РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

УТВЕРЖДАЮ

Зав. Кафедрой Информационных Технологий

к. ф. - м. н. профессор

И. Л. Толмачев

« » \_\_\_\_\_\_\_\_\_\_ 2017 г.

КУРСОВАЯ РАБОТА

на тему

«Поисковые системы»

02.03.02

Фундаментальная информатика и информационные технологии

Выполнил

Студент группы: НИ-102

Студенческий билет №: 10321445988

Винников А. В.

« » \_\_\_\_\_\_\_\_\_\_ 2017 г.

Руководитель

Доцент кафедры Информационных Технологий

к. ф. – м. н. А. Н. Виноградов

Москва 2017

**Оглавление**

1. Список определений...............................................................................................3
2. Введение...................................................................................................................3
3. Цель работы.............................................................................................................3
4. Основные определения для поисковых систем...................................................4
   1. Индексированный поиск.................................................................................4
   2. Нормализация слова. Стем.......................................................................5
   3. Метрика.............................................................................................................6
   4. Классификация источников............................................................................7
5. Поисковая система.................................................................................................8
   1. Структура поисковой системы.......................................................................8
   2. Реализация поисковой системы......................................................................8
      1. Основная программа...............................................................................8
      2. Управляющие команды..........................................................................9
         1. *Classes*..........................................................................................10
         2. *Classify*.........................................................................................10
6. Заключение...........................................................................................................11
7. Список литературы..............................................................................................11
8. Список определений

[N] – Номер ссылки на список литературы, представленный в конце работы.

Источник – документы, страницы или файлы в которых производится поиск.

Поисковый запрос (или просто запрос) – строка, вводимая пользователем, предназначенная для поиска. [4]

Результат – список найденных по запросу результатов, представленных в определенном формате.

Сниппет – фрагмент текста, который наиболее точно характеризует источник, и степень соответствия его к поисковому запросу. [2]

Окружение – В данном контексте, некоторое количество предыдущих и последующих слов, соседствующих с найденной из запроса строкой.

Оригинальное слово – некое слово, соответствующее множеству нормализованных слов из всех источников. [4]

1. Введение

Поисковая система – это система, предназначенная для поиска определенной информации в большом массиве данных. Самым известным примером поисковых систем являются поисковые веб-сервисы google.com, yandex.ru и т.д. Однако поиск веб-страниц далеко не основное предназначение для поисковых систем. В этой работе информация, которую будет искать система, представляет собой текст, а источником будут являться текстовые файлы. [1]

1. Цель работы

Написать программу для поиска фрагментов текста во множестве текстовых файлов, оптимизировать и описать ее работу.

1. Основные определения для поисковых систем
   1. Индексированный поиск

Одним из основных способов поиска информации, использующиеся практически во всех современных поисковых системах, является индексированный поиск. Он представляет собой словарь оригинальных слов из всех имеющихся источников. Выделяют два типа индекса: прямой и обратный. [3]

При прямом индексе создается словарь источников, в каждом из которых прописывают список оригинальных слов и их местоположения в тексте. [3]

При обратном(инвертированном) индексе создается словарь всех оригинальных слов из всех источников, к которым прописывают название источника и список местоположений этого слова в нем. [3]

Очевидно, что при использовании обратного индекса поиск будет осуществляться намного быстрее, при одинаковом количестве используемой памяти, однако, современные поисковые системы используют оба этих индекса. С помощью обратного индекса находится местоположения одного или нескольких наиболее значимых слов из запроса, тогда как прямой индекс необходим для поиска наиболее релевантного окружения слов найденного результата и создание подходящего сниппета.

В итоге, используя только метод индексированного поиска, можно уже получить работающую систему поиска, способную находить все вхождения слов из запроса в источниках, но не способную хоть как-то ранжировать эти результаты по релевантности.

В своей программе я использую аналогичную систему индексированного поиска.

