Московский государственный университет имени М.В. Ломоносова
Механико-математический факультет
Кафедра вычислительной математики

Курсовая работа

Поиск упоминаний персон и научных тематик в новостях для выявления возможного конфликта интересов при экспертизе.

Студент: Гвоздев Михаил Александрович
Научный руководитель: к.ф.-м.н. Кривчиков Максим Александрович

Группа: 510

Москва
2022

Содержание

Введение

В наше время Интернет является огромным хранилищем различных данных. Не все они одинаково полезны для конкретных задач. В данной работе рассматривается задача поиска экспертов в новостных публикациях, с целью нахождения связности между ними. Связность при этом может быть любого вида. Здесь мы считаем что связь это наличие совместного упоминания в публикации двух и более экспертов. Но чтобы этого достичь нужно научиться правильно искать нужные данные. Поисковые системы реализуют механизм получения срезов данных. Первым этапом в любом поиске является поверхностный сбор информации. Его осуществляют при помощи поисковых пауков [7]. Это программы для обхода заранее заготовленного списка адресов (URL). Они заносят найденные там тексты в специальные коллекции. Следующий этап это поиск упоминаний в самом тексте страницы. В отличие от поисковых пауков на нем нужно проходить весь текст содержащийся в ней, пытаясь найти нужную последовательность символов.

Перейдем к самой задаче данной работы. Нужно научиться находить ученых в полученных текстах и определять связаны ли они. Под связностью двух людей (рецензента и рецензируемого) будем понимать наличие знакомства, дружбы, родства или других отношений, которые могут оказать влияние на вынесение рецензентом вердикта на научную работу рецензируемого. Такие связи будем называть конфликтом интересов.

Существует большое количество промышленных реализаций отслеживания целевых слов в новостях. К бесплатным относятся Яндекс.Блоги и Google Alerts оба позволяют отслеживать конкретные слова во всех новостных изданиях, блогах и даже некоторых социальных сетях. Первый позволяет делать сужение поиска по определенной местности и дате: день, 2 недели, месяц. К платным же относятся Chotam, babkee, Медиалогия. Они специализируются на оценке отношения к бренду компании в социальных сетях и позволяют дополнительно узнавать ретроспективные данные, тональность текстов. На самом деле существует достаточно много похожих инструментов, однако ни перечисленные выше, ни остальные не помогают в решении данной задачи. Они не умеют автоматически выявлять связаны ли потенциально двое людей или нет и не способны дать ответ о наличии такого сложноопредилимого понятия как конфликт интересов.

Одним из вариантов, которыми можно искать конфликт интересов, являются графы знаний [16]. Знанием в них понимается некоторое утверждение которое известно. Они бывают разных видов [14]. Наиболее известными являются дополняющийся еженедельно Wikidata [5] с более чем 700 миллио-

нами записей и 18 миллионами объектами, Freebase – не обновляющийся с 3 миллиардами записей о 50 миллионах объектов и DBpedia – дополняющийся несколько раз в год краудсорсинговый проект с более чем 400 миллионами записей о более чем 4 миллионах объектов. Основным отличием графов знаний от обычных распределенных баз данных является наличие возможности выносить некоторые логические суждения об объекте.

Другим вариантом решения проблемы могла бы являться модель word2vec, которая сопоставляет контексту/слову наиболее вероятное следующее/предыдущее/то которому научили модель слово/контекст. Она создает специальные числовые векторы – эмбединги, для каждого слова. Таким образом можно было бы сравнивать контексты при помощи косинусного расстояния $\frac{|a||b|}{(a,b)}$, где a, b векторы эмбедингов. И делать выводы о связности двух имен на основе того насколько мало это расстояние. Однако существенным недостатком ее является необходимость в огромных вычислительных мощностях и больших объемах памяти.

В этой работе же будем выполнять только базовую часть всех предложенных идей. А именно создавать локальные, регулярно обновляемые коллекции и по ним проводить поиск сотрудников среди всех найденных текстов. Различными способами определять являются ли они именно теми людьми которых мы ищем. Текстами с потенциальным конфликтом интересов между рецензентом и рецензируемым, на данном этапе, будем считать те, в которых они оба одновременно упоминаются.

1. Известные результаты

В этом разделе перечисляются и описываются наиболее известные и близкие к теме работы. Последовательность изложения совпадает с изложением предыдущего раздела.

Поисковый паук, как было описано ранее, это программа, которая автоматически обходит определенный заранее список адресов (URL) и заносит полученные страницы в специальную коллекцию. Из полученных страниц, в свою очередь, извлекаются новые URL и добавляются в конец исходного набора. Так они работали изначально, пока весь объем данных в интернете был сравнительно мал. Пауки различаются по типам [22]. В частности, существуют *периодические* пауки общего назначения [13]. Они скачивают все указанные в списке URL, пока не наберут требуемое автору паука количество скачанных страниц и останавливаются. Эта процедура повторяется периодически, когда возникает необходимость в обновлении данных. Они не

