

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ
по дисциплине «Разработка ПО информационных систем»
Тема: Построение графа связности текста

Студенты гр. 6304

Корытов П.В.

Пискунов Я.А.

Цыганов М.А.

Преподаватель

Заславский М.М.

Санкт-Петербург

2019

ЗАДАНИЕ

Студенты

Корытов П.В.

Пискунов Я.А.

Цыганов М.А.

Группа 6304

Тема проекта: построение графа связности текста

Исходные данные:

Необходимо реализовать приложение для построения графа связности текста с возможностью визуализации данного графа и реализации на нем некоторых алгоритмов.

Содержание пояснительной записки:

«Оглавление»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Заключение»

«Список использованных источников»

«Приложение»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студенты

Корытов П.В.

Пискунов Я.А.

Цыганов М.А.

Преподаватель

Заславский М.М.

АННОТАЦИЯ

В наши дни в мире публикуется множество статей и работ, которые несут практическую пользу. Не секрет, что данные статьи не всегда пишутся добросовестно, но и среди честных авторов могут быть те, кто в силу тех или иных обстоятельств недостаточно хорошо разобрались в теме и не смогли донести аудитории свои мысли. Данный проект призван помочь авторам статей анализировать их собственные творения на предмет связности отдельных частей, чтобы выявить и в будущем избежать структурных ошибок, когда некоторые части статьи полностью повторяют другие или, наоборот, абсолютно не связаны с остальным содержанием публикации. Также программа, полученная в результате выполнения проекта, полезна при анализе данных публикаций на плагиат.

SUMMARY

Nowadays, many articles and works are published in the world that are of practical use. It is no secret that these articles are not always written in good faith, but among honest authors there may be those who, due to certain circumstances, did not understand the topic well enough and could not convey their thoughts to the audience. This project is designed to help the authors of articles analyze their own creations to determine the coherence of individual parts, to identify structural errors in the future, when some parts of the article are completely repeated by others or, on the contrary, are completely unrelated to the rest of the publication. Also, the program obtained as a result of the project may be to some extent useful in the analysis of these publications for plagiarism.

Оглавление

ВВЕДЕНИЕ	7
1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ	8
2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ	9
2.1. Макет UI	9
2.2. Описание сценариев использования	10
3. МОДЕЛЬ ДАННЫХ	11
3.1 . Вершины	11
3.2 . Ребра	11
3.3 . Оценка объема информации, хранимого в модели и сравнение с аналогом, использующим реляционное решение	11
4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ	13
4.1 . Краткое описание	13
4.2 . Схема экранов приложения	13
4.3 . Использованные технологии	18
4.4 . Инструкция по сборке и развертыванию приложения	18
4.5 . Ссылки на приложение	19
ЗАКЛЮЧЕНИЕ	20
Результаты	20
Недостатки и пути для улучшения полученного решения	20
Будущее развитие решения	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ А	22

ВВЕДЕНИЕ

Цель данного проекта – разработать программный продукт для построения графа связности текста и проанализировать полученные результаты с целью предоставления пользователю некоторых статистических данных в упорядоченном виде. В процессе создания продукта также было необходимо познакомиться с принципами работы нереляционных баз данных и использовать полученные знания на практике.

В качестве базы данных для проекта была выбрана СУБД Neo4j, так как она лучше всего оптимизирована для хранения и обработки данных в виде графов. Само приложение было решено сделать на языке Python с использованием инструмента PyQt5 для реализации GUI.

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Необходимо реализовать приложение для построения графа связности текста с возможностью визуализации данного графа и реализации на нем некоторых алгоритмов.

Необходимо предусмотреть возможность выбора способа разбиения текста на фрагменты, импорт и экспорт информации, наиболее эффективное взаимодействие с базой данных и читаемое предоставление результатов анализа конечному пользователю.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

Макет пользовательского интерфейса со встроенным описанием возможных действий пользователя вышел весьма большим по размеру. Его вид представлен на рис. 1 и рис. 2. Для более детального ознакомления можно рассмотреть электронную версию данного макета по ссылке: <https://github.com/moevm/nosql1h19-text-graph/wiki/Макет-интерфейса.>

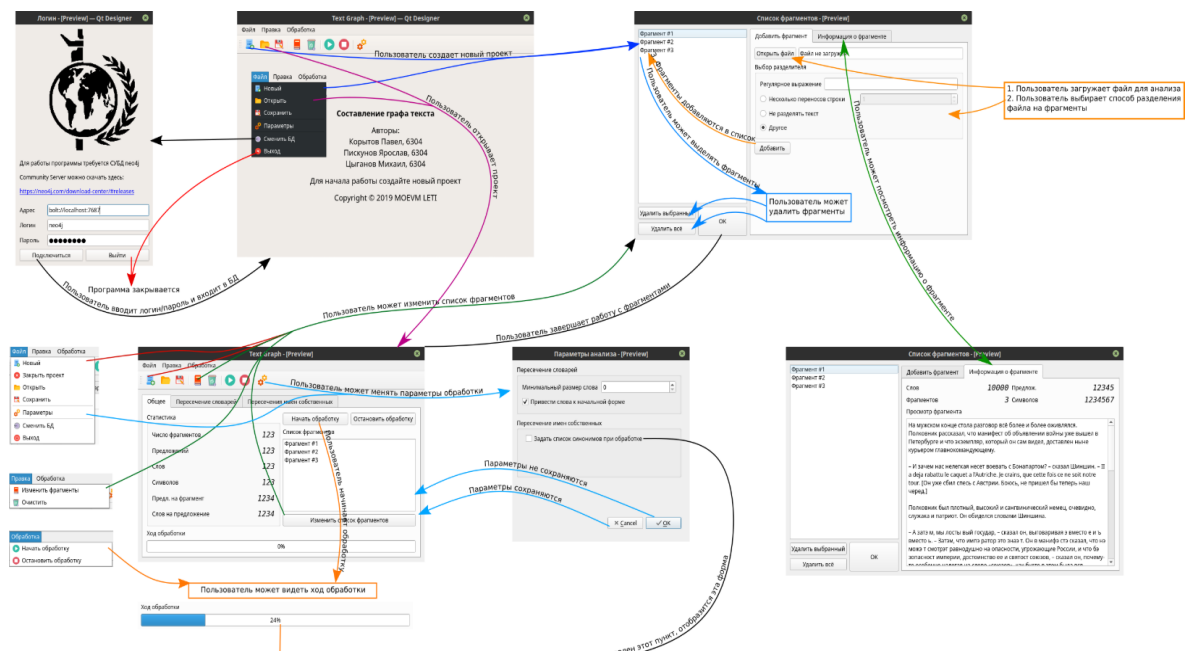


Рисунок 1 – макет окон интерфейса (1)

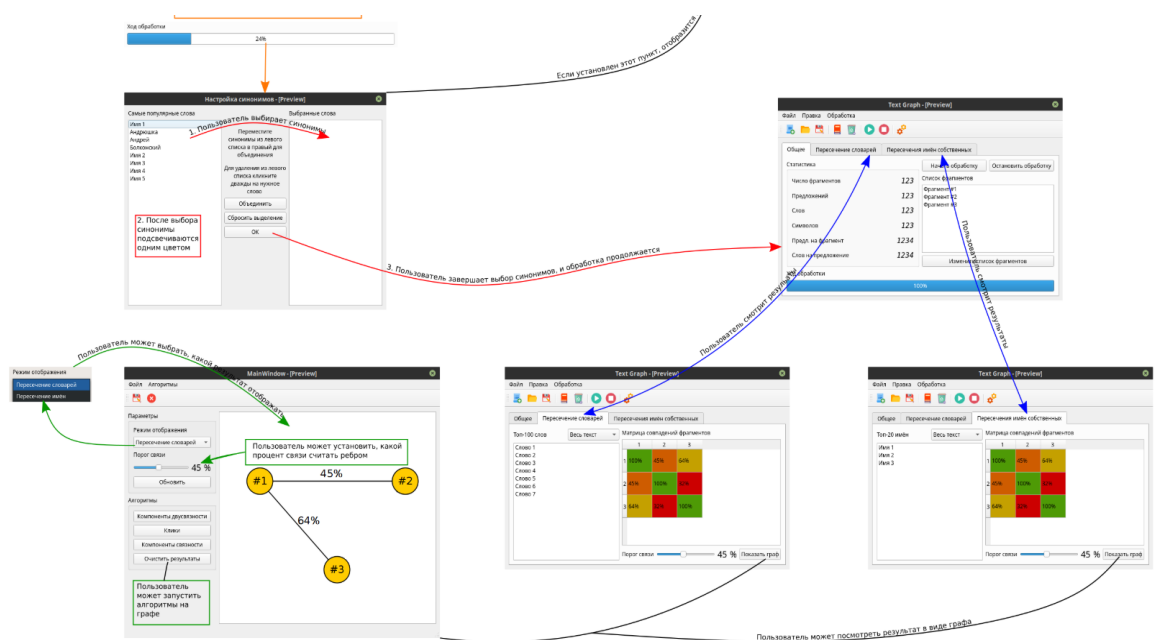


Рисунок 2 – макет окон интерфейса (2)

2.2. Описание сценариев использования

В системе только одно действующее лицо – пользователь. Также имеется один основной сценарий использования, несколько опциональных шагов и альтернативных сценариев. Это представлено ниже.

