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

##### «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

##### КАФЕДРА ВТ

##### ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1–2

##### «Разработка паука и модуля индексации 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 базы данных 4](#_bookmark1)

[Описание структуры разработанного паука 7](#_bookmark2)

[Результат мониторинга процесса индексации 15](#_bookmark2)

[Анализ проиндексированных таблиц 16](#_bookmark2)

[Вывод 19](#_bookmark2)

[Приложение 20](#_bookmark3)

# Постановка задачи

Задание.Реализовать модуль индексации поисковый системы включающий в себя: базу данных для хранения индексов страниц, процедуру обхода страниц и занесения их в базу данных.

# Описание структуры базы данных

# Для хранения полнотекстовых индексов документов была создана база данных на СУБД SQLite, состоящая из пяти таблиц:

# db\_urlList – это список проиндексированных URL.

# db\_wordList – список всех проиндексированных слов на страницах с признаком, следует ли это слово фильтровать при выдаче пользователю

# db\_wordLocation – список мест вхождения слов в документы.

# В оставшихся двух таблицах хранятся ссылки между документами:

# В таблице связей ссылок (db\_linkWord или db\_linkBetweenURL) хранятся идентификаторы двух URL, связанных ссылкой

# В таблице db\_linkWord есть два столбца – fk\_word\_id и fk\_link\_id – для хранения слов, составляющих ссылку.

# 

# Рисунок 1. Схема базы данных

# 1. Таблица db\_urlList (Список URL-адресов)

# id: INTEGER PRIMARY KEY AUTOINCREMENT — первичный ключ, автоматически увеличивающийся.

# url: TEXT — строка, содержащая URL-адрес.

# 2. Таблица db\_wordList (Список слов)

# id: INTEGER PRIMARY KEY AUTOINCREMENT — первичный ключ, автоматически увеличивающийся.

# word: TEXT — строка, содержащая слово.

# 3. Таблица db\_wordLocation (Местоположение слов в URL)

# id: INTEGER PRIMARY KEY AUTOINCREMENT — первичный ключ, автоматически увеличивающийся.

# fk\_word\_id: INTEGER — внешний ключ, ссылающийся на id из таблицы db\_wordList (идентификатор слова).

# fk\_url\_id: INTEGER — внешний ключ, ссылающийся на id из таблицы db\_urlList (идентификатор URL).

# location: INTEGER — числовое значение, указывающее местоположение слова на странице (например, позиция слова в тексте).

# 4. Таблица db\_linkWord (Связь между словами и ссылками)

# id: INTEGER PRIMARY KEY AUTOINCREMENT — первичный ключ, автоматически увеличивающийся.

# fk\_word\_id: INTEGER — внешний ключ, ссылающийся на id из таблицы db\_wordList (идентификатор слова).

# fk\_link\_id: INTEGER — внешний ключ, ссылающийся на идентификатор ссылки (в данном контексте может быть ссылка на URL-адрес или гиперссылку).

# 5. Таблица db\_linkBetweenURL (Связи между URL-адресами)

# id: INTEGER PRIMARY KEY AUTOINCREMENT — первичный ключ, автоматически увеличивающийся.

# fk\_fromURL\_id: INTEGER — внешний ключ, ссылающийся на id из таблицы db\_urlList (идентификатор исходного URL).

# fk\_toURL\_id: INTEGER — внешний ключ, ссылающийся на id из таблицы db\_urlList (идентификатор целевого URL).

# 

# Рисунок 2. Создание базы данных

# На рисунке 2 продемонстрирован код создания базы данных.

# Описание структуры разработанного паука

# Паук был разработан на языке программирования Python. Ниже показаны функции и пояснения к ним.

# 

# Рисунок 3. Функция createTables

# На рисунке 3 продемонстрирован функция создания базы данных.

# 

# Рисунок 4. Функция deleteTables

# На рисунке 4 продемонстрирован функция сброса базы данных до начального состояния.

# 

# Рисунок 5. Конструктор и деструктор

# На рисунке 5 изображен конструктор, который подключается к базе данных и вызывает функцию createTables, а также деструктор, где можно обнулить базу.

# 

# Рисунок 6. Функция addIndex.

# На рисунке 6 изображена функция addIndex в которой проверяется, была ли страница уже проиндексирована, если нет то извлекает текст со страницы и сохраняет их в базу данных.

# 

# Рисунок 7. Функция getTextOnly.

# На рисунке 7 изображена функция getTextOnly которая извлекает текстовый контент страницы, находя нужные html элементы и дальше вызывается фильтрация для удаления ненужных символов.

# 

# 

# Рисунок 8. Функции cleartext, separateWords, isIndexed.

# На рисунке 8 изображены функции cleartext, separateWords, isIndexed.

# Cleartext - удаляет все символы, кроме букв и цифр, из текста.

# separateWords - разделяет строку текста на отдельные слова с помощью функции clearText(), затем разбивает строку на слова и возвращает список.

# isIndexed - проверяет, была ли страница с данным URL уже проиндексирована. