##### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

##### «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

##### КАФЕДРА ВТ

##### ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3–4

##### «Разработка модуля ранжирования для поисковой системы»

##### по дисциплине «Интеллектуальные системы»

##### Выполнили: студенты гр. АММ2-24 Ириков Евгений Алексеевич

##### Атласюк Игорь Романович

##### Проверил: к.т.н., доцент Кафедры ВТ Яковина Ирина Николаевна

##### Новосибирск 2024

## Содержание

[Постановка задачи 3](#_bookmark0)

[Описание структуры модуля поиска 4](#_bookmark1)

[Результаты проверки работы 15](#_bookmark2)

[Вывод 19](#_bookmark2)

[Приложение 20](#_bookmark3)

# Постановка задачи

Задание: Разработать модуль обработки поисковых запросов и выдачи результатов с ранжированием. Выполнить анализ результатов ранжирования по эффективности для проиндексированного набора документов.

# Описание структуры модуля поиска

## Схема, описывающая структуру модуля поиска

На рисунке 1 продемонстрирована реализованная структура модуля поиска. В блоках изображены вызываемые функции, на стрелках указана их взаимосвязь с передаваемыми данными.

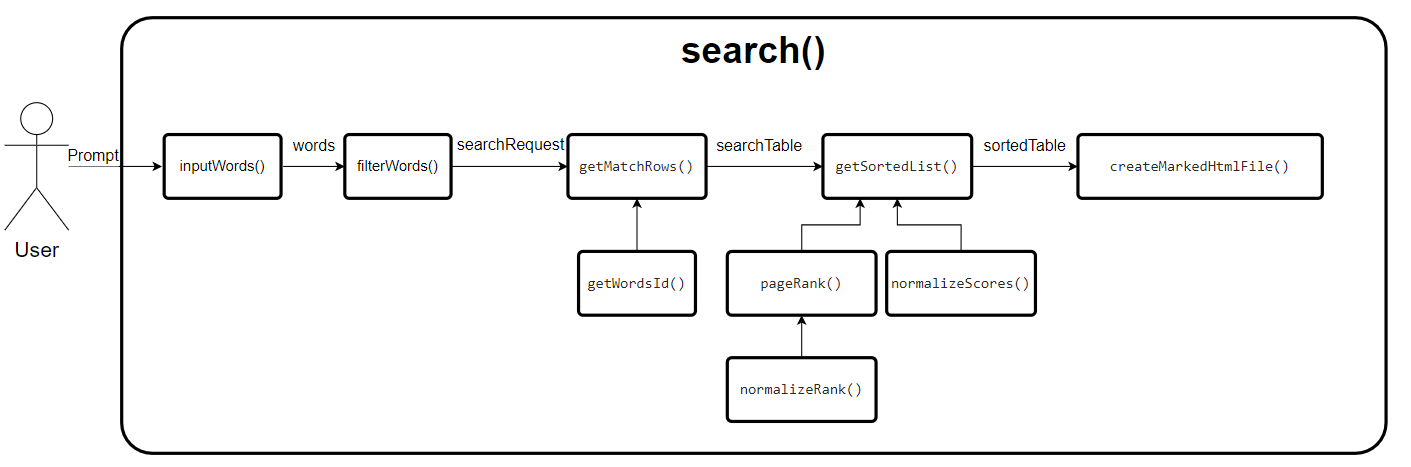


Рисунок 1. Общая схема модуля поиска

На рисунке 2 продемонстрирована программная реализации функции поиска, согласно схеме.

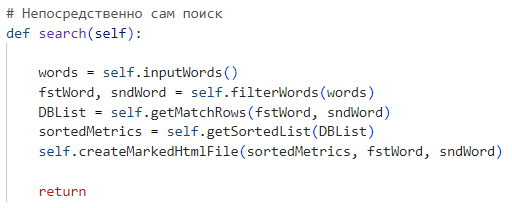


Рисунок 2. Программная реализации функции поиска

*Функция inputWords()*

В начале пользователь вводит искомые слова, которые считываются функцией inputWords для дальнейшей передачи программе.

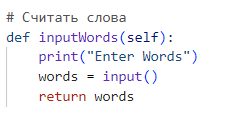


Рисунок 3. Программная реализации функции inputWords

*Функция filterWords()*

Далее эти слова нормализируются в леммы (именительный падеж, единственное число) функцией filterWords, формируя поисковой запрос.

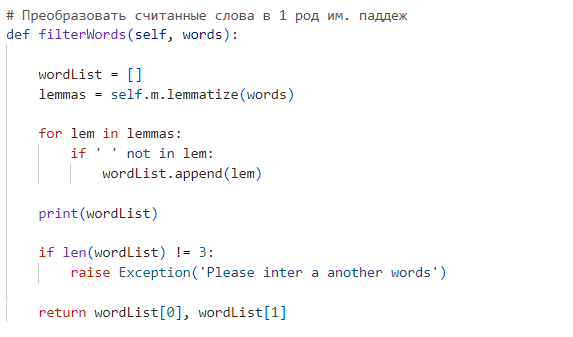


Рисунок 4. Программная реализации функции filterWords

*Функция getMatchRows()*

Далее функция getMatchRows формирует таблицу (рисунок 6), которая содержит сочетания позиций слов поискового запроса