самые эффективные с точки зрения скорости исполнения, зато отсутствует возможность не обновить заранее заданные страницы. Также существуют *пошаговые* пауки. Они имеют постоянный размер коллекции и продолжают свою работу непрерывно. Их задача беспрерывно работать и заменять наименее «полезные» страницы на более «полезные» по некоторому правилу [13]. Такой вид пауков появился в ответ на то, что данные в интернете постоянно меняются. Одни страницы появляются, другие наоборот исчезают. Не все они одинаково «полезные» и в силу ограниченности доступной памяти некоторые удаляются. Построение универсальной метрики полезности является отдельной сложной задачей. Если считать страницу важной по непрерывному количеству посещений ее, то можно удалить важную страницу на которой раз в месяц оплачиваются счета за важные услуги (например коммунальные). Или же считать полезными только те страницы на которые больше всего ежемесячная аудитория, что делает невозможным использование такого паука с некоторой узкой целью (поиска информации по своей узкой специальности). Следующим типом являются *распределенные* пауки [10]. Они состоят из многих пауков общего назначения, которые проверяют URL только в определенной области интернета, частое использование которой характерно для ограниченной географически территории. Например для определенной страны характерно использование сайтов в основном на ее языке. При этом есть центральный сервер, контролирующий и распределяющий URL между ними. Таким образом, достигается большая отказоустойчивость всей системы, хоть и существует некоторое ограничение скорости в силу использования пауков общего назначения. Эту проблему призваны были решать *параллельные* пауки [2]. В отличие от *распределенных* они обрабатывают единый массив URL, которые разделены на несколько машин, обрабатывающих эти URL параллельно, а не последовательно. Такое улучшение позволило повысить скорость выгрузки страниц. Подтипом пауков общего назначения являются *фокусированные*. Они обходят и добавляют в список дальнейшего обхода только те URL, по которым находится документ соответствующий некоторой теме (допустим теме поискового запроса). При этом используются различные методы подсчета обратных ссылок. Добавление новых ссылок происходит до тех пор пока не наберется нужное количество страниц или не обойдется весь список URL. В Google Inc. в 1998 году для подсчета обратных ссылок использовали PageRank [11].

Для описания алгоритма PageRank зададим следующие условия. Пусть на странице A цитируются страницы $S_1, ..., S_n$ (присутствуют их URL), $d \in (0, 1)$ параметр затухания (в работе брали 0.85), $C(A) -$ общее количество цитируемых страниц на странице A. Тогда $PR(A) = (1-d) + d*(\frac{PR(S_1)}{C(S_1)} + \frac{PR(S_2)}{C(S_2)} + ... + \frac{PR(S_n)}{C(S_n)})$,

где PR это PageRank. Существует множество вариантов, выбора параметров для построения фокусированных пауков. Поэтому они, в свою очередь, подразделяются на различные виды в зависимости от способа определения соответствия страницы теме и способа обработки страниц [9].

После того, как собран набор необходимых данных, актуальной становится следующая задача задача – надежное хранение и быстрый доступ к ним. Отвечая на такой запрос появился язык SQL [18] для работы с реляционными базами данных. Реляционная модель представляет собой набор двумерных таблиц. Каждая таблица состоит из строк – записей и столбцов – полей. Поля обязаны быть одного из допустимых типов. Таблицы могут быть связаны друг с другом при помощи различных ключей (ссылок). Изначально реляционные базы данных создавались для хранения на одной машине сравнительного небольшого объема данных. Когда объем их достаточно вырос старая парадигма баз данных перестала работать. Старая парадигма это выполнение всех трех свойств из следующего предложения. Слом парадигмы подтверждает CAP теорема [1], которая утверждает, что любая система общих данных может обладать не более чем двумя свойствами из трех следующих:

- Согласованность – существует только одна актуальная версия данных.
- Доступность – данные доступны в любой момент времени.
- Стабильность – устойчивость к физическим нарушениям связности частей данных.

В связи с этим стали появляться нереляционные базы данных и системы для управления ими. Они специализированы под определенные задачи и поэтому могут делать незначительные для их цели допущения в CAP теореме. В наше время можно выделить следующие типы нереляционных моделей данных [15]. Наиболее известной является модель *ключ-значение*. Принцип ее работы похож на идею хеш-таблицы. У каждой записи есть уникальный ключ, которому соответствует единственный хранящий запись сервер. Следующий тип это *документный*. Его идея состоит в том, что документы устроены гораздо сложнее, чем просто текстовые поля. Они могут содержать ссылки на другие документы, которые в свою очередь ссылаются на дополнительные и так далее. *Столбцовая* модель отличается от предыдущих тем, что хранит данны в виде столбцов, а не записей. Основополагающей системой для этой модели является Bigtable [12]. В ней данные хранятся в виде наборов столбцов одинакового типа. При этом таких наборов может быть максимум сотни, в отличие от реляционной модели в которой может быть неограниченное количество столбцов. Дальше идет *графовая*

модель. Она строится на основе модели графа из теории графов. Ее преимущество состоит в том что она позволяет эффективно обрабатывать данные с большим числом связей между объектами разной природы.

Следующий этап это поиск упоминаний в самом тексте страницы. В отличие от поисковых пауков он проходит весь текст содержащийся на ней, пытаясь найти нужную последовательность символов. Это можно делать как каждый раз проходя один и тот же текст или же строить инвертированный индекс [23]. Он представляет собой структуру данных, в которой каждому слову сопоставляются места, где оно упоминается в различных текстах. Прежде чем строить подобный индекс, слова нужно привести в некоторую единую форму. Одним из таких решений является стемминг. Этот процесс оставляет от слова только его основу, не обязательно являющуюся корнем этого слова. Одной из его наиболее популярных реализаций является стеммер Портера [21], который в последствии перерос в отдельный проект Snowball [3]. В общем виде все алгоритмы стемминга можно разбить на 3 группы [17]: усеченные, статистические и смешанные. Примером усеченного является алгоритм Портера. Статистических – концепция n-gram. Идея ее заключается в том, что похожие слова имеют очень высокий процент совпадения. Алгоритм разбивает исходное слово на части длины n. Так для слова , «слон» 3-граммами будут последовательности: «- -с», «-сл», «сло», «лон», «он-», «н- -», где, «-» означает пробел. Достоинством этой модели является ее независимость от языка, а недостатком долгая работа и большие затраты памяти. При помощи статистических методов оценивается есть ли заданное слово в том или ином документе. Другим вариантом нормализации слов является лемматизация. Она приводит слова в начальную форму (учил $\rightarrow$ учить, домов $\rightarrow$ дом). Такой вариант позволяет сохранить больше контекста при поиске определенный фраз. Однако есть более высокая вероятность ошибиться и в итоге только усложнить поиск [8]. Смешанная модель стемминга же сочетает в себе достоинства первых двух. Построенный при помощи этих методов инвертированный индекс позволяет значительно ускорить работу поисковых систем.

Дальше подходящие страницы следует отсортировать от наиболее вероятного к наименее (релевантность) на основе более продвинутых методов. Базово это можно выполнять при помощи различных вариаций алгоритма TF-IDF [20], [19]. Основная идея его заключается в том, что можно разбить формулу определения релевантности на 2 множителя. TF (term frequency) описывает как часто искомое слово встречается в тексте страницы (документе). Например $\frac{w}{W}$, где w – количество совпадающих слов, W – число всех слов в документе. IDF (inverse document frequency) же является величиной

пропорциональной отношению релевантных документов ко всем документам коллекции. В качестве нее можно брать например $log(\frac{N}{n})$, где n $-$ число релевантных документов, а N $-$ число всех документов. Итоговая формула для определения релевантности получается перемножением TF$\times$IDF, что в примере равно $\frac{w}{W}log(\frac{N}{n})$.

2. Методы сбора данных

Так как написание своего собственного поискового паука не является задачей этой работы, то используется паук широкого назначения от некоммерческой организации CommonCrawl [6]. Он обходит весь Интернет, который позволяет себя индексировать, примерно раз в месяц и загружает результаты на свои коллекции. Потом передает URL на них в открытый доступ. Он был выбран так как очень часто новостные страницы используют скриптовые языки на своих сайтах. А это значит что их очень сложно обходить и выгружать с них сами тексты новостей. Также на новостных страницах часто бывают дополнительные новости бегущей строкой, реклама по сторонам от текста или невидимые обычному посетителю сайта элементы регистрации. Также в данной работе не критична регулярность обновления коллекций новостей в 1 месяц.

CommonCrawl позволяет искать страницы по определенным доменам и URL. Так, вводя в специальном интерфейсе домен «$.ru$» вам будут даны ссылки на скачивание всех сайтов у которых доменом является «$.ru$». Или, например, введя в нем же «$https://www.kp.ru/$» вы получите ссылки на архивы для скачивания всей информации находящейся на этом сайте. Также можно указывать метаинформацию (например искать только страницы у которых в структуре хранения текстов в коллекции CommonCrawl указано «rus»), тем самым сужая область поиска.

3. Структура хранения данных

Как было сказано в введении базы данных бывают разных типов. В нашей конкретной задаче было достаточно обычной реляционной базы данных. Так как объемы информации не настолько большие чтобы выделять под это отдельный сервер и частота запросов невелика, было решено остановиться на базе данных Sqlite.

Загрузка информации происходит в несколько этапов:

1. Собираем список нужных новостных изданий.
2. Запись списка URL в базу данных в таблицу с именем queue колонку url.
3. Дальше в несколько процессов запускается скрипт загрузчика.
4. Загрузчик берет каждую запись из очереди.
5. При помощи API CommonCrawl получает ссылки на архивы для скачивания.
6. Пакетами, которые позволяет хранящий ресурс, данные загружаются в таблицу content.
7. Текст отправляется в колонку raw_cont, а адрес страницы в колонку url

Проиллюстрировать работу программы по загрузке и обработке информации можно следующей диаграммой [Рис. 1]

