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

Отчет по курсовому проекту
по курсу
«Информационный поиск»

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

Москва, 2018

Задание

Необходимо проиндексировать англоязычную Википедию, реализовать поиск по полученному индексу.

Метод решения

Дамп английской википедии брался с ресурса https://meta.wikimedia.org/wiki/Data_dump_torrents, конкретно взят архив enwiki-20170301-pages-articles.xml.bz2.

В архиве находится xml-файл размера около 60Гб.

Для извлечения и очистки текстов статей был разработан отдельный скрипт на Python3. Исходные тексты хранятся в PostgressSQL.

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.

Поисковый API состоит из двух запросов search и get_results. Первый принимает запрос обрабатывает и сохраняет результат на стороне сервера, возвращает id поиска *(случайный набор символов)*. Второй запрос принимает id поиска и номер страницы для отображения. Синтаксис поисковых запросов:

- Пробел или два амперсанда, «&&», соответствуют логической операции «И».
- Две вертикальных «палочки», «||» – логическая операция «ИЛИ»
- Восклицательный знак, «!» – логическая операция «НЕТ»
- Могут использоваться скобки.

Для демонстрации работы поисковой системы был реализован веб-сервис, реализующий базовую функциональность поиска из двух страниц:

- Начальная страница с формой ввода поискового запроса.
- «что где когда»/2 - такому запросу удовлетворяют документы содержащие в себе термины что, где и когда на расстоянии не более 2, без нарушения порядка. Если элемент /2 отсутствует то термины должны идти подряд.

Страница поисковой выдачи, содержащая в себе форму ввода поискового запроса, 50 результатов поиска в виде текстов заголовков документов и ссылок на эти документы, а так же ссылку на получение следующих 50 результатов.

Целые числа в индексе сжимаются с помощью VarByte.

Поддерживается операция прыжка по индексу:

1. Прыгать будем по документам
2. Пусть L длина списка документов в которых есть термин, тогда введем 2 функции
    1. p = P(L) — частота прыжка, то есть если номер элемента в списке длины L кратен p то за этим элементом находится блок прыжка.
    2. o = O(L) — длина прыжка.
3. p должно быть примерно равно корню L. Я выбрал след формулу P(L)=round_up(sqrt(L-1)).
4. Довольно очевидно что o <= p, иначе мы прыжком перепрыгиваем потенциальный прыжок.
5. Если o < 3 не будем прыгать и вставлять прыжки. Понятно что прыжок должен выполняться быстрее чем проход по элементам через которые мы прыгаем.

Первоначальная формула для o: O(L) = round_up((L-1) / (P(L)*3))

Поисковая выдача сортируется по TF-IDF индксам.

Используется лемматизатор из библиотеки NLTK.

Так же реализован зонный поиск для поиска по заголовкам, для результатов найденых в заголовках индекс TF-IDF умножается на 5.

Поисковая выдача обогащается за счет построения динамических сниппетов.

Побайтовое описание формата

Блок в файле-индексе имеет следующую структуру:

1. 4 байта, длина терма, l
2. l байт под сам терм
3. 4 байта, размер блока, sz
4. 4 байта, длина блока, n
5. n «вхождений» следующего вида
    1. 4 байта, номер документа
    2. 4 байта, размер «вхождения», k
    3. k байт, содержат список координат

TF-IDF подробно

Что бы в поисковой выдаче документы были отсортированы необходимо сначала получить все результаты поиска, что невозможно из-за ограничения по оперативной памяти, от временной записи результатов на диск придется отказаться, т.к. SSD диски имеют ограниченное число циклов записи. Реализована следующая схема выполняем поиск всех результатов, для каждого результата вычисляем индекс, будем хранить в специальном итераторе только 1000 лучших результатов, будем использовать для этого модуль heapq. То есть одновременно в памяти будет находиться порядка 1000 результатов поиска, это вполне можно себе позволить.

Минусы, основной недостаток это сильное ухудшение производительности, время выполнения запроса прямо пропорционально количеству результатов. Вторым минусом является ограничение на 1000 результатов, не считаю это плохим результатом, тк это число легко увеличивается, главное что бы хватало оперативной памяти. И третий момент — TF-IDF метрика считается отдельно для каждого файла-индекса.