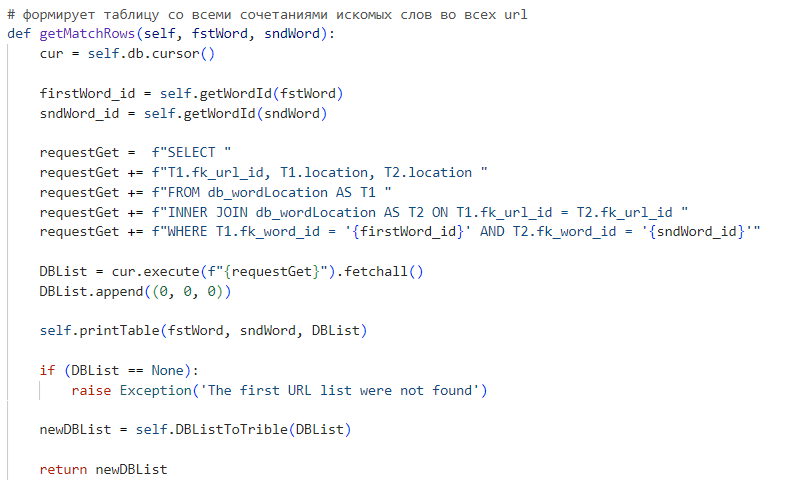


Рисунок 5. Программная реализации функции getMatchRows

Для работы функции используется вспомогательная функция getWordId, возвращающую уникальный идентификатор каждого слова поискового запроса из базы данных

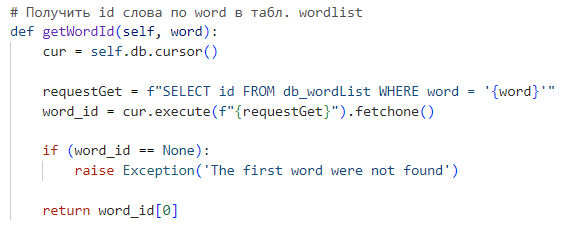


Рисунок 6. Программная реализации функции getWordId

Пример формируемой таблицы функцией getMatchRows:

* url\_ID – идентификатор адреса страницы
* wordLoc1 - позиция слова1 из поискового запроса на соотвующем url
* wordLoc2 - позиция слова2 из поискового запроса на соотвующем url

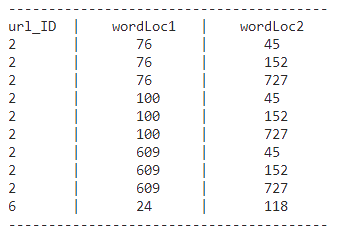


Рисунок 7. Пример таблицы getMatchRows

*Функция getSortedList()*

Далее выполняется функция getSortedList – функция-обертка, которая получает список вхождений комбинаций слов, вызывает расчет рангов *(функция pageRank)*, расчет метрик *(функции absPosMetrika, absPosMetrika, normalizeScores* *более подробно про них будет в следующем разделе.)* и выводит отсортированный список ранжированных URL.

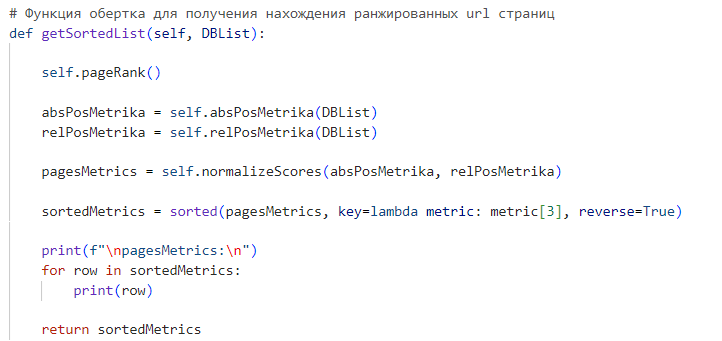


Рисунок 8. Программная реализации функции getSortedList

Пример формируемой таблицы функцией getSortedList:

* url\_ID – идентификатор адреса страницы
* M1 - нормализованное значение метрики расстоянии между словами
* M2 - нормализованное значение метрики расположения от начала документа
* M3 - усредненное значение M1 и М2, с учетом ранга соответствующей страницы
* URL\_text - текстовое значение url соответствующей страницы



Рисунок 9. Пример таблицы функции getSortedList

*Функция createMarkedHtmlFile()*

В конце функция createMarkedHtmlFile создает визуальную иллюстрацию к расположению искомых поискового запроса в полном тексте страницы, выделяя цветом слова поискового запроса

****

Рисунок 10. Программная реализации функции createMarkedHtmlFile

Пример помеченного HTML файла. Желтым выделены первое слова запроса, зеленым – второе.

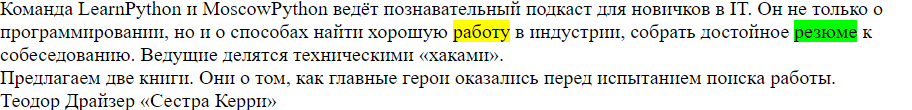
****

Рисунок 11. Пример файла функции createMarkedHtmlFile

**Описание алгоритмов для реализованных методов вычисления метрик**

Для ранжирования страниц использовались метрики «Расстояние между словами» и «Расположение от начала документа», а также алгоритм расчета ранга страниц на основе внешних ссылок PageRank.

Функция расчета метрики «Расстояние между словами» находит наименьшее полное расстояние между словами на странице.

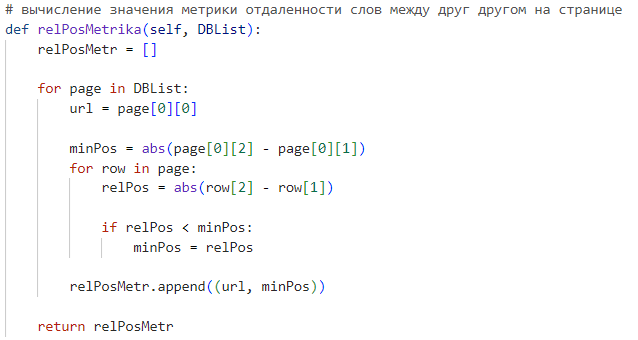


Рисунок 12. Программная реализации метрики «Расстояние между словами»

Функция расчета метрики «Расположение от начала документа» находит наименьшую позицию слов поискового запроса.