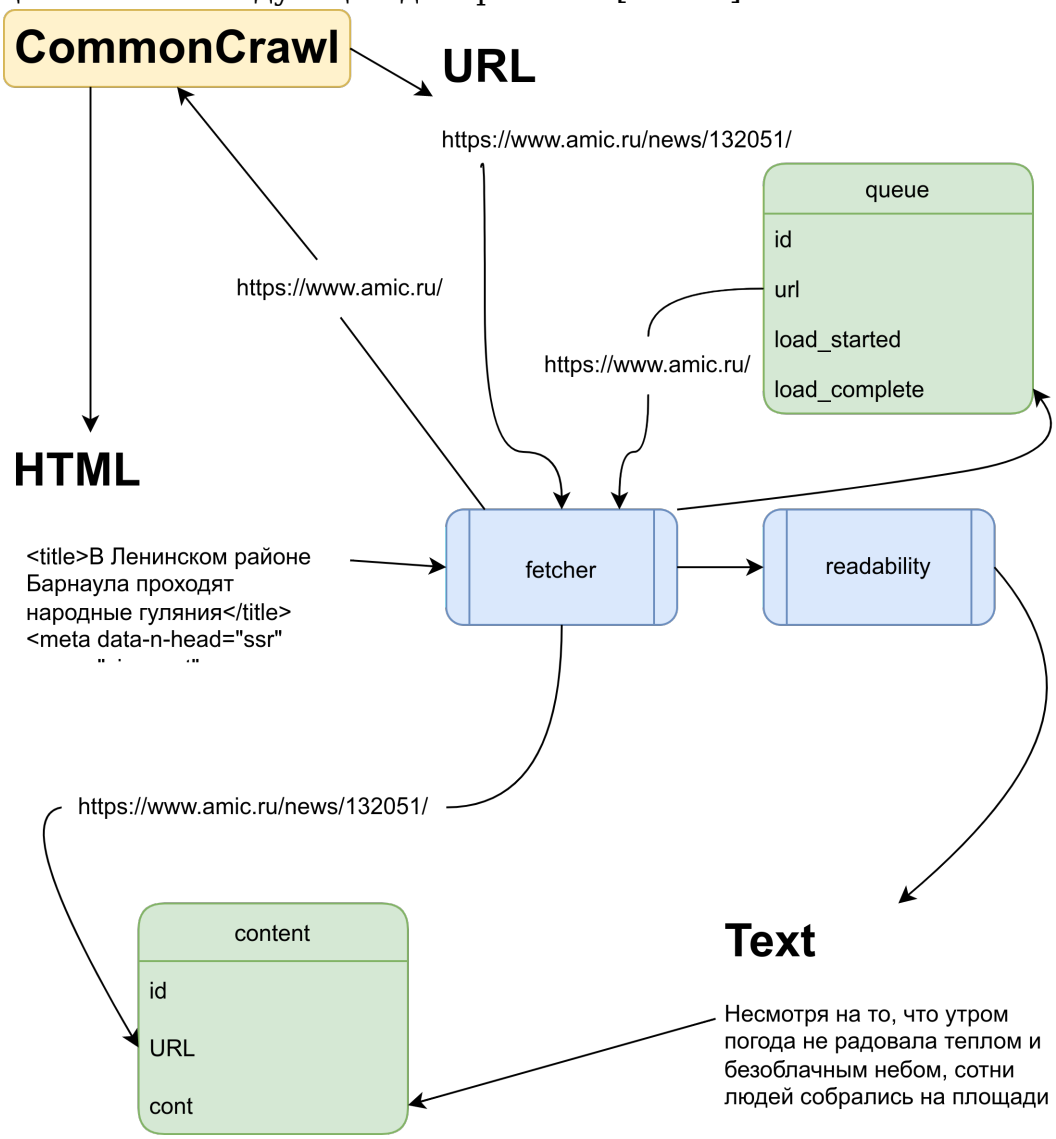

Рис. 1: Иллюстрация работы загрузчика

3.1. Структура таблиц

```
TABLE queue(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    url TEXT,
    load_started DEFAULT NULL,
    load_complete DEFAULT NULL
);
TABLE content(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    url TEXT,
    cont TEXT
);
```

Всего было выгружено 5Гб очищенных данных. Загрузка производилась в несколько процессов. Данные очищались на этапе получения от различных HTML тегов при помощи библиотеки BeautifulSoup и проекта readability в языке программирования Python.

3.2. Математическая модель структуры данных

Основана на реляционной алгебре. Основным элементом являются мультимножества.

1. Отношение – это таблица или ее подмножество.
2. Кортеж – строка в талице.
3. Атрибут – столбец в таблице.
4. Поле – ячейка таблицы определенного типа.
5. Тип ячейки в таблице – некоторое число $a \in \mathbb{N}$, где $\mathbb{N}$ – множество натуральных чисел.

   Соответствие типов:

   (a) t_1 – целое число
   (b) t_2 – текст (последовательность символов)
   (c) t_3 – булево значение (0 – Ложь, 1 – Истина)
   (d) t_4 – время

6. Множество уникальных элементов $U(name_1(t_1), ..., name_2(t_k))$ – набор уникальных кортежей, где у каждого кортежа есть k атрибутов с именами $name_1, ..., name_2$ соответствующими типам $t_1, ..., t_k$. Обозначение выборки определенных кортежей с атрибутами $name_n, ..., name_k$, у некоторых из которых задано целевое значение (например $name_t \in [10, 20]$, а $name_s \in \{\text{Саша}, \text{Маша}, \text{Женя}\}$), будет выглядеть так: $\pi(U(name_n, ..., name_k))_{\sigma(name_t \in [10,20], \quad name_s \in \{\text{Саша,Маша,Женя}\})}$

7. Мультимножество ($M(U(t_1, ..., t_k))$) построенное на множестве уникальных элементов $U(t_1, ..., t_k)$ – это упорядоченная пара (U, φ) такая, что $\varphi : U \to \mathbb{N}$, где $\mathbb{N}$ – множество натуральных чисел, отображение $\varphi : \forall x \in U \; \exists y \in \mathbb{N} : \varphi(x) = y$. (дальше множество уникальных элементов будет опускаться)

В нашем случае имеется 3 мультимножества:

1. $queue(id(t_1), url(t_2), load\_started(t_4), load\_complete(t_4))$ – содержит в себе набор из элементов. Служит для хранения очереди используемых в поиске новостных изданий. Каждый элемент представляет собой кортеж из 4 полей:
   - id – уникальный номер для каждого элемента
   - url – URL новостного ресурса
   - $load\_started$ – время начала загрузки записей новостного ресурса
   - $load\_complete$ – время окончания загрузки записей новостного ресурса
2. $content(id(t_1), url(t_2), cont(t_2))$ – содержит набор полученных из новостных изданий очищенных текстов новостей, где каждая запись является кортежем из 3 полей:
   - id – уникальный номер для каждого элемента
   - url – URL новости
   - $cont$ – чистый текст новости содержащийся по этому URL