* 1. Нормализация слова. Стем

Очевидно, что для использования индексированного поиска необходимо определить операцию нормализации каждого слова. Первое что может прийти на ум – это морфологический анализ. Понятно, что в отношении качества нормализации, это почти идеальный вариант. Морфологический анализ способен привести к нормальному виду любые «видоизмененные» слова (с добавлением приставок, суффиксов и, в отдельных случаях, даже слова с переменной корневой частью). Однако сложность алгоритма, большая скорость обработки и добавление огромных дополнительных словарей делает интеграцию морфологического анализа в небольшую поисковую систему неэффективной и, как оказывается, не необходимой. [3]

Более упрощенным методом нормализации слова является метод стемминга. Стемминг – это алгоритм укорачивания слова при помощи «отрезания» от него лишних частей окончаний и суффиксов (при этом стем никак не влияет на приставки). Он не только выигрывает у морфологического анализа отсутствием необходимости добавлять огромные словари, но и намного быстрее в обработке. Алгоритм стема прост и находится в свободном доступе. Для русскоязычной реализации я использовал алгоритм стемминга Портьера(1)\*. [5]

Однако, основным, и крайне неприятным минусом стеммирования являются ошибки которые он допускает при обработке. Самой распространённой ошибкой является идентичность стема совершенно разных слов. Сама по себе эта ошибка не так страшна, но при ранжировании результатов поиска, стеммирование слов собственных, которые, по идее, должны быть с наиболее высоким tf-idf могут совпадать со стемом каких-либо общеупотребимых слов, tf-idf которых крайне низок. В итоге, за счет этого мы получаем снижение эффекта ранжирования.

* 1. Метрика

Метрика — это способ задания всем результатам поиска некоторых коэффициентов (далее - рангов), которые будут соответствовать их релевантности относительно остальных результатов, и по которым они в дальнейшем будут ранжироваться. [3]

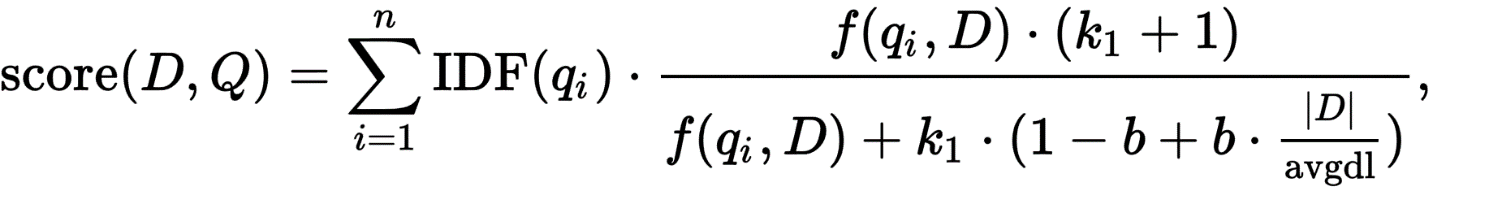
Существует огромное множество метрик, как по свойству местоположения слова в источнике (расстояния между словами из запроса, близость к концу или началу файла, близость к наиболее оригинальным словам и т.д.) так и по количеству слов в искомом и окружающих источниках (количество слов в тексте в целом, tf-idf и все его разновидности).[3]

Можно до бесконечности наращивать в поисковой системе все больше и больше самых разных метрик, но в итоге, по причине взаимоисключающих свойств отдельных метрик, можно прейти к необоснованно долгому, и совершенно не эффективному поиску.

В моей программе используется гибрид из двух основных методов задания метрики. Сначала результаты безусловно сортируются по количеству вхождений слов из запроса. Затем формируется ранг результата по следующим принципам:

1. Основу ранга составляет tf-idf всего результата (вместе с окружением).
2. Далее, чем ближе слова из запроса стоят друг к другу тем больше увеличиваем ранг.
3. Если у основного найденного слова в результате низкий tf-idf, ранг становится меньше.
4. Если в результат входит всего одно слово из запроса с высоким tf-idf (либо у остальных слов из запроса tf-idf равен нулю) ранг результата так-же повышается.