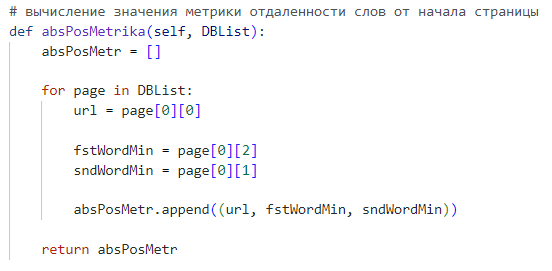


Рисунок 13. Программная реализации метрики

«Расположение от начала документа»

Функция нормализации normalizeScores() приводит ранги к диапазону от 0.0 до 1.0. Ранги масштабируются по близости к наилучшему результату, которому приписывается ранг 1.0.

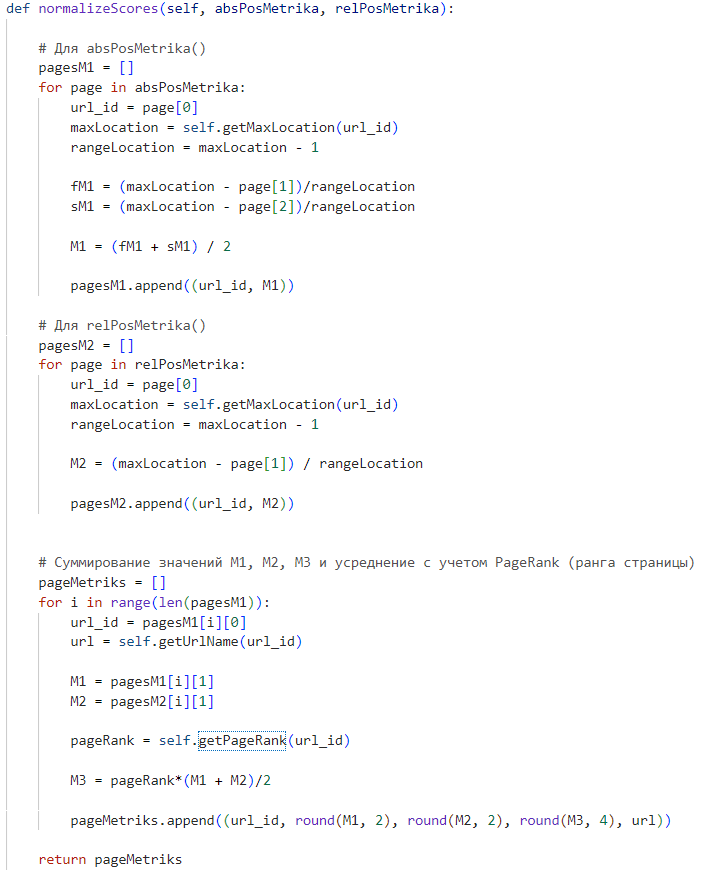


Рисунок 14. Программная реализации функции нормализации

Для ранжирования на основе внешних ссылок используется алгоритм PageRank. Алгоритм приписывает каждой странице ранг, оценивающий ее значимость. Значимость страницы вычисляется исходя из веса ссылающихся на нее страниц и общего количества ссылок, имеющихся на каждой из них. Для более корректной оценки веса страницы алгоритм рассчитывается несколько раз.

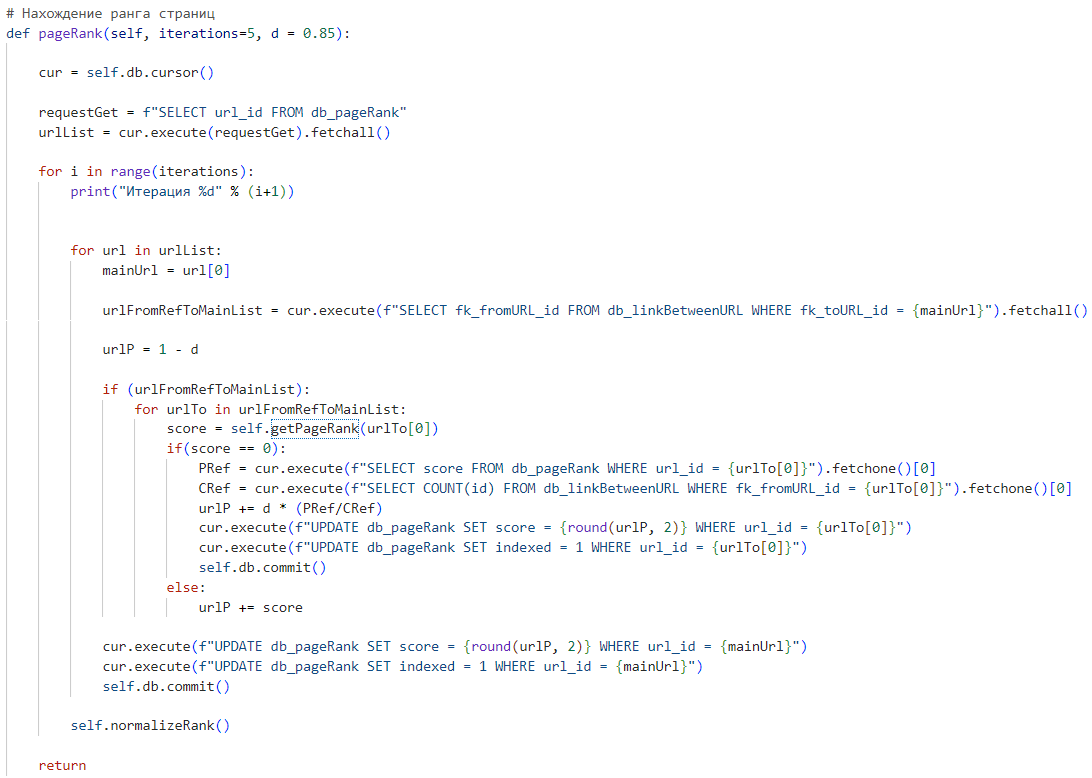


Рисунок 15. Программная реализации функции PageRank

После каждой итерации необходимо нормализовать полученные значения. Для этого реализована дополнительная функция normalizeRank()