Действующее лицо: Пользователь

Основной сценарий

1. Пользователь инициирует БД и запускает программу
2. Пользователь создает новый проект
3. Пользователь открывает файл в формате *.txt с художественным произведением
4. Пользователь задает способ разделения текста на фрагменты
5. Пользователь нажимает кнопку "Добавить".
6. Отображается разбиение текста на фрагменты. Пользователь проверяет корректность разбиения
7. Пользователь завершает работу с фрагментами и нажимает "начать обработку"
8. Происходит обработка. По окончании обработки становятся активными вкладки с результатами
9. Пользователь переходит по вкладкам и просматривает результаты
10. Пользователь переходит в режим просмотра результатов в виде графа
11. Пользователь видит, какие фрагменты сильно связаны, взаимодействуя с отображением графа

Опциональные шаги

- Пользователь изменяет настройки перед обработкой. В таком случае возможно появление дополнительных диалогов на этапе обработки (настройка синонимов, [...])
- Пользователь может выйти на каждом шаге
- Пользователь может сохранить результат обработки

Альтернативные сценарии

- К базе данных не подключится => Выводится ошибка
- В программе уже загружен проект => Выводится предупреждение
- Файл не открывается => Выводится ошибка

3. МОДЕЛЬ ДАННЫХ

3.1. Вершины

В вершинах графа хранятся фрагменты текста в нормализованном виде. Так, например, для текста произведения Л.Н. Толстого «Война и мир» можно произвести разбиение на параграфы, главы и части. И, соответственно, в каждой вершине графа будет храниться нормализованный текст одного параграфа, главы или части.

3.2. Ребра

В ребрах хранится информация о связях между фрагментами. В результате работы алгоритмов в ребре сохраняется коэффициент от 0 до 1, обозначающий степень пересечения фрагментов («0» - полностью не совпадают, «1» - абсолютно идентичны), и слова, по которым было обнаружено пересечение. В качестве алгоритмов анализа используется собственный алгоритм пересечения словарей, который выделяет наиболее часто встречающиеся в фрагментах слова и сравнивает эту информацию, и алгоритм сравнения текстов, разработанный компанией Google.

3.3. Оценка объема информации, хранимого в модели и сравнение с аналогом, использующим реляционное решение

Для удобства анализа и написания формул был использован Mathcad. Для более детального ознакомления с результатами анализов можно ознакомиться с файлом по ссылке: https://github.com/moevm/nosql1h19-text-graph/blob/master/docs/Моделирование_БД.pdf или в приложении А. Для примера снова был взят текст произведения Л.Н. Толстого «Война и мир».

В результате анализа можно сделать следующие выводы относительно объема информации:

- При разбиении на параграфы ребра составляют подавляющую часть объема. В этом случае отсекание части связей приводит к почти пропорциональной экономии места.

- В случае разбиения на главы объем БД гораздо меньше. В этом случае текст составляет более двух третей объема, и экономия места при отсечении вершин в несколько раз ниже.

- В случае разбиения на части текст занимает подавляющее большинство объема. Отсечение вершин в данном случае не имеет смысла.

В качестве альтернативного реляционного решения выбрана СУБД MySQL. Занимаемый объем был рассчитан также для случаев разбиения на части, главы и параграфы. Детальная информация об анализе также представлена в приложении А. Там же описано и моделирование запросов в каждом из случаев. В результате сравнения можно сделать следующие выводы:

- Увеличение объема при увеличении числа текстов – $O(n^2)$.
- Отсечение части связей приводит к уменьшению места на $O(\sigma)$, при росте в $O(n^2)$ это не имеет смысла.

- Neo4j тратит немного меньше (на 60,13%) места на хранение данных за счет динамического выделения памяти и отсутствия индексов.

- Разбиение большого текста на небольшие фрагменты нерационально и приводит к резкому возрастанию объема БД.

- При реализации алгоритмов на SQL эффективность может быть ниже в $O(m/n)$ раз. Это существенная разница, так как для больших текстов $m \gg n$. Кроме того, периодически необходимы трудоемкие операции вроде UNION.


4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Реализация представляет собой настольное GUI-приложение, в котором имеется возможность выбрать файл или несколько, разбить его на фрагменты с помощью регулярных выражений и проанализировать связность этих фрагментов, выбрав один из предложенных алгоритмов. Также отображается общая информация по тексту. После имеется возможность сохранить результат.

4.2. Схема экранов приложения

Ниже, на рис. 3 – рис. 12 представлены экраны приложения в порядке появления перед пользователем.



Логин



Для работы программы требуется СУБД neo4j
Community Server можно скачать здесь:
<https://neo4j.com/download-center/#releases>