Оптимизация, возьмем два запроса «black and yellow» и «black yellow», 80%+ результатов поиска совпадает, но в одном случае обработка запроса примерно 5 сек, в другом 1 сек. Логично выкидывать высокочастотные слова из поиска, то есть если ввести порог для IDF индекса, ниже которого слова будут игнорироваться. Если же пользователь ищет что-то где присутствует высокочастотное слово можно использовать цитатный поиск, который игнорирует данный порог.

Булевы операции над TF-IDF индексом, при точном совпадении с запросом необходимо так же вычислять TF-IDF индекс, для этого необходимо определить логические операции над этим индексом:

- термин, для одиночного термина индекс будет вычисляться по обычной формуле.
- &, для логического И, индекс двух слагаемых просто складывается.
- |, для логического ИЛИ есть два варианта, когда только одно из слагаемых Истино возвращаем индекс слагаемого, если оба Истины то возвращаем сумму.
- /, для цитатного поиска все не очевидно, будем брать число вхождений цитаты в документ, умножать на число слов в цитате в квадрате, делить на число слов в документе, и умножать на константу заменяющую IDF.

!, отрицание имеет значение индекса как у слагаемого.

Статистика

Было проиндексированно 8,5 млн документов, суммарная длина всех слов около 21 миллиарда символов, на индексацию ушло 34 тысячи секунд, получено 112 индексов размера порядка 100мб. Скорость выполнения запроса из одного слова порядка 25 секунд.

Оценка качества

Оценка проводилась на выборке из 1e6 случайных статей. Сравнивать будем с встроенным поиском в Википедию и поиском Google(site:en.wikipedia.org). Считаю такое сравнение некорректным за счет сильного отличия корпусов документов по которым производится поиск, мой корпус меньше в 8,5 раз. Сравнение проводится скриптом, который выполняет один и тот же запрос во всех поисковых системах и сравнивает сколько результатов выдачи моей системы присутствуют в эталонных выдачах, сравниваются топ-100 результатов. Список запросов(P@1/P@3/P@5/P@10):

- from which books bible composed (1/1/1/1)
- What? Where? When? (0/0/0/0)
- game (1/3/4/8)
- russian aircraft factories (0/1/1/1)
- without me the nation are not complete (0/0/0/0)
- name of inhabitants of embankments (0/0/0/0)
- where they crowned Nicholas 2 (1/1/1/2)
- assistant prosecutor (0/0/0/0)
- kings of gas (ничего не найдено)
- administrative code (1/1/1/1)
- organic compounds (1/2/3/4)
- game of thrones (ничего не найдено)
- Sigmund Freud (0/1/2/2)
- Barack Obama (1/1/2/3)
- World War (0/0/1/1)
- Cristiano Ronaldo (1/2/2/2)
- Star Wars (1/3/4/4)
- Indigenous australian (0/0/0/0)
- Web scraping (0/0/0/0)
- Illuminati (1/1/2/3)
- September 11 attacks (почти все результаты вида «List of minor planets: 102001–103000»)
- List of The Big Bang Theory episodes (один странный результат)
- Princess Charlotte of Cambridge (0/0/0/0)
- when is mothers day (0/0/0/0)
- how to solve a rubix cube (0/0/0/0)
- how to write a cover letter (0/0/0/0)
- what is amazon prime (0/0/0/0)
- what is my spirit animal (0/0/0/0)
- what does og mean (0/0/0/0)
- is pluto a planet (ничего не найдено)

Большинство запросов дают мало совпадений, до двух совпадений. Лучшие результаты:

- «from which books bible composed», 6 совпадений,
- «game», 4 совпадения,
- «organic compounds», 4 совпадения,
- «Illuminati», 10 совпадений