Алгоритм сбора данных в этой модели будет определяться следующими соотношениями. Операция выбора из таблицы $queue$ записи с $id = 1$ выглядит так $\pi(queue)_{\sigma(id=1)}$. Общее количество элементов в мультимножестве set с атрибутом id можно определить функцией $len : set \to \mathbb{N}$. Она определяется так $len(set) = \#\{(id \neq NULL) \in set\}$. Первым шагом алгоритма будет выбор следующего новостного ресурса $(current\_id, current\_url) =$

$\pi(queue(id, url))_{\sigma(id \in \min\limits_{id}(\pi(queue(id))_{\sigma(load\_started=NULL)}))}$. По окончании обработки по исходному id будет записана дата окончания обработки новостного ресурса

$$\pi(queue(load\_complete))_{\sigma(id=current\_id)} = current\_date.$$

Полученный $current\_url$ же передается при помощи Интернет соединения на специальный интерфейс CommonCrawl, который возвращает последовательность URL для обхода $url_1, url_2, ...url_m$. Для каждого элемента этой последовательности загружаются сжатые архивы, в которых хранятся нужные страницы. В итоге получается последовательность страниц с HTML-тегами $p_1, p_2, ...p_f$. Полученная последовательность отправляется на обработку специальной программе которая отдает пары $(url_1, cont_1), (url_2, cont_2), ..., (url_n, cont_n)$. То есть можно написать отображения

$$get\_url(current\_url) := (url_1, url_2, ..., url_n)$$

и

$$get\_text(current\_url) := (cont_1, cont_2, ..., cont_n).$$

Полученные записи заносятся в мультимножество $content$ следующим образом

$$\pi(content(id, url, cont))_{\sigma(id=NULL)} =$$

$$= ((last\_id + 1, last\_id + 2, ..., last\_id + n), get\_url(current\_url), get\_text(current\_url)),$$

где $last\_id = len(content)$.

3.3. Замечания о реализации

Скорость работы загрузчика в один процесс составляет около 50 новостей в минуту. Ее можно было повысить до 200 записей в минуту за счет работы 4 fetcher скриптов совместно. Дальнейшее увеличение количества работающих задач не приносило успеха, так как СУБД SQLite не позволяет одновременно писать в таблицу больше чем одному процессу. До 4 включительно так как из-за сетевых задержек это не оказывало влияния и также из-за ограниченности в вычислительных мощностях. Для дальнейшего повышения скорости загрузки, если таковая потребуется, нужно будет перейти на более промышленную базу данных и переписать fetcher на асинхронное выполнение. Были загружены базы данных на 100, 200 тысяч и 1 миллион записей, это соответствует 356 мегабайтам, 1 гигабайту и 5 гигабайтам занятого дискового пространства. При этом из исходной очереди обработки новостных изданий длиной в 200 записей проходилось только 10 для 100 000 записей и примерно столько же для 1 гигабайта. Для базы на 1 000 000

записей было решено обходить по странице с каждого новостного издания, чтобы получать новости с 200 новостных изданий. Сырые тексты, которые загружались с открытых быз данных CommonCrawl, обрабатывались открытым проектом readability, который к сожалению помимо плюсов – очистка текста, убирание рекламы, имел и неприятный минус – пропуск заголовка новости. В новостях по нему обычно можно сделать вывод о большей части содержания статьи, поэтому его исправление желательно в дальнейшей работе.

4. Поиск упоминаний персон

Как и было указано ранее весь поиск конкретной информации в тексте можно разбить на несколько групп в зависимости от того какой способ нормализации мы используем. Существуют разные способы нормализации текстов. На их основе существуют различные способы поиска - статистическая, усекающая и смешанная. Также можно использовать регулярные выражения.

4.1. Определения

1. Полнотекстовый поиск – поиск который ведется по всему документу или по существенной его части.
2. Стемминг – процесс редуцирования слова до его основы или корня.
3. Лемматизация – приведение словоформы к начальной форме.
4. Начальная форма слова – единственная форма для каждого слова. В русском языке для существительных это единственное число, именительный падеж, глаголов – инфинитив, прилагательные – мужской род, единственное число и так далее.
5. Нормализация слова – приведение слова к его начальной форме.

4.2. Описание модели

В данной работе используется стемминг и на нем строится усекающая модель нормализации текстов, так как нет доступа к нужными вычислительными ресурсами для применения остальных.

Также так как было выгружено около 100 000 различных записей их нужно правильно индексировать и искать по ним, так как пройти 100 000 последовательностей из более чем 10 000 символов программой на языке Python довольно долго. Это связано с тем, что Python – интерпретируемый язык

программирования с динамической типизацией, поэтому код на этом языке, как правило, проигрывает в производительности аналогичному коду на языках программирования C/C++ как минимум на порядок (в 10 раз). Поэтому используем язык Python с подключенной к нему СУБД SQLite и прямые SQL запросы. Для полнотекстового поиска используем расширение СУБД FTS5 [4], это специальный набор команд для работы с виртуальными таблицами в СУБД SQLite и поддерживающий полнотекстовый поиск. При помощи нее создается B-дерево всех документов. Так получается кратно ускорить работу полнотекстового поиска по базе данных. Для самого полнотекстового поиска используется отбрасывание окончаний у имени, фамилии и отчества в итоге оставшаяся основа слова при помощи регулярного выражения прогоняется по всей базе текстов.

На втором этапе уже составляется таблица пересечений людей в одном тексте (то есть в одном тексте встретились 2 фамилии из базы данных сотрудников). Дальше уже на оставшихся текстах происходит прогон на наличие в тексте также соответствующих имен и отчеств. Этот этап позволяет избавиться от таких совпадений как для фамилии Орехов и слова орехи, фамилии Толстой и слова толстой. Поиск осуществляется не по всему тексту, а только по его небольшому интервалу вокруг предполагаемо найденной фамилии. В работе используется интервал от 50 до 100 символов, так как существуют двойные фамилии а также длинные южные.

Проиллюстрировать работу программы по поиску упоминаний людей в тексте можно следующей диаграммой [Рис. 2]

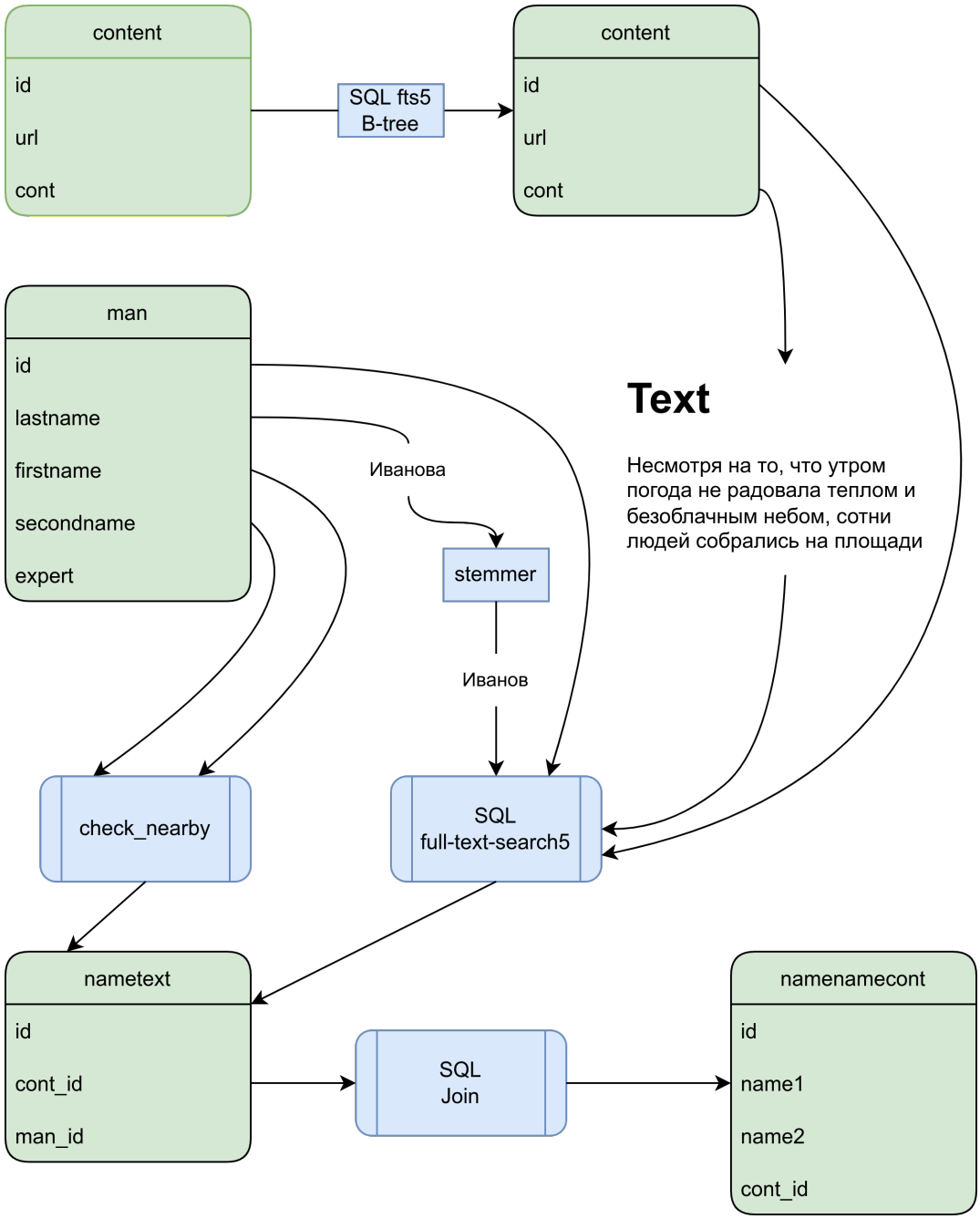

Рис. 2: Работа поиска

4.3. Структура таблиц

```
CREATE TABLE man(id primary key, lastname, firstname, middlename, expert);
CREATE TABLE content(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    url TEXT,
    cont TEXT
);
```

```
CREATE TABLE nametext (id integer primary key AUTOINCREMENT,
cont_id int, name_id int);
CREATE TABLE namenamecont (id integer PRIMARY KEY AUTOINCREMENT,
name1 int, name2 int, cont_id int);
```

4.4. Математическая модель поиска

Пусть у нас есть следующие мультимножества:

1. $man(id(t_1), lastname(t_2), firstname(t_2), middlename(t_2), expert(t_3))$ – содержит набор фамилий имен и отчеств сотрудников, где каждая запись является кортежем из 5 полей:
   - id – уникальный номер для каждого кортежа
   - $lastname$ – фамилия сотрудника
   - $firstname$ – имя сотрудника
   - $middlename$ – отчество сотрудника
   - $expert$ – пометка о том, является ли данный сотрудник экспертом. Эксперт это сотрудник, который может быть выбран для написания рецензии
2. $content(id(t_1), url(t_2), cont(t_2))$ – содержит набор полученных из новостных изданий очищенных текстов новостей, где каждая запись является кортежем из 3 полей:
   - id – уникальный номер для каждого кортежа
   - url – URL новости
   - $cont$ – чистый текст новости содержащийся по этому URL
3. $nametext(id(t_1), cont\_id(t_1), name\_id(t_1))$ – содержит набор соответствий имен тем текстам в которых они упоминаются. Каждая запись это кортеж из 3 полей:
   - id – уникальный номер для каждого кортежа
   - $cont\_id$ – номер текста соответствующий id этого текста в мультимножестве $content$
   - $name\_id$ – номер id записи о сотруднике в мультимножестве man
4. $namenamecont(id(t_1), name1(t_1), name2(t_1), cont\_id(t_1))$ – содержит набор текстов в которых упоминается хотя бы 2 человека из таблицы man. Каждая запись является кортежем из 4 полей:
   - id – уникальный номер для каждого кортежа

- $name1$ – номер записи о сотруднике в мультимножестве man, который упоминается в этом тексте
- $name2$ – номер записи о сотруднике в мультимножестве man, который упоминается в этом тексте
- $cont\_id$ – номер текста в котором упоминаются сотрудники совместно равный id текста в мультимножестве $content$

Тогда математическая модель поиска кортежей из мультимножества

$$man(id(t_1), lastname(t_2), firstname(t_2), middlename(t_2), expert(t_3))$$

в мультимножестве

$$content(id(t_1), url(t_2), cont(t_2))$$

будет являться отображением $s : man \to \{0,1\}^n$, где $n = \#\{id \in man\}$. Дальше опускаем указание типов, так как они остаются неизменными. Функцию стемминга обозначим:

$$stem(a_1...a_n) = \begin{cases} a_1...a_n, \text{ если } a_n \neq "a" \\ a_1...a_{n-1}, \text{ иначе} \end{cases}$$

Тогда узнать есть ли упоминание человека $\pi(man)_{\sigma(id=t)}$ в таблице текстов $content$ можно так

$$s_1(\pi(man)_{\sigma(id=t)}, content) :=$$

$$\{(t, \quad H(\#\bigcup_{g=1}^{\#\{id \in content\}} \pi(content)_{\sigma(stem(man(id=t,lastname)) \in content(id=g,cont))})), \},$$

где H сдвинутая функция Хевисайда:

$$H : \mathbb{N} \to \{0,1\}$$

$$H(n) = \begin{cases} 0, & n < 1 \\ 1, & n \geq 1 \end{cases}$$

Узнать же записи в которых упоминается человек можно следующим образом:

$$s_0(\pi(man)_{\sigma(id=t)}, content) :=$$

$$\{(t, \quad \bigcup_{g=1}^{\#\{id \in content\}} \pi(content(id))_{\sigma(stem(man(id=t,lastname)) \in content(id=g,cont))}), \}$$

Если будет искаться несколько элементов из таблицы, то ответом будут множества размера в количество разыскиваемых элементов. Каждый элемент будет являться упорядоченной парой с первым аргументом номером

id, а вторым найденными значениями. Эти данные и заносятся в таблицу $nametext$.

Результатом же заносимым в таблицу $namenamecont$ будет пересечение $s_0(\pi(man)_{\sigma(id=t)}, content)$ и $s_0(\pi(man)_{\sigma(id=j)}, content)$ по всем $j \neq t$.

Чтобы найти позицию элемента $a_1...a_n$ в тексте $text$ будем использовать $fts(a_1...a_n, text)$. Тогда узнать позицию слова в тексте можно так:

$$s_2(\pi(man)_{\sigma(id=t)}, content) :=$$

$$\{(t, \bigcup_{g=1}^{\#\{id \in content\}} fts(stem(man(id = t, lastname)),$$

$$\pi(content(cont))_{\sigma(stem(man(id=t,lastname)) \in content(id=g,cont))})), \}$$

4.5. Замечания о реализации

Получившаяся математическая модель была реализована на языков программирования Python и SQL. Полученная программа была проверена для 100 000 текстов. В результате остается примерно 1% текстов, в которых упоминается хотя бы один сотрудник, и 0.05% – где упоминаются хотя бы 2 сотрудника. Если ограничить стемминг только фамилией, тем самым увеличив полноту поиска, то количество текстов возрастает, но падает его качество. Возникают срабатывания на фамилию Орехов и части предложения в духе «Ореховые волокна обладают свойством связывать ...» или фамилии Толстой и «... вызывает опухоли в толстой кишке ...». Также есть проблемы своевременного обновления базы людей, так в базе данных не было данных о докторе химических наук, сотруднике РАН Иване Смирнове, но при этом была новость где он упоминался именно в этом ключе. При этом добавление требования совпадения имени тоже не до конца решает задачу. В базе данных есть сотрудник Иван Смирнов и при этом есть текст «Владимир Смирнов лишился должности в результате авиаколлапса...» в котором говорится о бывшем руководителе авиационной компании, который не был сотрудником РАН. Время составления таблицы соответствий текстов упоминаемых в ним именам для 100 000 текстов и 1400 сотрудников составляет 35 минут. Построение же итоговой таблицы с текстами где упоминается хотя бы 2 человека занимает ≈ 2.18 секунд. Сама таблица при этом содержит 178 тысяч записей.