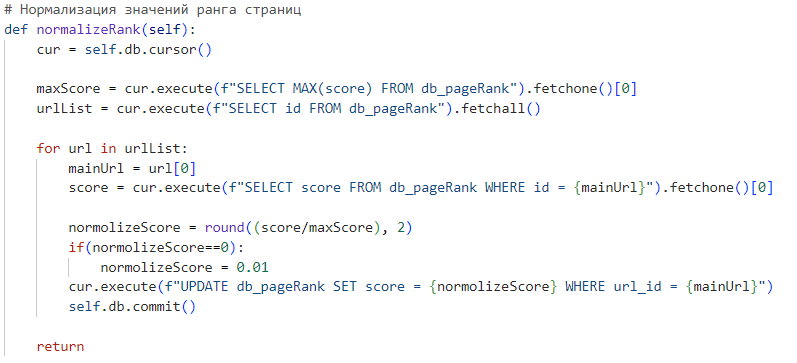


Рисунок 16. Программная реализации функции

нормализации PageRank

Также для хранения данных о весах страницы и информации были ли они проиндексированы была создана дополнительная таблица в базе данных

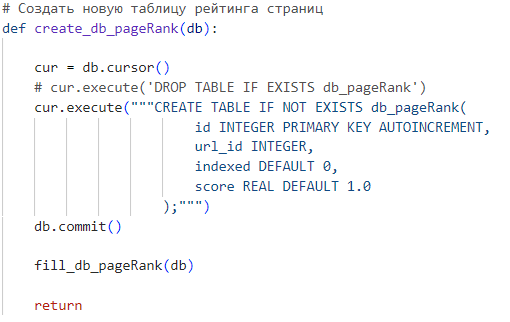


Рисунок 17. Создание дополнительной таблицы

для расчета ранга

# Результаты проверки работы

## Выбранный поисковый запрос

Для поиска введем запрос, состоящий из слов “которые” и “были”

Они переводятся в леммы (именительный падеж, единственное число)

“который” и “быть”



Рисунок 18. Поисковой запрос

## Уникальные идентификаторы для каждого слова из поискового запроса

Также из БД мы получаем уникальные идентификаторы для каждого слова, где

* word\_id – уникальный идентификатор слова из поискового
* word – соответствующее слово

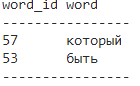


Рисунок 19. Уникальные идентификаторы

## Содержимое первых 20 строк табл. 1 со всеми сочетаниями позиций всех слов поискового запроса.

Функцией getMatchRows мы получаем таблицу, которая содержит все сочетания позиций всех слов поискового запроса.

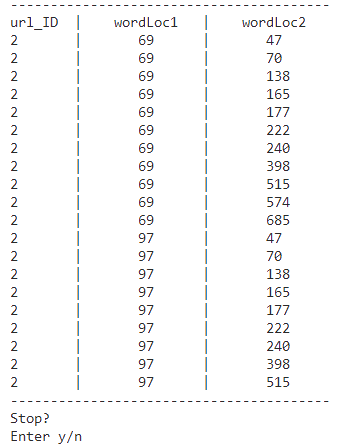


Рисунок 20. Содержимое первых 20 строк табл.1

Пояснения к таблице:

* url\_ID – идентификатор адреса страницы
* wordLoc1 - позиция слова1 из поискового запроса на соотвующем url
* wordLoc2 - позиция слова2 из поискового запроса на соотвующем url

## Количество уникальных url-адресов из БД содержащих слова поискового запроса (до процедуры ранжирования)

Выведем количество уникальных url-адресов из БД содержащих слова поискового запроса (до процедуры ранжирования)



Рисунок 21. Количество уникальных url-адресов

## Результаты расчета метрик для страниц, подходящих под поисковый запрос.

Выведем результаты расчета метрик для страниц, подходящих под поисковый запрос, упорядоченных по убыванию значений M3



Рисунок 22. Таблица 2 ранжированных страниц

Пояснения к таблице:

* url\_ID – идентификатор адреса страницы
* M1 - нормализованное значение метрики расстоянии между словами
* M2 - нормализованное значение метрики расположения от начала документа
* M3 - усредненное значение M1 и М2, с учетом ранга соответствующей страницы
* URL\_text - текстовое значение url соответствующей страницы

## Текстовое содержимое страниц с выделенными цветом словами поискового запроса

Посмотрим содержание трех страниц, имеющих максимальное значение M3, обработанных функцией createMarkedHtmlFile()

Рассмотрим первую страницу. Расстояние между словами поискового запроса у нее 0.78, что видно в последнем абзаце, где расстояние между словами – 10.

Расположение от начала страницы 0.99, т.к. первое встречаемое слово поискового запроса находится в заголовке страницы.