Адрес

Логин

Пароль

Рисунок 3 – экран авторизации

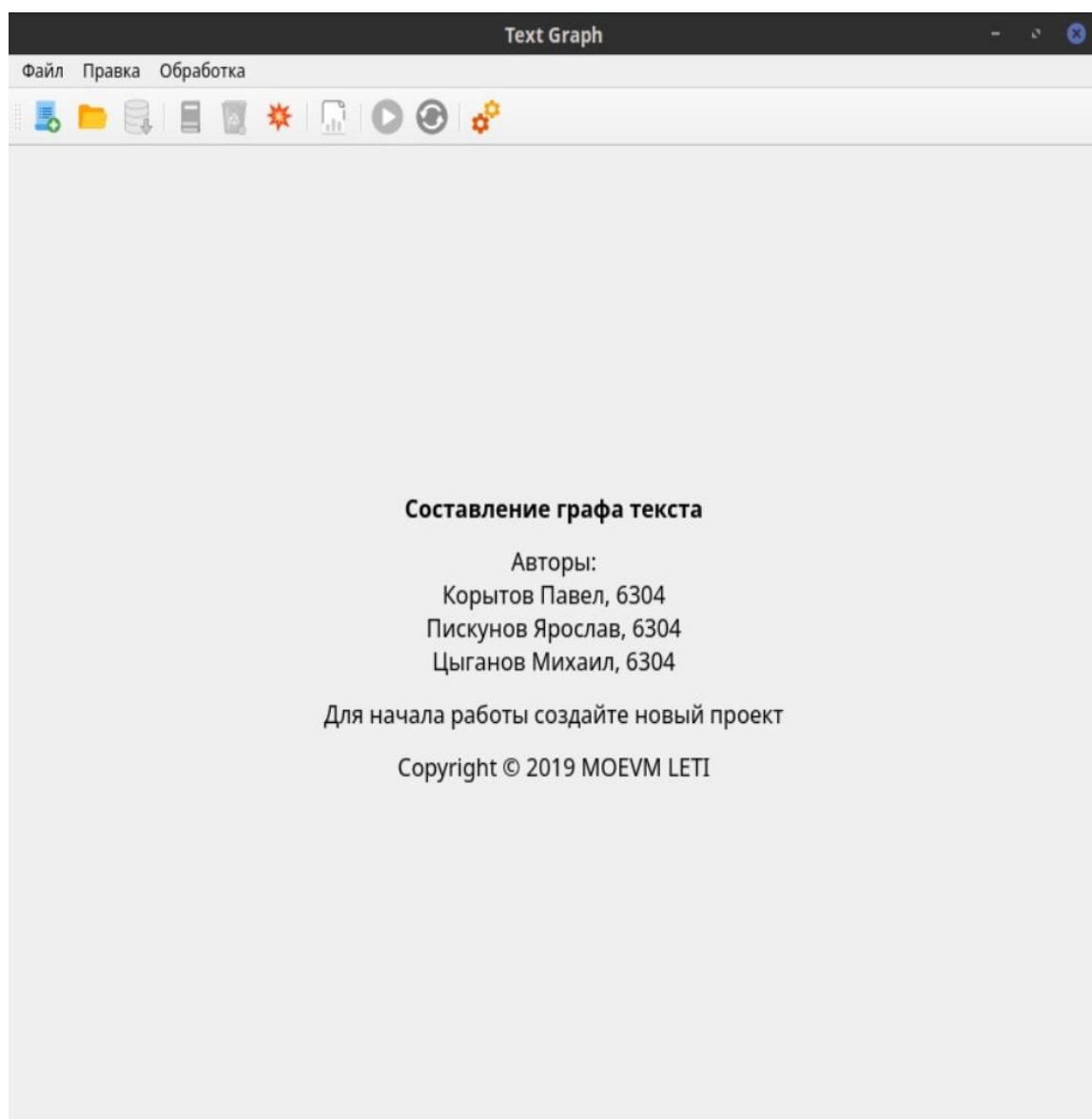


Рисунок 4 – окно программы до выбора текста

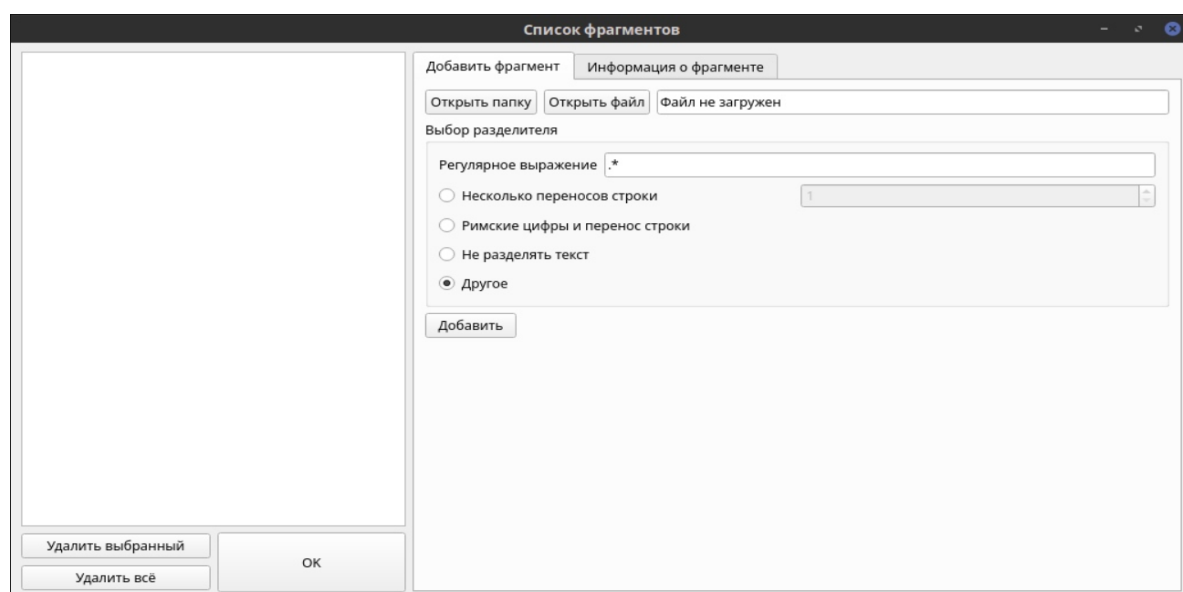


Рисунок 5 – окно разбиения текста на фрагменты до выбора текста

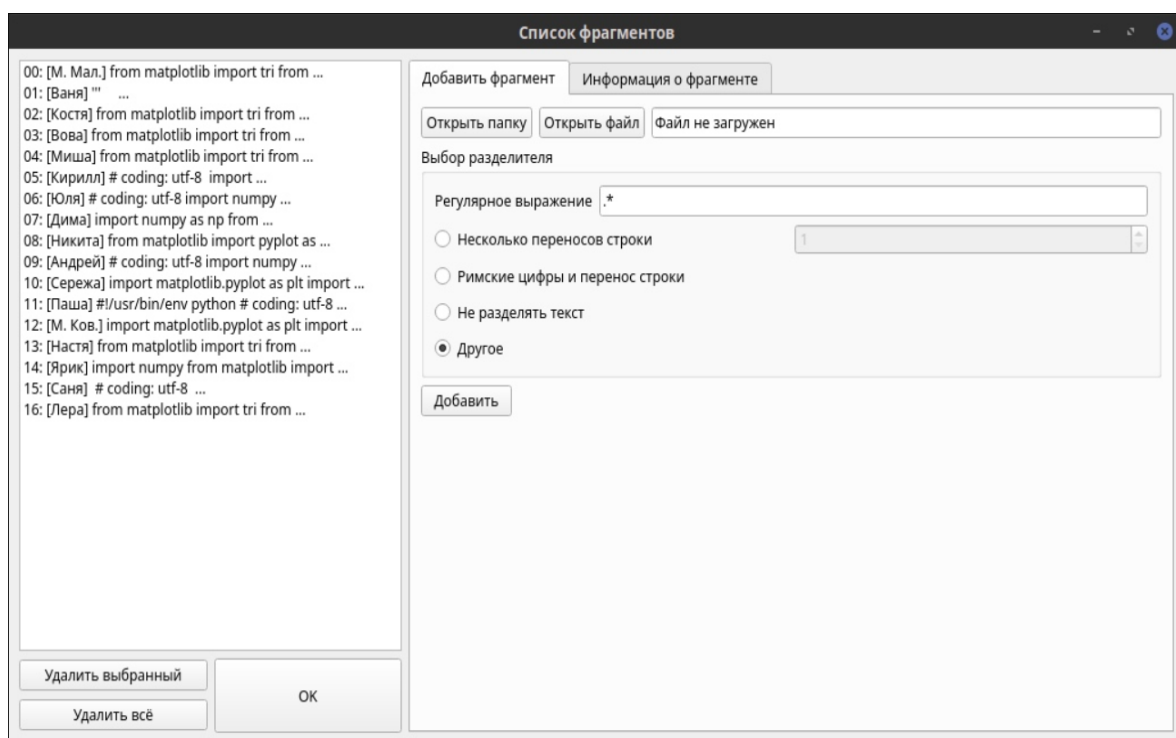


Рисунок 6 – окно разбиения текста на фрагменты после выбора текста

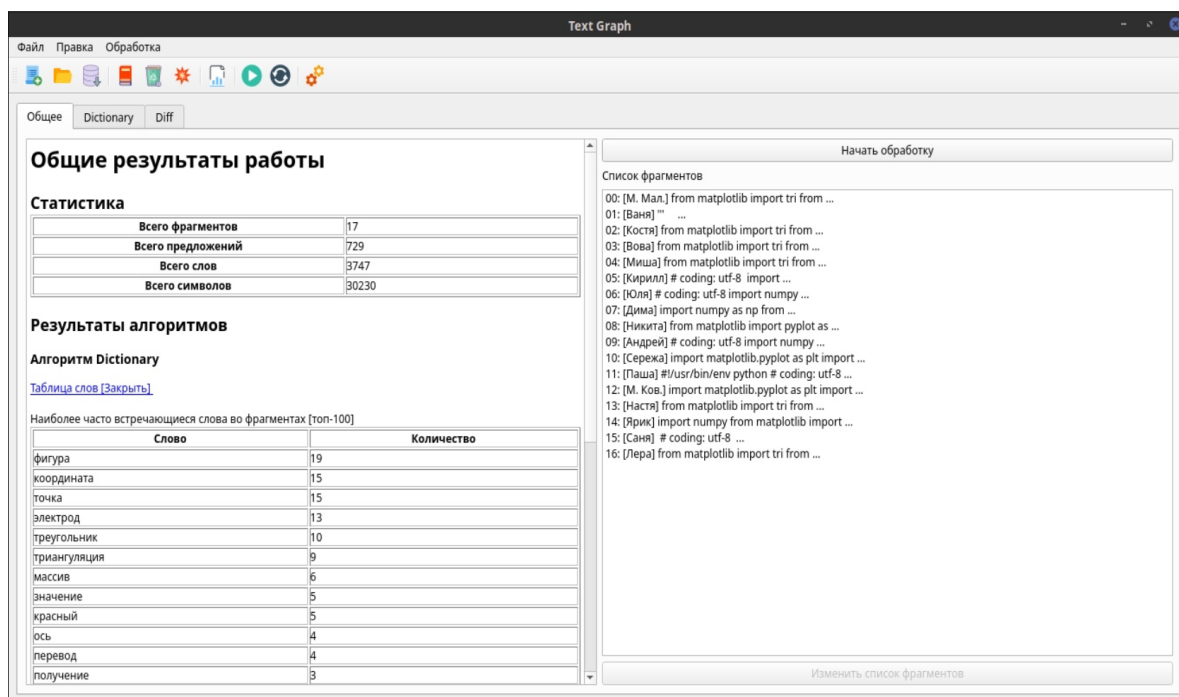


Рисунок 7 – окно программы после выбора текста

Параметры анализа

Общие настройки анализа

Включенные алгоритмы Изменить список

Минимальный вес ребра [0-1]

