прямой индекс с документами хранится в БД(описано в ЛР1).

Токенизация

Процесс разбиения статьи на токены довольно примитивен, заменяем все символы кроме букв латинского алфавита и цифр на пробелы, далее делаем сплит и получаем набор токенов, далее токены проходят через лемматизатор, (в примерах описанных ниже лемматизатор возвращает неизмененную строку).

Дальше реализовано два способа возврата токенов, как сет уникальных токенов(теряем позиции, не сможем сортировать результаты поиска) или как словарь, где токен — ключ, а значение — список позиций.

Код извлечения уникальных токенов из документа:

```
def extract_token_list(obj):
    text = obj.get('text', '')

    #prepare text
    text = re.sub(wsre, ' ', text)
    text = text.lower()

    token_list = text.split()
    return token_list

def extract_token_set(obj):
    return {i for i in extract_token_list(obj) if i and check_token(i)}
```

Бинарный формат

Индекс сохраняется на диск в виде 2 файлов с одинаковым названием и расширениями .id(index data) и .ih(index header).

Id-файл состоит из блоков каждый блок имеет одинаковую структуру:

1. Число L описывающее длину токена.
2. Токен состоящий из L байт.
3. Число N описывающее длину списка идентификаторов документов в которых встречается данный токен.
4. N чисел являющихся вышеописанным списком.

Все форматы чисел можно варьировать, в текущей реализации все числа это 4-х байтные беззнаковые числа.

Заголовочный файл представляется из себя список смещений на все блоки в файле с данными и имеет следующий формат:

1. Число N описывающее длину списка смещений.
2. N чисел являющихся вышеописанным списком.

Блоки с токенами хранятся в файле в отсортированном порядке, для осуществления сортировки слиянием и бинарного поиска по индексу.

Код функций для конвертации данных в бинарный формат:

```
@classmethod
def _str_struct(cls, l):
    if l > 256 ** cls.len_size:
        raise Exception("[IndexStorage._str_struct] Too long string(len={})".format(l))
    return struct.Struct('{}{}s'.format(cls.len_format, l))

@classmethod
def _lst_struct(cls, l):
    if l > 256 ** cls.len_size:
        raise Exception("[IndexStorage._lst_struct] Too long list(len={})".format(l))
    return struct.Struct('{}{}'.format(cls.len_format, cls.int_format * l))

@classmethod
def _int_struct(cls):
    return struct.Struct(cls.int_format)

@classmethod
def _len_struct(cls):
    return struct.Struct(cls.len_format)
```

Слияние индексов

После построения маленьких индексов необходимо слить их в один. Для этого будем действовать по след алгоритму:

1. разбить список индексов на пары, если индексов нечетное число, то с лишним ничего не делать
2. попарно слить индексы используя сортировку слиянием в новый индекс
3. получаем новый список индексов который включает в себя лишний индекс из п.1 если такой был
4. повторяем п.1-3 пока длина списка не станет равной 1.

Код основного цикла в функции слияния индексов:

```
while pos[0] < n[0] or pos[1] < n[1]:
    if pos[0] < n[0] and pos[1] < n[1]:
        if term[0] == term[1]:
            l1 = self._read_list(f[0], p[0])
            l2 = self._read_list(f[1], p[1])
            lo = set(l1) | set(l2)
            termo = term[0]
            read_term(0)
            read_term(1)
        else:
            if term[0] < term[1]:
                cur = 0
            else:
                cur = 1

            lo = self._read_list(f[cur], p[cur])
            termo = term[cur]
            read_term(cur)
    else:
        if pos[0] >= n[0]:
            cur = 1
        else:
            cur = 0

        lo = self._read_list(f[cur], p[cur])
        termo = term[cur]
        read_term(cur)


    hout.append(fout.tell())
    fout.write(self.storage.str2byte(termo))
    fout.write(self.storage.lst2byte(list(lo)))
```

Статистика

Индексировались все статьи английской Википедии, порядка 20ГБ текста, после очистки. Конфигурация системы 1Гб DDR3 памяти, 1 физ. ядро AMD A8-7650K, 100ГБ SSD M.2.

На выходе было получено 23 индекса размером около 200МБ.
Среднее число токенов в индексе 1,3 млн.
Затраченное время примерно 4500 сек.
Средняя скорость 4,6 МБ/с.
Среднее время обработки статьи 0.5мс.

Время затраченное на слитие индексов примерно 1300 сек.
Итоговый размер индекса около 4Гб.

Выводы

Реализация индексатора описанная выше позволяет работать с любым числом документов, но при этом ограничена по размеру документа, он должен полностью помещаться в оперативную память вместе с индексом построенным по нему.

Еще несколько особенностей реализации:

- Хранение индекса в одном файле удобно для бинарного поиска, с несколькими нюансами. При хранении индекса на HDD можно быстро сломать диск, а так же это сильно снизит скорость обработки запросов.
- Класс описывающий сохранение индекса легко изменяется на другие структуры и типы данных.
- Из-за применения операции сортировки слиянием необходимо иметь запас памяти примерно в 2 раза превосходящий конечный размер индекса.