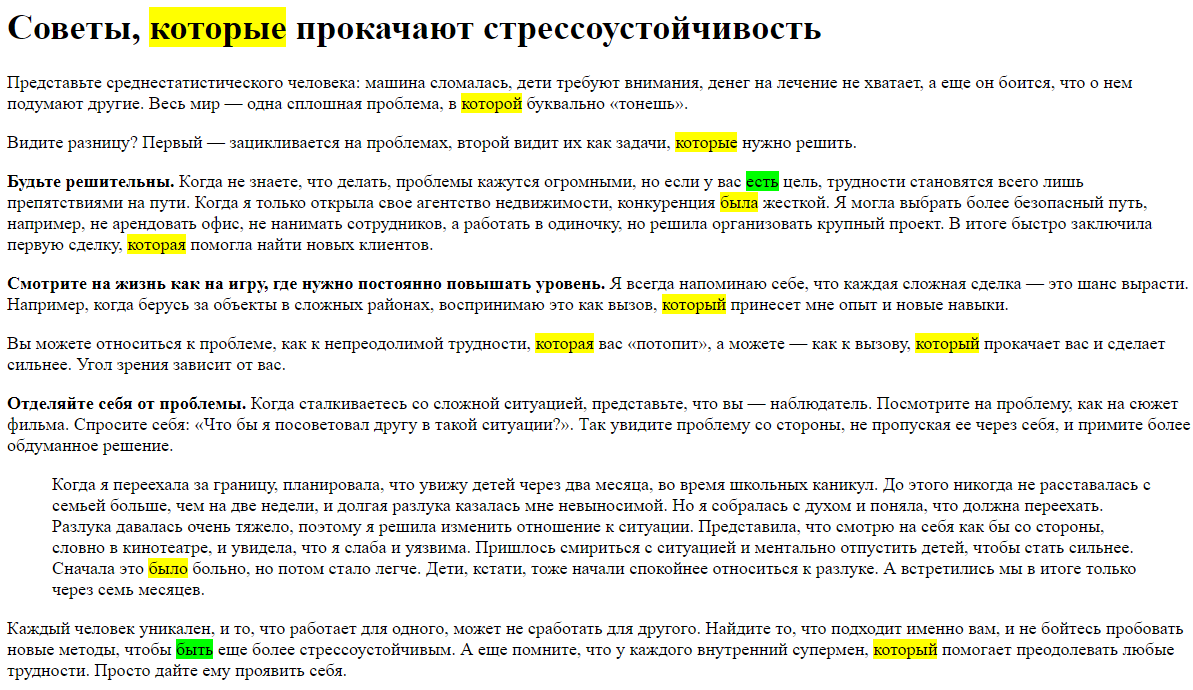


Рисунок 23. Топ-1 Релевантная страница

Рассмотрим вторую страницу. Расстояние между словами поискового запроса у нее 0.77, так как расстояние между словами – 11, что почти идентично первой странице.

Расположение от начала страницы 0.97, т.к. первое встречаемое слово поискового запроса также, как и в первом примере находится в заголовке страницы, но чуть дальше

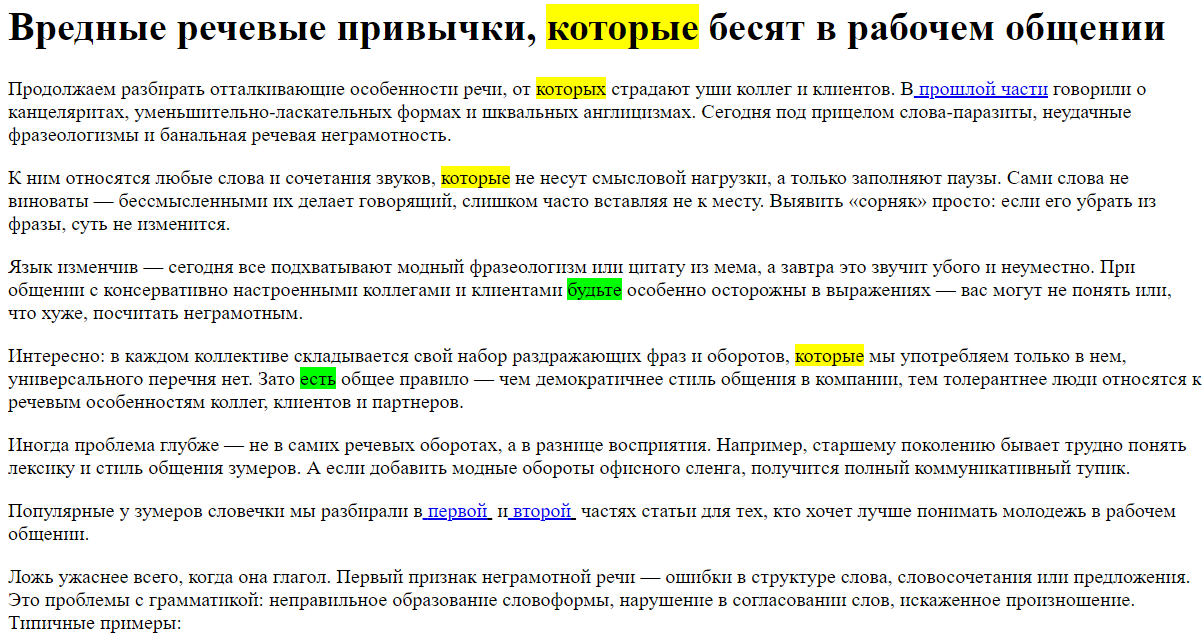


Рисунок 24. Топ-2 Релевантная страница

Рассмотрим третью страницу. Расстояние между словами поискового запроса у нее 0.96, что видно в последнем абзаце.

Расположение от начала страницы 1.0, т.к. первое встречаемое слово поискового запроса находится в самом начале документа.