☒ Провести подсчёт общих результатов

Основное окно

☐ Автоматически обновлять результаты

Алгоритм работы со словарями

Сколько слов запоминать

Минимальная частота

Исключить слова Изменить список

Алгоритм сравнения строк

☒ Построчное сравнение (быстрее)

Отображение графа

Размер вершины

Ширина стрелок

Масштабирование layout-ов

Расчёт гравитации

☐ Включить гравитацию

Частота расчёта гравитации (мс)

Сила притяжения вершин к центру

Отмена
ОК (перезапустить)
ОК

Рисунок 8 – окно настройки параметров анализа

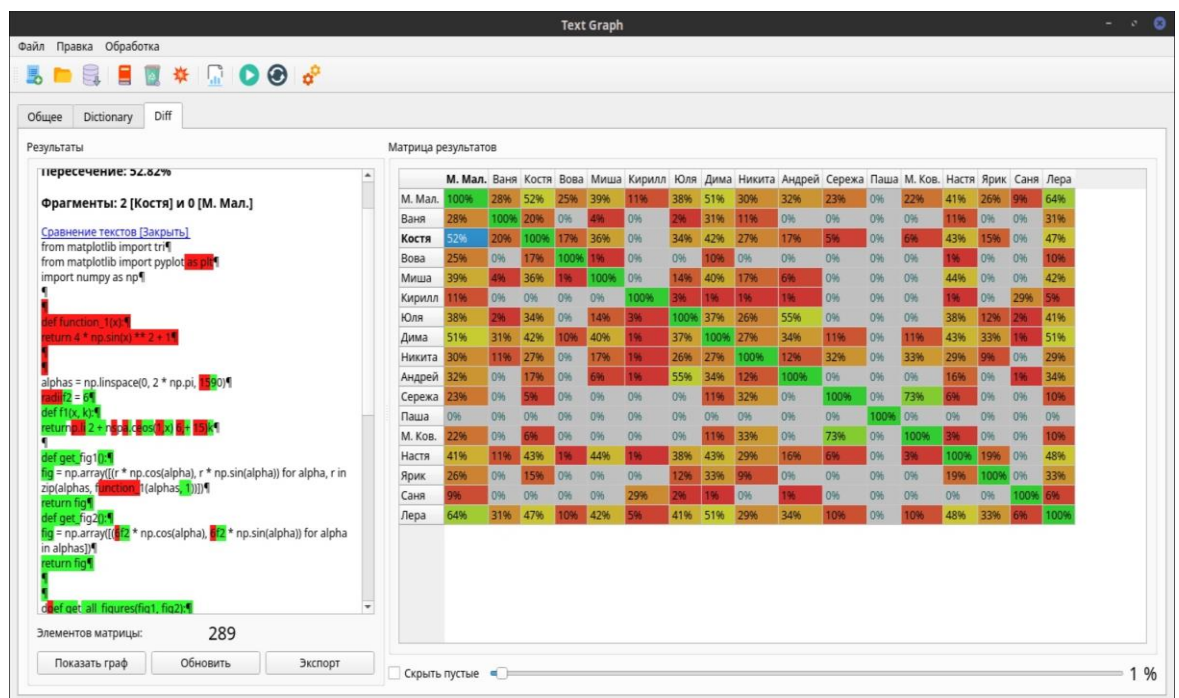


Рисунок 9 – окно с результатами работы алгоритма

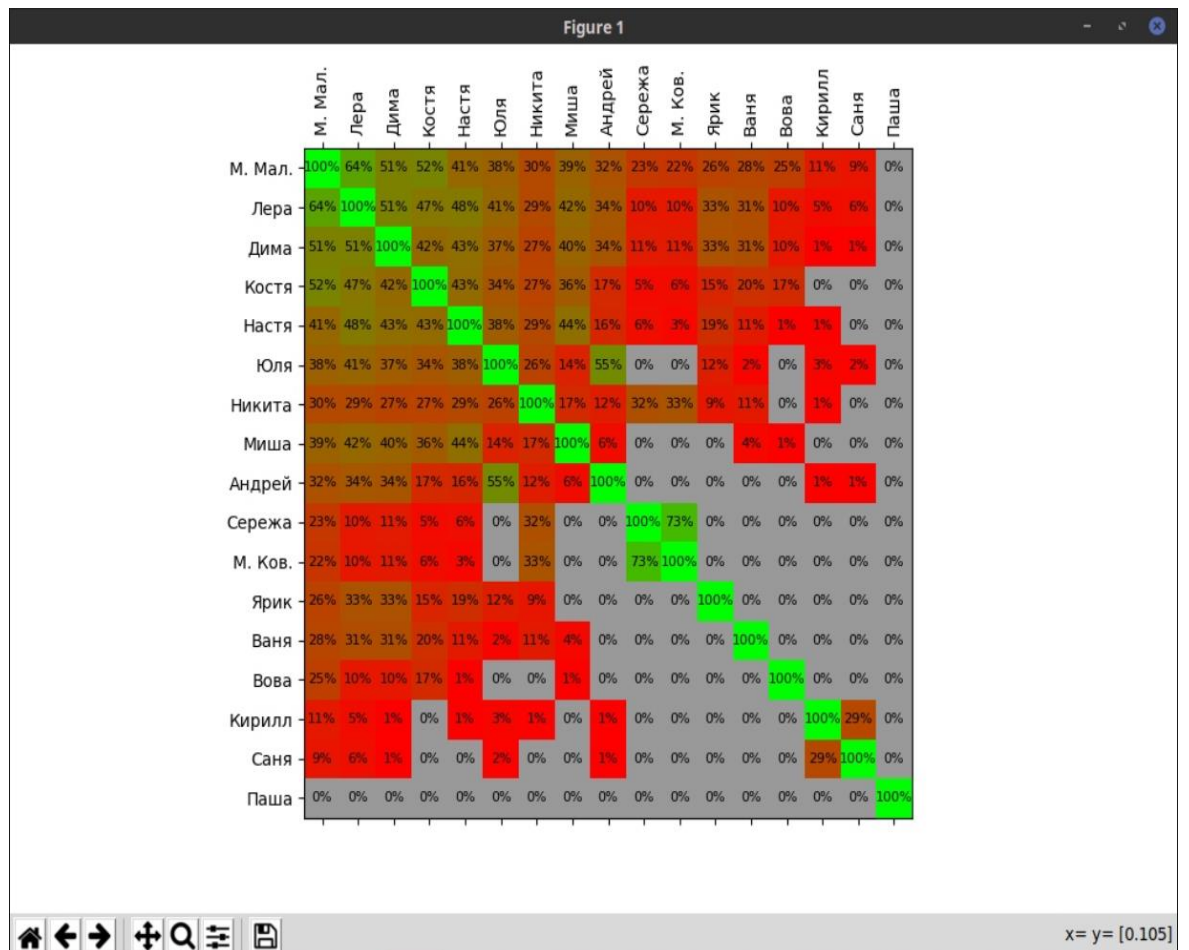


Рисунок 10 – упорядоченная матрица результатов алгоритма

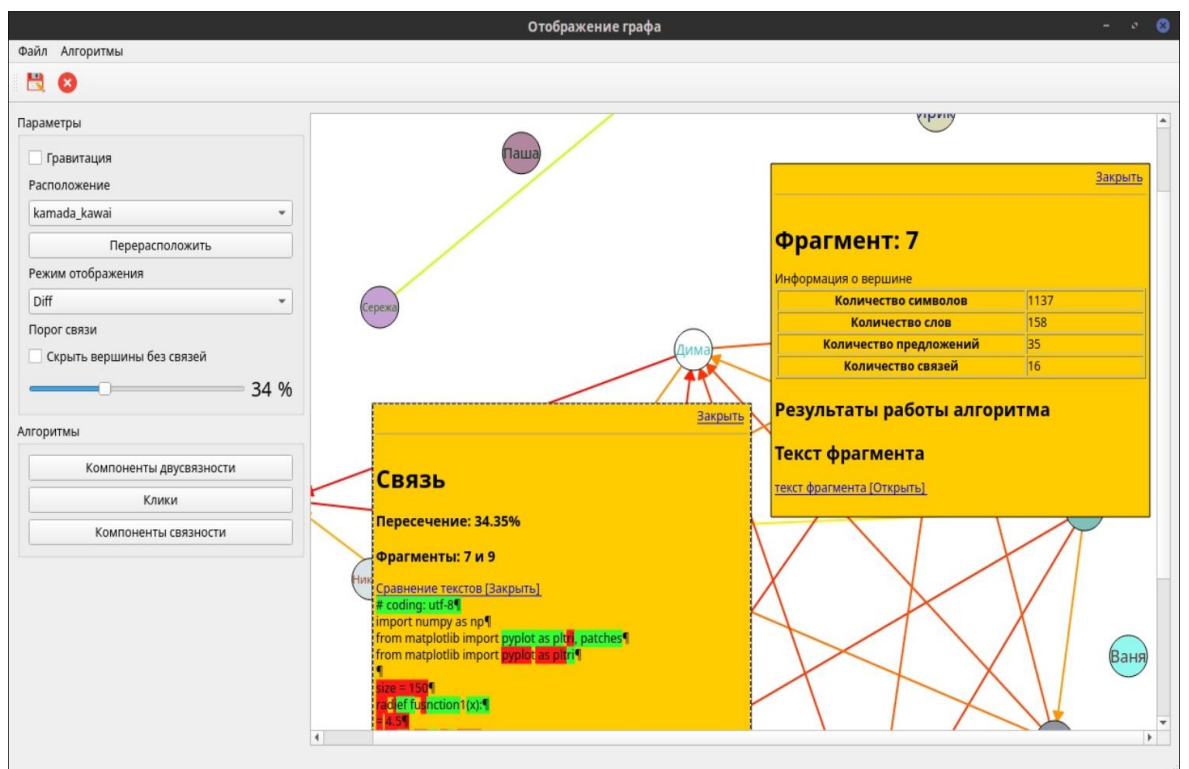


Рисунок 11 – окно представления графа по результатам алгоритма

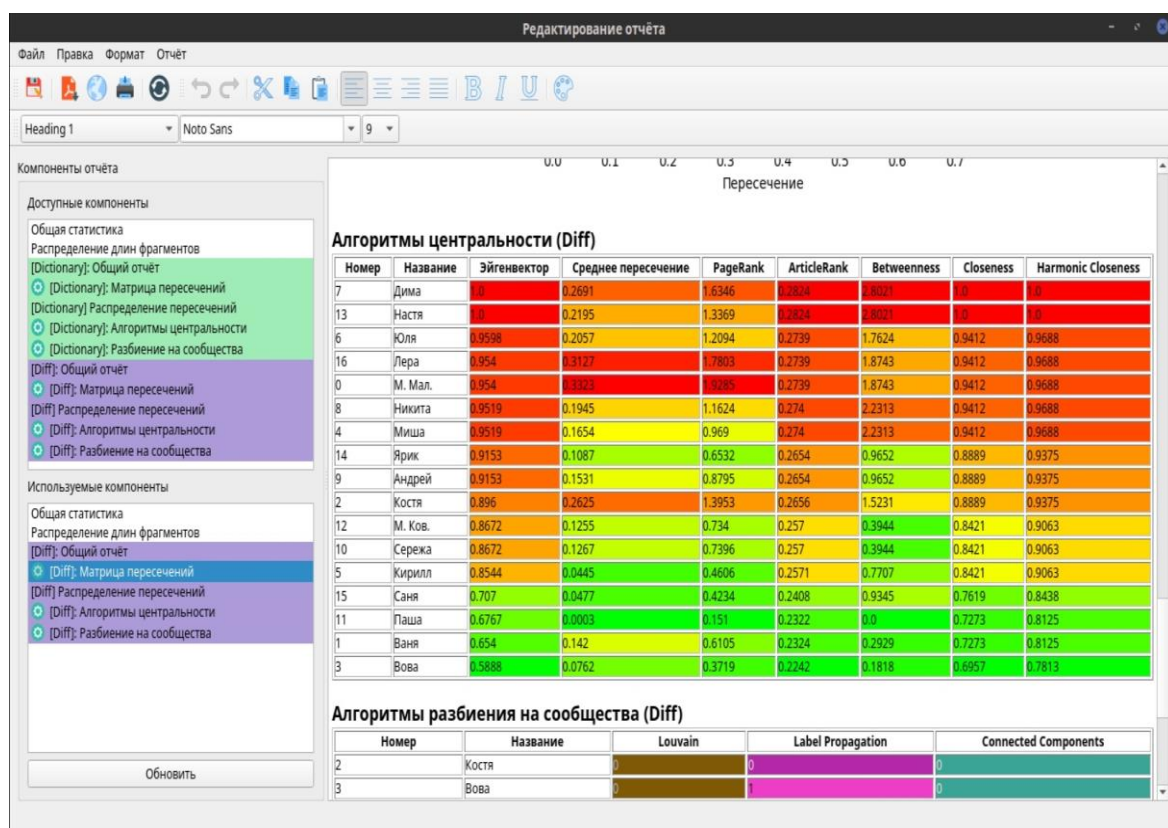


Рисунок 12 – окно экспорта результатов в отчет

4.3. Используемые технологии

- БД: neo4j
- ORM: neomodel
- Backend: Python 3
- Frontend: PyQt5
- Развертывание: PyInstaller, dpkg-deb
- Анализ текстов: NLTK, Mystem
- Работа с графами: Networkx
- Экспорт в pdf: wkhtmltox
- IDE, плагины и средства разработки: neovim с python-language-server, pylama, pudb, PyCharm

4.4. Инструкция по сборке и развертыванию приложения

Инструкция по сборке программы из исходников может быть найдена по ссылке: <https://github.com/moevm/nosql1h19-text-graph/wiki/Запуск-из-ИСХОДНИКОВ>.

4.5. Ссылки на приложение

- Репозиторий с приложением – <https://github.com/moevm/nosql1h19-text-graph>
- Релизный пакет для Linux – <https://github.com/moevm/nosql1h19-text-graph/releases/download/1.0/text-graph-amd64.deb>