5. Заключение

В данной работе построена математическая модель графа связей сотрудников научных организаций, создана структура хранения данных, реализованы программы выполняющие регулярное сохранение на локальные хранилища данных с новостных изданий, поиск среди полученных текстов упоминаний людей и выявление текстов в которых потенциально может возникнуть конфликт интересов. Все представленные в работе методы обработки текстов и поиска упоминаний находятся на зачаточном уровне и требуют доработки и улучшения. С целью повышения качества поиска, за счет лучшего понимания того, является ли упоминаемый в тексте человек сотрудником из нашей базы данных или нет, в будущем планируется расширение типов информации о каждом конкретном сотруднике. Это планируется сделать путем добавления мест работы/рождения/проживания/обучения и предыдущих мест работы. Потенциальное развитие работы представляется в повышении скорости загрузчика и объема данных на основе которых строится индекс. Также видится возможность развития при помощи добавления триплетов и более длинных n-плетов. На основе которых при помощи онтологий или моделей машинного обучения будут строиться логические заключения. На текущем этапе кажется, что онтологии построенные по графам знаний, дадут более богатый спектр возможностей для отличения однофамильцев. Так как они позволяют строить базовые логические суждения по данным представленным в виде огромного графа, где есть связи между между связанными понятиями. Например, все доступные данные о человеке будут иметь центральной вершиной некоторый уникальный для него номер. И если уметь строить базовые логические суждения по этим данным то можно довольно легко различать однофамильцев.

Список литературы

1. CAP Twelve Years Later: How the «Rules» Have Changed // InfoQ [Электронный ресурс]. URL: https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/ (дата обращения: 17.05.2022).
2. Parallel Crawlers [Электронный ресурс]. URL: https://ra.ethz.ch/CDstore/www2002/refereed/108/index.html (дата обращения: 05.05.2022).
3. Snowball: A language for stemming algorithms [Электронный ресурс]. URL: http://snowball.tartarus.org/texts/introduction.html (дата обращения: 20.05.2022).

4. SQLite FTS5 Extension [Электронный ресурс]. URL: https://www.sqlite.org/fts5.html (дата обращения: 22.05.2022).
5. Wikidata [Электронный ресурс]. URL: https://www.wikidata.org/wiki/Wikidata:Main_Page (дата обращения: 21.05.2022).
6. So you’re ready to get started. – Common Crawl [Электронный ресурс]. URL: https://commoncrawl.org/the-data/get-started/ (дата обращения: 21.05.2022).
7. AbuKausar Md., S. Dhaka V., Kumar Singh S. Web Crawler: A Review // International Journal of Computer Applications. 2013. № 2 (63). С. 31–36.
8. Balakrishnan V., Ethel L.-Y. Stemming and Lemmatization: A Comparison of Retrieval Performances // Lecture Notes on Software Engineering. 2014. № 3 (2). С. 262–267.
9. Batsakis S., Petrakis E. G. M., Milios E. Improving the performance of focused web crawlers // Data & Knowledge Engineering. 2009. № 10 (68). С. 1001–1013.
10. Boldi P. [и др.]. UbiCrawler: a scalable fully distributed Web crawler // Software: Practice and Experience. 2004. № 8 (34). С. 711–726.
11. Brin S., Page L. The anatomy of a large-scale hypertextual Web search engine // Computer Networks and ISDN Systems. 1998. № 1 (30). С. 107–117.
12. Chang F. [и др.]. Bigtable: A Distributed Storage System for Structured Data // ACM Transactions on Computer Systems. 2008. № 2 (26). С. 1–26.
13. Cho J., Garcia-Molina H. The Evolution of the Web and Implications for an Incremental Crawler С. 18.
14. Färber M., Rettinger A. Which Knowledge Graph Is Best for Me? // 2018.
15. Grolinger K. [и др.]. Data management in cloud environments: NoSQL and NewSQL data stores // Journal of Cloud Computing: Advances, Systems and Applications. 2013. № 1 (2). С. 22.
16. Hogan A. [и др.]. Knowledge Graphs // ACM Computing Surveys. 2022. № 4 (54). С. 1–37.
17. Jivani A. G. A Comparative Study of Stemming Algorithms 2011. (2). С. 9.
18. Melton J. SQL language summary // ACM Computing Surveys. 1996. № 1 (28). С. 141–143.
19. Ramos J. Using TF-IDF to Determine Word Relevance in Document Queries.
20. Salton G., Buckley C. Information processing & management // Term-weighting approaches in automatic text retrieval. 1988. № 5 (24). С. 513–523.
21. Willett P. The Porter stemming algorithm: then and now // Program. 2006. № 3 (40). С. 219–223.
22. Wireless Communication and Computing) student, CSE Department, G.H. Raisoni Institute of Engineering and Technology for Women, Nagpur, India [и

др.]. Study of Web Crawler and its Different Types // IOSR Journal of Computer Engineering. 2014. № 1 (16). С. 01–05.
23. Zobel J., Moffat A. Inverted files for text search engines // ACM Computing Surveys. 2006. № 2 (38). С. 6.