Хоть данная страница и обладает лучшими метриками, она находится на третьем месте из-за учета ранжирования на основе внешних ссылок, где ее вес меньше первых двух страниц

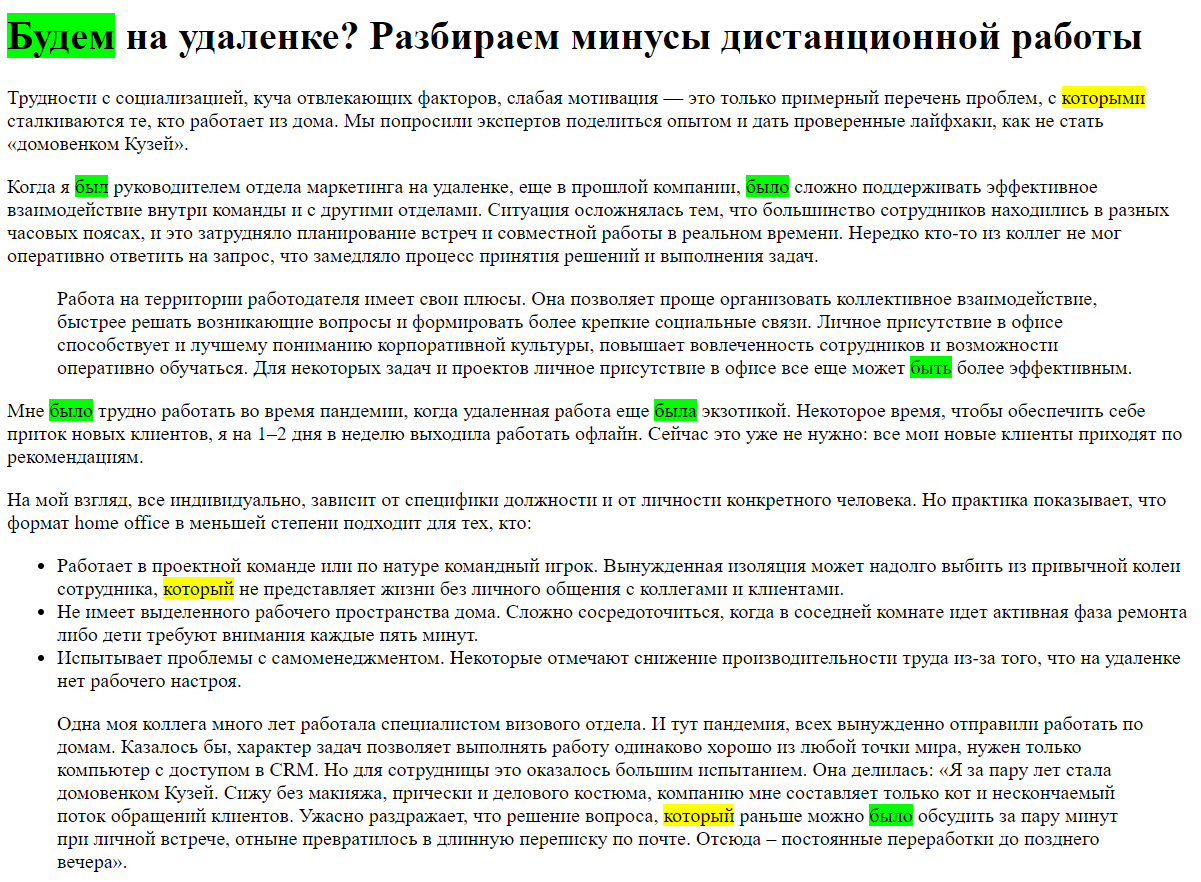


Рисунок 25. Топ-3 Релевантная страница

# Вывод

# По итогам проделанной работы был реализован модуль обработки поисковых запросов и выдачи результатов с ранжированием. Основные результаты работы:

# Спроектирована схема, описывающая структуру модуля поиска.

# Модуль разработан в соответствии с рекомендациями. Реализован метод search(), который осуществляет обрабатывает поисковый запрос, ранжирует страницы по их метрикам и на основе входящих ссылок и находит наиболее релевантные из них.

# Проведен анализ поискового запроса и его результата. Полученные результаты подтвердили корректность работы модуля, а также предоставили полезные аналитические данные о структуре веб-контента.

Разработка модуля обработки поисковых запросов позволила более точно понять устройство поисковых ботов, а также принципов на основе которых они выводят релевантные страницы

# Приложение

from bs4 import BeautifulSoup

import requests

import sqlite3

from pymystem3 import Mystem

import os

import re

# Заполнить новую таблицу рейтинга страниц

def fill\_db\_pageRank(db):

cur = db.cursor()

requestGet = f"SELECT id FROM db\_urlList"

urlList = cur.execute(requestGet).fetchall()

for url in urlList:

requestSet = f"INSERT INTO db\_pageRank (url\_id) VALUES ({url[0]})"

cur.execute(requestSet)

db.commit()

return

# Создать новую таблицу рейтинга страниц

def create\_db\_pageRank(db):

cur = db.cursor()

# cur.execute('DROP TABLE IF EXISTS db\_pageRank')

cur.execute("""CREATE TABLE IF NOT EXISTS db\_pageRank(

id INTEGER PRIMARY KEY AUTOINCREMENT,

url\_id INTEGER,

indexed DEFAULT 0,

score REAL DEFAULT 1.0

);""")

db.commit()

fill\_db\_pageRank(db)

return

# Поисковик

class Searcher:

# 0. Конструктор

def \_\_init\_\_(self):

self.path = os.path.dirname(os.path.abspath(\_\_file\_\_))

self.dbFileName = os.path.join(self.path, 'crawl.db')

self.db = sqlite3.connect(self.dbFileName)

# create\_db\_pageRank(self.db)

self.m = Mystem()

return

# 0. Деструктор

def \_\_del\_\_(self):

return

############################################################

######### Все о БД #########

############################################################

# Получить id слова по word в табл. wordlist

def getWordId(self, word):

cur = self.db.cursor()

requestGet = f"SELECT id FROM db\_wordList WHERE word = '{word}'"

word\_id = cur.execute(f"{requestGet}").fetchone()

if (word\_id == None):

raise Exception('The first word were not found')

return word\_id[0]