В своей программе я реализовывал такой тип tf-idf как Okapi BM25. [3]



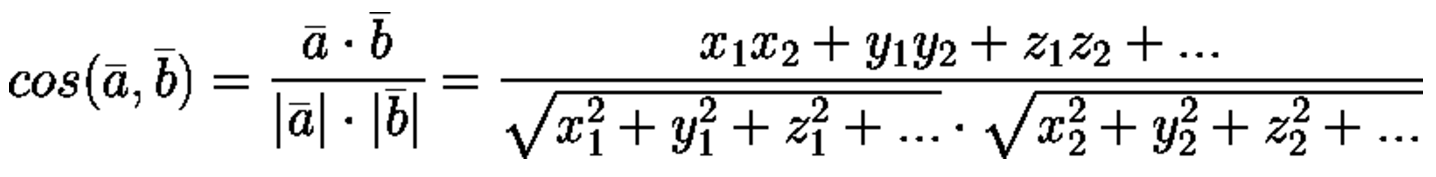
Однако на практике этот метод ранжирование эффективен только для огромного количества источников, поэтому мне пришлось использовать нечто среднее между Okapi BM25 и обычной формулой tf-idf.

* 1. Классификация источников

Сама по себе классификация источников не является целью поисковой системы. Однако добавление tf-idf открывает перед этой системой некоторые дополнительные возможности. В контексте моей программы, классификация источников, это всего лишь степень «похожести» двух отдельно взятых источников. Что в последствии дает возможность создавать некие абстрактные классы по которым эти источники могут разделяться. [3]

В теории, классификация источников может служить для поисковой системы дополнительным критерием ранжирования. То есть она не будет влиять непосредственно на ранг, но будет учитываться в процессе сортировки. Опять же это дает результат только при большом количестве источников, поэтому для своей программы я выбрал более нейтральное назначение для классификации, а именно – использование фильтра поиска по классам.

В своей поисковой системе я использую векторную модель для классификации файлов. Составляется вектор значений tf-idf всех слов, соответствующий каждому источнику. При этом если слово не входит в источник, его tf-idf, соответственно, равен нулю. Затем находится косинус между всеми векторами, значение которого равно скалярному произведению векторов. [3]



Результатом и будет некий коэффициент сходства двух источников относительно всех остальных.

1. Поисковая система

5.1. Структура поисковой системы

Итоговая программа содержит всего три ресурсных файла.

stem.cpp – это реализация стемминга которая описывает всего одну функцию – stem. Эта функция получает на вход переменную типа string и стеммирует ее.

Linker.cpp – это основной код программы, в нем описаны все действия по составлению словаря, поиск, а так-же пользовательский интерфейс.

Extended.cpp – файл с реализацией дополнительных управляющих программ, таких как: Классификация файлов, фильтр по классам, информация о файлах и т.д.

5.2. Реализация поисковой системы

5.2.1. Основная программа

Перед началом работы с программой необходимо добавить файлы в формате .txt в которых будет осуществятся поиск. Затем их названия необходимо вписать в файл Input.txt.

При запуске, программа дает нам выбрать – использовать все имеющиеся файлы, или выбрать из них те, которые программа будет использовать. Оба этих выбора для удобства были реализованы с помощью псевдографического консольного интерфейса со стрелочным управлением.

Важное уточнение, далее в работе вместо термина «источник» будет использоваться термин «файл» т.к. назначение именно этой программы - производить поиск по текстовым файлам и находить отдельные фрагменты текста.

Далее идет процесс создания словаря. На этом этапе программа читает все файлы, создает на их основе словарь с обратным и прямым индексом, список оригинальных нормализованных слов, вычисляет tf-idf и создает другую служебную информацию, необходимую ей для работы.

Далее программа выводит сообщение что вывод словаря завершен и можно вводить поисковый запрос. При вводе запрос разбивается по словам, и начинается процесс поиска отдельных слов по обратному индексу, все совпадения заносятся в массив(вектор) и обрабатываются. После этого для каждого результата высчитывается ранг и происходит многоступенчатое ранжирование.

Сортировка результатов происходит в соответствии с ранжированием (т.е. ранг - это не основная мера сортировки). Результат выводится в таком виде:

1. Сниппет
2. Название файла
3. Позиция первого слова
4. Ранг всего сниппета

При этом т.к. программа предназначена для поиска по файлам, нет наиболее подходящего (и файлу и запросу) по смыслу фрагмента текста, поэтому сниппетом является все окружение найденных слов из запроса.