Примеры кода

Код класса-интератора по блоку:

```
class ResIter:
    def __init__(self, index=None, op=None, fname=None, pos=0, children=None, empty=False, term=None, read_list=False, idf=None, coef=1):
        self.term = term
        self.fname = fname
        self.position = pos
```

```
    self.op = op
    self.neg = False
    self.children = children
    self.index = index
    self.read_list = read_list
    self.backup = None

    self.idf = idf
    if self.idf:
        self.idf /= math.log(10)

    self.coef = coef

    self.jump_cnt = 0
    self.jump_suc = 0
    self.jump_pos = None
    self.unjump_data = None

    if empty:
        self.type = RES_TYPE_EMPTY
    elif op and children:
        self.type = RES_TYPE_OP
        if self.op == '!':
            self.neg = True
    else:
        self.type = RES_TYPE_RES
        with open(self.index._index_file(fname=fname), 'br') as fidx:
            fidx.seek(pos, 0)
            self.block_size = storage.read_int_raw(fidx)
            self.length = storage.read_int(fidx)
            self.current_pos = fidx.tell()
            self.current_i = 0

            self.jump_p, self.jump_o = storage.get_jump_op(self.length)

@staticmethod
def _quote(a, b, d):
    ch1, ch2 = a.next(), b.next()
    while ch1 and ch2:
        if ch1['id'] == ch2['id']:
            poses = []
            bset = set(ch2['lst'])
            for i in ch1['lst']:
                for k in range(d):
                    if i+k+1 in bset:
                        poses.append(i)
                        break
            if poses:
                return (ch1['id'], poses)
            ch1, ch2 = a.next(), b.next()
        elif ch1['id'] < ch2['id']:
            ch1 = a.next()
```

```
        else:
            ch2 = b.next()
    return None

def quote(self, children, d):
    if len(children) == 1:
        return children[0].next()
    r = self._quote(children[-2], children[-1], d)
    while r:
        t = self.quote(children[:-2]+[FakeRes(r)], d)
        if t:
            return t
        r = self._quote(children[-2], children[-1], d)

    return None

def next(self):
    if self.type == RES_TYPE_EMPTY:
        return None
    if self.type == RES_TYPE_OP:
        if self.op == '&':
            ch1, ch2 = self.children
            r1, r2 = ch1.next(), ch2.next()
            if ch1.neg == ch2.neg:
                self.neg = ch1.neg
                while r1 and r2:
                    if r1['id'] == r2['id']:
                        return {'id': r1['id'], 'tf-idf': r1['tf-idf']+r2['tf-idf'], 'lst': [], 'q':r1['q']+r2['q']}
                    elif r1['id'] < r2['id']:
                        if ch1.can_jump():
                            j = ch1.jump()
                            if j['id'] < r2['id']:
                                r1 = j
                                continue
                            r1 = ch1.unjump()
                        r1 = ch1.next()
                    else:
                        if ch2.can_jump():
                            j = ch2.jump()
                            if j['id'] < r1['id']:
                                r2 = j
                                continue
                            r2 = ch2.unjump()
                        r2 = ch2.next()
            else:
                if ch1.neg:
                    ch1, ch2 = ch2, ch1
                    r1, r2 = r2, r1

                while r1 and r2:
                    if r1['id'] == r2['id']:
                        r1, r2 = ch1.next(), ch2.next()
```

```
                continue
            elif r1['id'] < r2['id']:
                ch2.revert()
                return {'id': r1['id'], 'tf-idf': r1['tf-idf']+r2['tf-idf'], 'lst': [], 'q':r1['q']}
            else:
                if ch2.can_jump():
                    j = ch2.jump()
                    if j['id'] < r1['id']:
                        r2 = j
                        continue
                    r2 = ch2.unjump()
                r2 = ch2.next()

    if self.op == '|':
        if len(self.children) != 2:
            for i in self.children:
                r = i.next()
                while r:
                    return r
        else:
            ch1, ch2 = self.children
            r1, r2 = ch1.next(), ch2.next()
            if ch1.neg == ch2.neg:
                self.neg = ch1.neg
                while r1 or r2:
                    if not r1: return r2
                    if not r2: return r1

                    if r1['id'] == r2['id']:
                        return {'id': r1['id'], 'tf-idf': r1['tf-idf']+r2['tf-idf'], 'lst': [], 'q': r1['q']+r2['q']}
                    elif r1['id'] < r2['id']:
                        ch2.revert()
                        return r1
                    else:
                        ch1.revert()
                        return r2
            else:
                if ch1.neg:
                    ch1, ch2 = ch2, ch1
                    r1, r2 = r2, r1

                self.neg = True

                while r1 and r2:
                    if r1['id'] == r2['id']:
                        r1, r2 = ch1.next(), ch2.next()
                        continue
                    elif r1['id'] < r2['id']:
                        if ch1.can_jump():
                            j = ch1.jump()
                            if j['id'] < r2['id']:
                                r1 = j
```

```
                            continue
                        r1 = ch2.unjump()
                    r1 = ch2.next()
                else:
                    ch1.revert()
                    return r2
        if self.op == '!':
            self.neg = True
            return self.children[0].next()
        if self.op[0] == '/':
            n, d = map(int, self.op[1:].split('_'))
            for i in self.children:
                i.read_list = True
            r = self.quote(self.children, d)
            if not r:
                return None
            return {'id': r[0], 'lst': r[1], 'tf-idf': n*n*len(r[1])*0.012, 'q': [i.term for i in
self.children]}
        else:
            self.save()
            with open(self.index._index_file(fname=self.fname), 'br') as fidx:
                fidx.seek(self.current_pos, 0)
                if self.current_i < self.length:
                    doc = storage.read_int(fidx)
                    doc_len = storage.read_int(fidx)
                    rate = storage.read_int(fidx)
                    tf = rate / doc_len
                    tf = TF_F(tf) * self.coef
                    if self.read_list:
                        lst = storage.read_list(fidx)
                    else:
                        lst = storage.skip_list(fidx)
                    if storage.ENABLE_JUMPS and self.current_i % self.jump_p == 0 and self.jump_o >
2 and self.current_i < self.length-1:
                        self.jump_pos = storage.read_int_raw(fidx)
                    else:
                        self.jump_pos = None
                    self.current_pos = fidx.tell()
                    self.current_i += 1
                    return {'id': doc, 'tf-idf': tf*self.idf, 'lst': lst, 'q':[self.term]}

        return None

    def can_jump(self):
        return self.jump_pos is not None

    def jump(self):
        #log("Try to jump", self.jump_pos)
        self.unjump_data = (self.current_i, self.current_pos)
        self.current_i = min(self.jump_o + self.current_i-1, self.length-1)
        self.current_pos += self.jump_pos
        self.jump_pos = None
```

```
        self.jump_cnt += 1
        self.jump_suc += 1
        return self.next()

    def unjump(self):
        self.current_i, self.current_pos = self.unjump_data
        self.jump_suc -= 1
        return self.next()

    def update_jump_cnt(self):
        if self.type == RES_TYPE_OP:
            self.jump_cnt = 0
            self.jump_suc = 0
            for i in self.children:
                self.jump_cnt += i.jump_cnt
                self.jump_suc += i.jump_suc

    def save(self):
        self.backup = (self.current_i, self.current_pos)

    def revert(self):
        if self.backup:
            self.current_i, self.current_pos = self.backup
```