# Получить url по url\_id в табл. db\_urlList

def getUrlName(self, url\_id):

cur = self.db.cursor()

requestGet = f"SELECT url FROM db\_urlList WHERE id = '{url\_id}'"

url = cur.execute(f"{requestGet}").fetchone()

if (url == None):

raise Exception('The URL were not found')

return url[0]

# Получить score по url\_id в таблице db\_pageRank

def getPageRank(self, url\_id):

cur = self.db.cursor()

indexed = cur.execute(f"SELECT indexed FROM db\_pageRank WHERE url\_id = {url\_id}").fetchone()[0]

if(indexed==0):

score = 0

else:

score = cur.execute(f"SELECT score FROM db\_pageRank WHERE url\_id = {url\_id}").fetchone()[0]

return score

############################################################

######### Все о искомых Словах #########

############################################################

# Считать слова

def inputWords(self):

print("Enter Words")

words = input()

return words

# Преобразовать считанные слова в 1 род им. паддеж

def filterWords(self, words):

wordList = []

lemmas = self.m.lemmatize(words)

for lem in lemmas:

if ' ' not in lem:

wordList.append(lem)

print(wordList)

if len(wordList) != 3:

raise Exception('Please inter a another words')

return wordList[0], wordList[1]

# формирует таблицу со всеми сочетаниями искомых слов во всех url

def getMatchRows(self, fstWord, sndWord):

cur = self.db.cursor()

firstWord\_id = self.getWordId(fstWord)

sndWord\_id = self.getWordId(sndWord)

requestGet = f"SELECT "

requestGet += f"T1.fk\_url\_id, T1.location, T2.location "

requestGet += f"FROM db\_wordLocation AS T1 "

requestGet += f"INNER JOIN db\_wordLocation AS T2 ON T1.fk\_url\_id = T2.fk\_url\_id "

requestGet += f"WHERE T1.fk\_word\_id = '{firstWord\_id}' AND T2.fk\_word\_id = '{sndWord\_id}'"

DBList = cur.execute(f"{requestGet}").fetchall()

DBList.append((0, 0, 0))

self.printTable(fstWord, sndWord, DBList)

if (DBList == None):

raise Exception('The first URL list were not found')

newDBList = self.DBListToTrible(DBList)

return newDBList

# Форматируем текст - преобразуем в именительный паддеж 1 число, удаляем все союзы и тд

def textFilter(self, text):

textList = []

clearText = self.clearText(text)

lemmas = self.m.lemmatize(clearText)

for lem in lemmas:

if (' ' not in lem) and ('\n' not in lem):

textList.append(lem)

return textList, clearText

############################################################

######### Все о метриках #########

############################################################

# вычисление значения метрики отдаленности слов между друг другом на странице

def relPosMetrika(self, DBList):

relPosMetr = []

for page in DBList:

url = page[0][0]

minPos = abs(page[0][2] - page[0][1])

for row in page:

relPos = abs(row[2] - row[1])

if relPos < minPos:

minPos = relPos

relPosMetr.append((url, minPos))

return relPosMetr

# вычисление значения метрики отдаленности слов от начала страницы

def absPosMetrika(self, DBList):

absPosMetr = []

for page in DBList:

url = page[0][0]

fstWordMin = page[0][2]

sndWordMin = page[0][1]

absPosMetr.append((url, fstWordMin, sndWordMin))

return absPosMetr

############################################################

######### Все о нормализации #########

###########################################################

# Получить максимально возможное отдаление слова

def getMaxLocation(self, url\_id):

cur = self.db.cursor()

requestGet = f"SELECT MAX(location) FROM db\_wordLocation WHERE fk\_url\_id = {url\_id}"

maxLocation = cur.execute(f"{requestGet}").fetchone()[0]

return maxLocation + 1

# Нормализация значений метрик

def normalizeScores(self, absPosMetrika, relPosMetrika):

# Для absPosMetrika()

pagesM1 = []

for page in absPosMetrika:

url\_id = page[0]

maxLocation = self.getMaxLocation(url\_id)

rangeLocation = maxLocation - 1

fM1 = (maxLocation - page[1])/rangeLocation

sM1 = (maxLocation - page[2])/rangeLocation

M1 = (fM1 + sM1) / 2

pagesM1.append((url\_id, M1))

# Для relPosMetrika()

pagesM2 = []

for page in relPosMetrika:

url\_id = page[0]

maxLocation = self.getMaxLocation(url\_id)

rangeLocation = maxLocation - 1

M2 = (maxLocation - page[1]) / rangeLocation

pagesM2.append((url\_id, M2))

# Суммирование значений M1, M2, M3 и усреднение с учетом PageRank (ранга страницы)

pageMetriks = []

for i in range(len(pagesM1)):

url\_id = pagesM1[i][0]

url = self.getUrlName(url\_id)

M1 = pagesM1[i][1]

M2 = pagesM2[i][1]

pageRank = self.getPageRank(url\_id)

M3 = pageRank\*(M1 + M2)/2

pageMetriks.append((url\_id, round(M1, 2), round(M2, 2), round(M3, 4), url))

return pageMetriks

# Нормализация значений ранга страниц

def normalizeRank(self):

cur = self.db.cursor()

maxScore = cur.execute(f"SELECT MAX(score) FROM db\_pageRank").fetchone()[0]

urlList = cur.execute(f"SELECT id FROM db\_pageRank").fetchall()

for url in urlList:

mainUrl = url[0]

score = cur.execute(f"SELECT score FROM db\_pageRank WHERE id = {mainUrl}").fetchone()[0]

normolizeScore = round((score/maxScore), 2)

if(normolizeScore==0):

normolizeScore = 0.01

cur.execute(f"UPDATE db\_pageRank SET score = {normolizeScore} WHERE url\_id = {mainUrl}")

self.db.commit()

return

############################################################

######### Все о ранге страницы #########

############################################################

# Нахождение ранга страниц

def pageRank(self, iterations=5, d = 0.85):

cur = self.db.cursor()

requestGet = f"SELECT url\_id FROM db\_pageRank"

urlList = cur.execute(requestGet).fetchall()

for i in range(iterations):

print("Итерация %d" % (i+1))

for url in urlList:

mainUrl = url[0]

urlFromRefToMainList = cur.execute(f"SELECT fk\_fromURL\_id FROM db\_linkBetweenURL WHERE fk\_toURL\_id = {mainUrl}").fetchall()

urlP = 1 - d

if (urlFromRefToMainList):

for urlTo in urlFromRefToMainList:

score = self.getPageRank(urlTo[0])

if(score == 0):