После отображения результата, можно повторно вводить поисковый запрос.

5.2.2 Управляющие команды

На этапе ввода запроса можно так-же ввести управляющие команды. Они характерны тем, что все составлены на английском языке, а так как в программе не реализован поиск по английскому языку, неопределенности быть не может.

Список команд:

*exit* – выход из программы.

*classes* – вывод служебной информации классификации(коэффициента схожести файлов) в виде таблицы.

*classify* – создание абстрактных классов и их сопоставление с файлами. Далее можно настроить фильтр поиска по классам(или ввести “0” для отмены).

*max\_change* – изменение максимально допустимого числа результатов для отображения.

*file-info* – Дополнительная информация отдельно по каждому файлу.

*word\_info -* Дополнительная информация отдельно по каждому оригинальному слову.

*stem –* вывод в стеммированном виде один из результатов поиска, или сам запрос.

5.2.2.1. *Classes*

При вводе команды *classes* составляется матрица схожести файлов на основе векторной модели по словам. Особенность реализации этой классификации в том, что она будет корректно работать только при наличии трех и более файлов. Это происходит по понятной причине, ведь когда вычисляется скалярное умножение векторов по словам, при количестве файлов меньше двух, все слова которые есть у поисковой системы используются в первом же сравнении. По сути это означает что системе классификации просто не с чем сравнивать степень схожести, ведь она вычислят это значение относительно всех остальных файлов.

5.2.2.1. Classify

Команда *classify* использует ровно такой-же метод классификации, как и команда *classes*. Отличия в том, что она не выводит эту служебную информацию, а использует ее для создания абстрактных классов, которые смогут разделять файлы по смыслу.

В своей программе, для определение соответствия файла какому-либо классу, я использую пороговое значение. Изначально (и так наверно правильнее всего) этот «порог» высчитывался по замысловатой формуле, но в последствии, наблюдая в процессе тестирования за тем что он оставался примерно в одинаковом диапазоне, я заменил формулу на константу. Это можно очень легко объяснить тем, что при слишком малом количестве файлов, степень классификации имеет довольно мало практического значение, а значит им можно пренебречь. Классификация необходима при большом количестве файлов, однако, так как программа не сможет обработать слишком много файлов, пороговое значение всегда будет примерно на одном уровне.

6. Заключение

В ходе работы была создана программа для поиска фрагментов текста из текстовых файлов. Работа этой программы была всячески модернизирована и оптимизирована. Наибольший приоритет оптимизации был отдан скорости работы поиска и обработки, а не количеству используемой памяти. И хотя у нас есть полностью работающая программа, способная обрабатывать достаточное количество файлов, уровень ее реализации крайне далек от современных поисковых систем, так как компании, работающие в этом направлении несколько десятков лет, смогли добиться куда большой эффективности работы, а все их методы скрыты коммерческой тайной.

7. Список литературы

[1] – Сайт procomputer.su статья «Поисковые системы»

Ссылка: <http://procomputer.su/osnovy-interneta/54-poiskovaya-sistema>

[2] – Информационный портал <http://ktonanovenkogo.ru> статья «Как работают поисковые системы»

Ссылка: <http://ktonanovenkogo.ru/seo/search/kak-rabotayut-poiskovye-sistemy-snippet-index.html#poiskoviki7>

[3] – Интернет=блог <http://vas3k.ru> пост «Введение в поиск и обработку языка»

Ссылка: <http://vas3k.ru/blog/352/>

[4] – Сайт habrahabr.ru пост «Поисковые технологии или в чем загвоздка написать свой поисковик» Автор @cast

Ссылка: <https://habrahabr.ru/post/123671/>

[5] – Статья «Стеммер Портьера для русского языка» Автор Pavel Perestoronin

Ссылка: https://medium.com/@eigenein/%D1%81%D1%82%D0%B5%D0%BC%D0%BC%D0%B5%D1%80-%D0%BF%D0%BE%D1%80%D1%82%D0%B5%D1%80%D0%B0-%D0%B4%D0%BB%D1%8F-%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%BE%D0%B3%D0%BE-%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0-d41c38b2d340