ЗАКЛЮЧЕНИЕ

Результаты

В ходе выполнения данной работы было создано настольное GUI-приложение, в котором имеется возможность выбрать файл или несколько, разбить его на фрагменты с помощью регулярных выражений и проанализировать связность этих фрагментов, выбрав один из предложенных алгоритмов. Также отображается общая информация по тексту. После имеется возможность сохранить результат. Были освоены методы работы с нереляционной СУБД neo4j.

Недостатки и пути для улучшения полученного решения

В текущей реализации приложения реализовано всего два алгоритма анализа, еще один находится в разработке на момент релиза. Сами алгоритмы не идеально работают и нет возможности дать 100% правильности их работы. Для улучшения полученного решения необходимо доработать существующие алгоритмы анализа и добавить новые.

Будущее развитие решения

Планируется доработать алгоритм анализа фрагментов по упоминаемым в них именам собственным с использованием библиотеки natasha. Также планируется выпустить полноценный релиз для ОС Windows.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Python: <https://docs.python.org/3/>
2. Документация PyQt5: <https://www.riverbankcomputing.com/static/Docs/PyQt5/>
3. Документация neo4j: <https://neo4j.com/docs/>
4. Документация neomodel: <https://neomodel.readthedocs.io/en/latest/index.html>
5. Документация по PyInstaller: <https://pyinstaller.readthedocs.io/en/stable/>

ПРИЛОЖЕНИЕ А

Моделирование БД

Вершины

В вершинах графа будет храниться выбранный фрагмент текста в нормализованном виде.

Анализируемый текст - "Война и мир", Л.Н. Толстой.

Текст делится на параграфы. Параграфы формируют главы, главы - части.