PRef = cur.execute(f"SELECT score FROM db\_pageRank WHERE url\_id = {urlTo[0]}").fetchone()[0]

CRef = cur.execute(f"SELECT COUNT(id) FROM db\_linkBetweenURL WHERE fk\_fromURL\_id = {urlTo[0]}").fetchone()[0]

urlP += d \* (PRef/CRef)

cur.execute(f"UPDATE db\_pageRank SET score = {round(urlP, 2)} WHERE url\_id = {urlTo[0]}")

cur.execute(f"UPDATE db\_pageRank SET indexed = 1 WHERE url\_id = {urlTo[0]}")

self.db.commit()

else:

urlP += score

cur.execute(f"UPDATE db\_pageRank SET score = {round(urlP, 2)} WHERE url\_id = {mainUrl}")

cur.execute(f"UPDATE db\_pageRank SET indexed = 1 WHERE url\_id = {mainUrl}")

self.db.commit()

self.normalizeRank()

return

############################################################

######### Все о создании помеченной HTML #########

############################################################

# Нахождение пар (отфильтрованное слово - слово на странице)

def createMatch(self, fstWord, sndWord, pMain):

textList = []

filterTextList = []

clearTextList = []

for data in pMain:

if data.text == '':

continue

textList.append(data.text)

for i in range(len(textList)):

filteredText, clearText = self.textFilter(textList[i])

filterTextList.append(filteredText)

clearTextList.append(clearText)

matchList = []

for i in range(len(filterTextList)):

for j in range(len(filterTextList[i])):

if (filterTextList[i][j] == fstWord) or (filterTextList[i][j] == sndWord):

matchList.append(clearTextList[i].split()[j])

return matchList

# Создать страницу с помеченныит словами

def createMarkedHtmlFile(self, sortedMetrics, fstWord, sndWord):

urlList = []

for i in range(3):

urlList.append(sortedMetrics[i][4])

for i in range(len(urlList)):

print(f"Creating HTML file N{i+1}")

html\_doc = requests.get(urlList[i]).text # получить HTML-код страницы по текущему url

soup = BeautifulSoup(html\_doc, "html.parser")

pMain = soup.find('div', class\_='post\_\_content').find\_all(['p', 'blockquote'])

matchList = self.createMatch(fstWord, sndWord, pMain)

newHtml = f""

for word in html\_doc.split():

if word in matchList:

word = f"<span style='background-color: #FFFF00'>{word}</span>"

newHtml += f" {word} "

with open(f"page{i}.html", 'w', encoding="utf-8") as f:

f.write(newHtml)

return

############################################################

######### Доп. Функции #########

############################################################

# Вспомогательное преобразование в тройной массив

def DBListToTrible(self, DBList):

count = 0

for i in range(len(DBList) - 1):

if DBList[i][0] != DBList[i+1][0]:

count += 1

newArray = [list() for \_ in range(count)]

j = 0

for i in range(len(DBList) - 1):

newArray[j].append(DBList[i])

if DBList[i][0] != DBList[i+1][0]:

j += 1

return newArray

# Вывод таблицы 1 в консоль

def printTable(self, fstWord, sndWord, DBList):

print("----------------------------------------")

print(f"url\_ID\t| wordLoc1\t| wordLoc2")

counter = 0

for row in DBList:

print(f"{row[0]}\t|\t{row[1]}\t|\t{row[2]}")

counter += 1

if counter%10==0:

print("----------------------------------------")

print("Stop?\nEnter y/n")

key = input()

if key.lower()=='y':

break

print("----------------------------------------")

return

# Удалить все кроме букв и цифр

def clearText(self, text):

return re.sub(r'[\W ]+', ' ', text)

############################################################

######### Основные Функции #########

############################################################

# Непосредственно сам поиск

def search(self):

words = self.inputWords()

fstWord, sndWord = self.filterWords(words)

DBList = self.getMatchRows(fstWord, sndWord)

sortedMetrics = self.getSortedList(DBList)

self.createMarkedHtmlFile(sortedMetrics, fstWord, sndWord)

return

# Функция обертка для получения нахождения ранжированных url страниц

def getSortedList(self, DBList):

# self.pageRank()

absPosMetrika = self.absPosMetrika(DBList)

relPosMetrika = self.relPosMetrika(DBList)

pagesMetrics = self.normalizeScores(absPosMetrika, relPosMetrika)

sortedMetrics = sorted(pagesMetrics, key=lambda metric: metric[3], reverse=True)

print("\n----------------------------------------------------------------------------------------------------------------------------------")

print(f"url\_ID\tM1\tM2\tM3\tURL\_text")

for row in sortedMetrics:

print(f"{row[0]}\t{row[1]}\t{row[2]}\t{round(row[3]\*92, 2)}\t{row[4]}")

print("----------------------------------------------------------------------------------------------------------------------------------\n")

return sortedMetrics

def main():

searcher = Searcher()

searcher.search()

if \_\_name\_\_ == '\_\_main\_\_':

main()