Код содержащий «низкоуровневую» запись в файл индекса:

```
int_size = 4
int_format = 'I'
jump_size = 1 * int_size

ENABLE_VARBITE = True
ENABLE_LIST_CMP = True
ENABLE_JUMPS = True

def get_jump_op(l):
    p = max(1, round_up(sqrt(l-1) / 3)) # jump number
    o = min(p-1, max(1, round_up(math.log(l, 1.3)))) # jump len

    return p, o

def write_int_raw(a):
    return struct.pack(int_format, a)

def write_int(a):
    if ENABLE_VARBITE:
        return encode_int(a)
    return write_int_raw(a)

def write_str(s):
    b = bytes(s, encoding='utf8')
    return write_int(len(s)) + struct.Struct('{}s'.format(len(b))).pack(b)

def write_term(s, idf):
```

```
    return write_int(int(1000*idf)) + write_str(s)

def write_list(l):
    if ENABLE_LIST_CMP:
        nl = [l[0]]
        for i in range(1, len(l)):
            nl.append(l[i] - l[i-1])
        l = nl

    b = b''.join([write_int(i) for i in l])
    return write_int(len(b)) + b

def write_entrance(doc_id, doc_len, coords):
    return write_int(doc_id) + write_int(doc_len) + write_int(len(coords)) + write_list(coords)

def write_jump(offset):
    return write_int_raw(offset)

def write_block(block, doc_lens):
    l = len(block)
    p, o = get_jump_op(l)

    entrances = []
    sizes = [0]

    doc_ids = list(block.keys())
    doc_ids.sort()

    for i in range(l): # encode entrances
        doc_id = doc_ids[i]
        entrance = list(block[doc_id])
        entrance.sort()
        entrances.append(write_entrance(doc_id, doc_lens[doc_id], entrance))
        sizes.append(sizes[-1] + len(entrances[-1]))

    sizes = sizes[1:]

    b = write_int(l)

    for i in range(l): # add jumps and join all bites
        b += entrances[i]
        if ENABLE_JUMPS and i % p == 0 and o > 2 and i < l-1:
            to = min(l-1, i+o-1)
            b += write_jump(sizes[to] - sizes[i])

    b = write_int_raw(len(b)) + b

    return b
```

Выводы

В ходе выполнения курсового проекта была реализована полноценная поисковая система. Полученная система имеет большой потенциал развития, но на ее примере я понял как работают полнотекстовые поисковые движки.