Производится моделирование для случаев разбиения на параграфы, главы и части

Выборка значений из текста

Символов в параграфах

$$S_{par} := [1015 \ 412 \ 456 \ 231 \ 353 \ 245 \ 176 \ 286 \ 171 \ 420 \ 75 \ 1420 \ 189 \ 429 \ 537]^T$$

Символов в главах

$$S_{chp} := [12550 \ 7519 \ 11371 \ 16585 \ 7438 \ 22404 \ 8428 \ 4589 \ 6643 \ 3787 \ 8037 \ 8391]^T$$

Символов в частях

$$S_{blk} := [267558 \ 214656 \ 121649 \ 182496 \ 176234 \ 113721 \ 162835]^T$$

Средние значения

$$s_{par_m} := \text{floor} \left(\text{mean} (S_{par}) \right) = 4.27 \cdot 10^2$$

$$s_{chp_m} := \text{floor} \left(\text{mean} (S_{chp}) \right) = 9.81 \cdot 10^3$$

$$s_{blk_m} := \text{floor} \left(\text{mean} (S_{blk}) \right) = 1.77 \cdot 10^5$$

$$s_{total} := \sum S_{blk} = 1.24 \cdot 10^6 \quad - \text{ всего символов}$$

$$n_{par} := \text{floor} \left(\frac{s_{total}}{s_{par_m}} \right) = 2.9 \cdot 10^3 \quad - \text{ всего параграфов}$$

$$n_{chp} := \text{floor} \left(\frac{s_{total}}{s_{chp_m}} \right) = 1.3 \cdot 10^2 \quad - \text{ всего глав}$$

$$n_{blk} := \text{length} (S_{blk}) = 7 \quad - \text{ всего частей}$$

Некоторые константы

$$v_{sym} := 4 \quad - \text{ вес символа (здесь и далее - байт)}$$

$$v_{float} := 4 \quad - \text{ вес float}$$

$$v_{node_neo4j} := 15 \quad - \text{ вес вершины в neo4j (из документации)}$$

$$v_{edge_neo4j} := 33 \quad - \text{ вес связи в neo4j}$$

$$v_{prop_neo4j} := 41 \quad - \text{ вес свойства в neo4j}$$

Объем вершин в памяти

Далее размеры зависят от длины фрагмента.

$$v_{node_one} (s_m) := s_m \cdot v_{sym} + v_{node_neo4j} + v_{prop_neo4j} \quad - \text{ вес одной вершины}$$

$$v_{node_total} (n, s_m) := v_{node_one} (s_m) \cdot n \quad - \text{ вес всех вершин}$$

Ребра

В ребрах хранится информация о связях между фрагментами. На данный момент планируется реализовать два алгоритма вычисления связей:

- Пересечение словарей
- Пересечение имен собственных

В этом случае в ребре будет храниться коэффициент от 0 до 1 и 5 слов, по которым обнаружено пересечение.

В случае наличия достаточного времени в программу будут добавлены и другие алгоритмы.

Некоторые предположения

$s_{word_m} := 6$ - среднее количество символов в значащем слове

Объем ребра в памяти

$$v_{edge_one} := v_{float} + s_{word_m} \cdot v_{sym} + v_{edge_neo4j} + v_{prop_neo4j} \cdot 2 = 1.43 \cdot 10^2$$

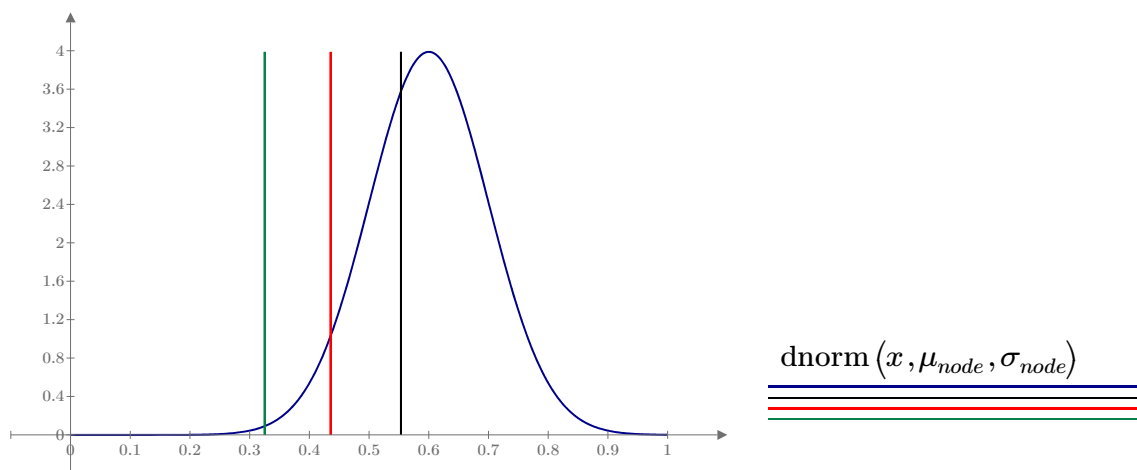
Моделирование

Характер связей

Предположим, что вероятность связи от одного фрагмента к другому подчинена закону нормального распределения.

Проверим затраты памяти без отсечения связей или с отсечением по 68% (условно 1σ), 95% (2σ) и 99.7% (3σ) от левого хвоста

$$\mu_{node} := 0.6 \quad \sigma_{node} := 0.1 \quad \sigma_0 := 0 \quad \sigma_1 := 1 - 0.68 \quad \sigma_2 := 1 - 0.95 \quad \sigma_3 := 1 - 0.997$$



x

$$qnorm(\sigma_1, \mu_{node}, \sigma_{node})$$

$$qnorm(\sigma_2, \mu_{node}, \sigma_{node})$$

$$qnorm(\sigma_3, \mu_{node}, \sigma_{node})$$

Таким образом, вес всех ребер:

$$v_{edge_total}(n, \sigma) := \text{floor} \left(n \cdot n \cdot (1 - \sigma) \cdot v_{edge_one} \right)$$

И общий вес:

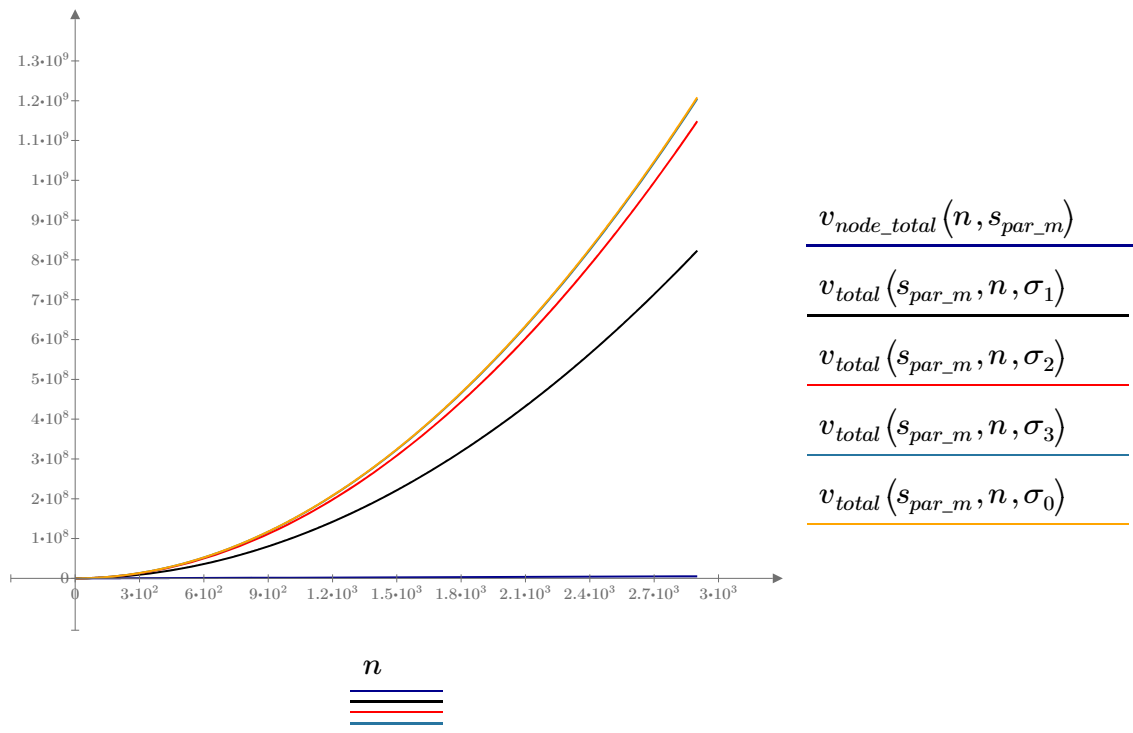
$$v_{total}(s, n, \sigma) := v_{node_total}(n, s) + v_{edge_total}(n, \sigma)$$

$$\Sigma := [\sigma_1 \ \sigma_2 \ \sigma_3]^T$$

$$P_{saving}(s, n) := \text{for } i \in 0 \dots \text{length}(\Sigma) - 1 \quad \left| \begin{array}{l} \text{return } r \\ r_i \leftarrow 1 - \frac{v_{total}(s, n, \Sigma_i)}{v_{total}(s, n, \sigma_0)} \end{array} \right| \quad \begin{array}{l} \text{- ДОЛЯ ЭКОНОМИИ ОТ ОТКАЗА ОТ ЧАСТИ} \\ \text{СВЯЗЕЙ} \end{array}$$

$$p_{node}(s, n, \sigma) := \frac{v_{node_total}(n, s)}{v_{total}(s, n, \sigma)} \quad \text{- ДОЛЯ ВЕРШИН}$$

Моделирование для параграфов



$$v_{total}(s_{par_m}, n_{par}, \sigma_0) = 1.21 \cdot 10^9 \quad \text{- общий вес при разбиении на параграфы}$$

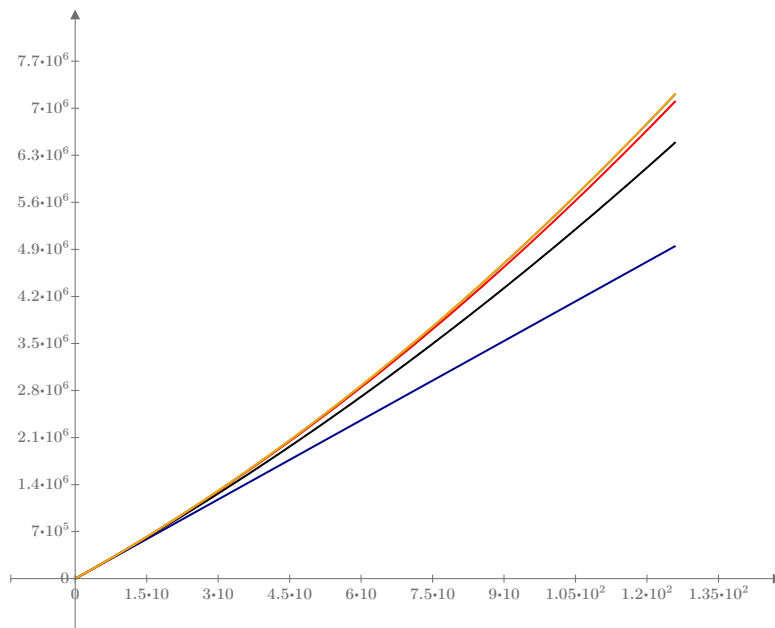
$$P_{saving}(s_{par_m}, n_{par}) = \begin{bmatrix} 31.86\% \\ 4.98\% \\ 0.3\% \end{bmatrix} \quad \text{- экономия места при отказе от части связей (32%, 5%, 0.3%)}$$

$$p_{node}(s_{par_m}, n_{par}, \sigma_0) = 0.42\% \quad \text{- доля вершин в общем объеме}$$

Как видно, в случае разбиения на параграфы ребра составляют подавляющую часть объема. В этом случае отсекание части связей приводит к почти пропорциональному

ЭКОНОМИЮ МЕСТА.

Моделирование для глав



$$v_{node_total}(n, s_{chp_m})$$

$$v_{total}(s_{chp_m}, n, \sigma_1)$$

$$v_{total}(s_{chp_m}, n, \sigma_2)$$

$$v_{total}(s_{chp_m}, n, \sigma_3)$$

$$v_{total}(s_{chp_m}, n, \sigma_0)$$

n

$$v_{total}(s_{chp_m}, n_{chp}, \sigma_0) = 7.22 \cdot 10^6$$

- общий вес при разбиении на главы

$$P_{saving}(s_{chp_m}, n_{chp}) = \begin{bmatrix} 10.06\% \\ 1.57\% \\ 0.09\% \end{bmatrix}$$

- экономия места при отказе от части связей (32%, 5%, 0.3%)

$$p_{node}(s_{chp_m}, n_{chp}, \sigma_0) = 68.56\%$$

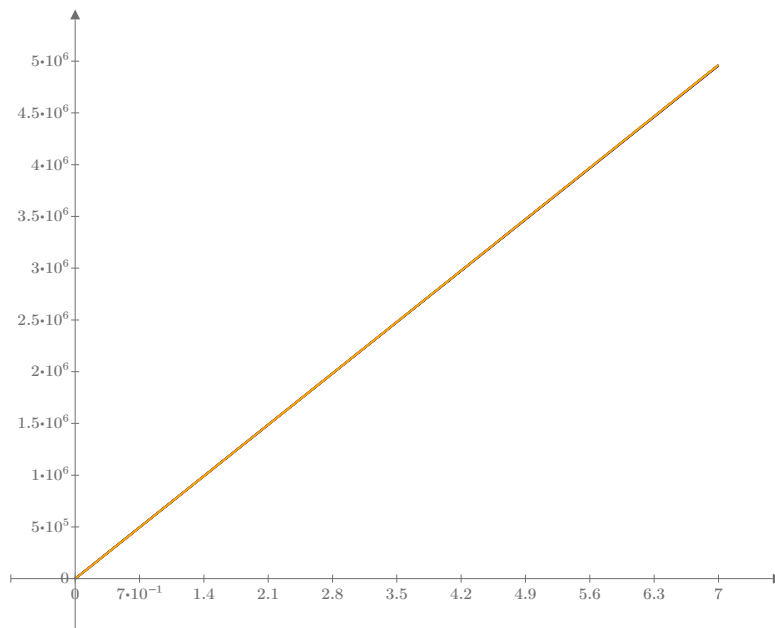
- доля вершин в общем объеме

$$\frac{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)}{v_{total}(s_{par_m}, n_{par}, \sigma_0)} = 0.6\%$$

- отношение объемов БД при разбиении на главы и на параграфы

В случае разбиения на главы объем БД гораздо меньше. В этом случае текст составляет более двух третей объема, и экономия места при отсечении вершин в несколько раз ниже

Моделирование для частей



$$\frac{v_{node_total}(n, s_{blk_m})}{\frac{v_{total}(s_{blk_m}, n, \sigma_1)}{\frac{v_{total}(s_{blk_m}, n, \sigma_2)}{\frac{v_{total}(s_{blk_m}, n, \sigma_3)}{v_{total}(s_{blk_m}, n, \sigma_0)}}}}$$

n



$$v_{total}(s_{blk_m}, n_{blk}, \sigma_0) = 4.96 \cdot 10^6 \quad - \text{общий вес при разбиении на части}$$

$$P_{saving}(s_{blk_m}, n_{blk}) = \begin{bmatrix} 0.05\% \\ 0.01\% \\ 0 \end{bmatrix} \quad - \text{экономия места при отказе от части связей (32\%, 5\%, 0.3\%)}$$

$$p_{node}(s_{blk_m}, n_{blk}, \sigma_0) = 99.86\% \quad - \text{доля вершин в общем объеме}$$

$$\frac{v_{total}(s_{blk_m}, n_{blk}, \sigma_0)}{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)} = 68.73\% \quad - \text{отношение объемов БД при разбиении на части и на главы}$$

Подавляющее большинство объема занимает текст. Отсечение вершин не имеет смысла

Сравнение с SQL

Для анализа выберем БД MySQL

```
CREATE DATABASE nosql_graph_text CHARACTER SET utf8 COLLATE  
utf8_unicode_ci;
```

Пусть фрагменты хранятся в следующей таблице:

```
CREATE TABLE texts (  
    id INT AUTO_INCREMENT PRIMARY_KEY,  
    text LONGTEXT  
)
```

Пусть связи хранятся в таблицах такого вида:

```
CREATE TABLE link_1 (  
    id INT AUTO_INCREMENT PRIMARY_KEY,  
    text_1_id INT,  
    text_2_id INT,  
    intersection FLOAT,  
    word_1 NVARCHAR(20),  
    word_2 NVARCHAR(20),  
    word_3 NVARCHAR(20),  
    word_4 NVARCHAR(20),  
    word_5 NVARCHAR(20),  
  
    CONSTRAINT text1_index_constraint  
    INDEX ind_text1(text_1_id),  
  
    CONSTRAINT text2_index_constraint,  
    INDEX ind_text2(text_2_id),  
  
    CONSTRAINT text1_key_constraint  
    FOREIGN_KEY fk_text1(text_1_id)  
    REFERENCES texts(id)  
    ON_UPDATE CASCADE  
    ON_DELETE CASCADE,  
  
    CONSTRAINT text2_key_constraint  
    FOREIGN_KEY fk_text2(text_2_id)  
    REFERENCES texts(id)  
    ON_UPDATE CASCADE  
    ON_DELETE CASCADE  
)
```

Вычислим объем тех же данных в MySQL.

Некоторые константы:

$$v_{nvarchar_sql}(n) := n \cdot 4 + 1$$
$$v_{int_sql} := 4$$
$$v_{longtext_sql}(n) := n \cdot 4 + 4$$
$$v_{float_sql} := 4$$

Индексы

При использовании движка InnoDB первичный ключ не занимает лишнего места. Для использования внешних ключей нужны индексы.

Размер индекса обычно близок к результату следующей формулы:

$$v_{index_sql}(n, v_{field_pk}, v_{field}) := n \cdot (v_{field_pk} + v_{field}) \cdot 3$$

Размер таблиц в памяти

$$v_{texts_sql}(n, s) := n \cdot (v_{int_sql} + v_{longtext_sql}(s))$$

$$v_{links_index_sql}(n, \sigma) := v_{index_sql}(n \cdot n \cdot (1 - \sigma), v_{int_sql}, v_{int_sql})$$

$$v_{links_fields_sql}(n, \sigma) := n \cdot n \cdot (1 - \sigma) \cdot (v_{int_sql} \cdot 3 + v_{nvarchar_sql}(20) \cdot 5 + v_{float_sql})$$

$$v_{links_sql}(n, \sigma) := \text{floor}(v_{links_index_sql}(n, \sigma) + v_{links_fields_sql}(n, \sigma))$$

$$v_{total_sql}(s, n, \sigma) := v_{texts_sql}(n, s) + v_{links_sql}(n, \sigma)$$

Сравнение

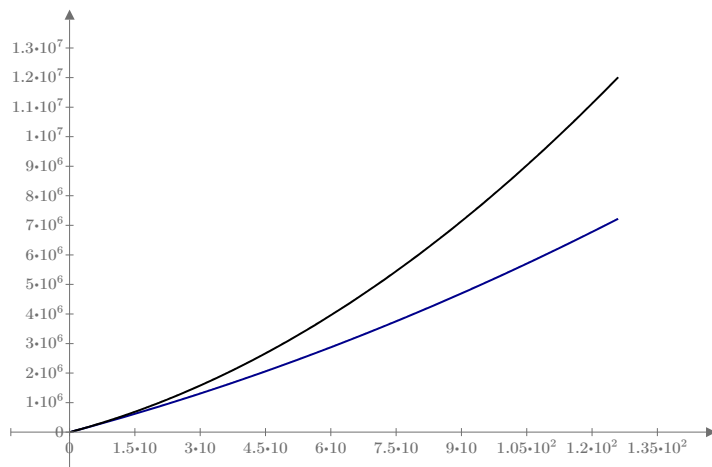
$$\frac{v_{total}(s_{blk_m}, n_{blk}, \sigma_0)}{v_{total_sql}(s_{blk_m}, n_{blk}, \sigma_0)} = 99.71\% \quad - \text{сравнение для разбиения по частям}$$

$$\frac{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)}{v_{total_sql}(s_{chp_m}, n_{chp}, \sigma_0)} = 60.13\% \quad - \text{сравнение для разбиения по главам}$$

$$\frac{v_{total}(s_{par_m}, n_{par}, \sigma_0)}{v_{total_sql}(s_{par_m}, n_{par}, \sigma_0)} = 32.23\% \quad - \text{сравнение для разбиения по параграфам}$$

Получается, что при разбиении по частям объем примерно одинаков; при разбиении по главам и параграфам - несколько выигрывает NoSQL.

График зависимости



n

$$\frac{v_{total}(s_{chp_m}, n, \sigma_0)}{v_{total_sql}(s_{chp_m}, n, \sigma_0)}$$

$$\frac{v_{total_sql}(s_{chp_m}, n, \sigma_0)}{v_{total_sql}(s_{chp_m}, n, \sigma_0)}$$

Моделирование запросов

Получить тексты с процентами связи > N

Cypher

```
MATCH (t1:Text)-[rel:ALG1]-(t2:Text)
WHERE rel.intersection > ${N}
RETURN id(t1), id(t2), rel.intersection
```

SQL

```
CREATE procedure get_intersect_more(IN N float)
BEGIN
    SELECT text1_id, text2_id, intersection FROM link_1
    WHERE intersection > N
END;
```

Добавление вершины

Cypher

```
CREATE (t:Text {text: ${TEXT}})
```

SQL

```
INSERT INTO texts
VALUES ${TEXT}
```

Кратчайший путь

Cypher

```
MATCH (start:Text{id: ${start_id}}), (end:Text{id: ${end_id}})
CALL algo.shortestPath.stream(start, end, 'cost' , {
    nodeQuery: 'MATCH (n:Text) RETURN id(n) AS id',
    relationshipQuery: 'MATCH (t1:Text)-[rel:${REL_TYPE}]- (t2:Text)
        WHERE rel.intersection > ${N}
        RETURN id(t1) AS source, id(t2) as target, 1 as weight'
})
YIELD nodeId, cost
RETURN nodeId, cost
```

SQL

На SQL нужно во многом реализовывать вне языка запросов. Например, для Алгоритма Дейкстры нужны следующие процедуры

```
CREATE procedure get_weight(IN id1 INT, IN id2 INT, IN max FLOAT)
BEGIN
```

```
    SELECT intersection FROM links
    WHERE links.text_1_id = id1 AND links.text_2_id = id2 AND links.intersection >=
max
END
```

```
CREATE procedure get_adjascent_nodes(IN id, IN max FLOAT)
BEGIN
```

```
    SELECT text_1_id AS text_id
    FROM links
    WHERE links.text_2_id = id
UNION
    SELECT text_2_id AS text_id
    FROM links
    WHERE links.text_1_id = id
```

END

Вышеприведённые запросы имеют шанс получить сложность $O(m)$ (правда, создание индексов поможет сгладить этот недостаток). В neo4j, в отличие от SQL-варианта, для поиска веса связи между вершинами не нужно итерироваться по всем связям графа.

Поэтому neo4j может быть эффективнее SQL в $O(m/n)$ раз. Рационально предположить, что это верно и для нижеприведённых алгоритмов

PageRank

Cypher

CALL algo.pageRank.stream('Text', 'ALG1', {iterations:20, dampingFactor:0.85})

YIELD nodeId, score

RETURN algo.getNodeById(nodeId).id **AS** page,score

ORDER BY score **DESC**

Компоненты связности

Cypher

CALL algo.unionFind.stream('Text', 'ALG1', {
 weightProperty:'intersection',
 defaultValue:0.0,
 threshold: \${N},
 concurrency: 1
})

YIELD nodeId,setId

RETURN algo.getNodeById(nodeId).id **AS** fragId, setId

ORDER BY setId, fragId

Выводы

По результатам сравнения эффективности neo4j и MySQL получены следующие результаты:

- Увеличение объема при увеличении числа текстов - $O(n^2)$.
- Отсечение части связей приводит к уменьшению места на $O(\sigma)$, при росте $O(n^2)$ это не имеет смысла
- neo4j тратит немного меньше места на хранение данных за счет динамического выделения памяти и отсутствия индексов

$$\frac{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)}{v_{total_sql}(s_{chp_m}, n_{chp}, \sigma_0)} = 60.13\%$$

- Разбиение большого текста на небольшие фрагменты нерационально и приводит к резкому возрастанию объема БД.
- При реализации алгоритмов на SQL эффективность может быть ниже в $O(m/n)$ раз. Это существенная разница, т.к. для больших текстов $m \gg n$. Кроме того, периодически необходимы трудоемкие операции вроде UNION.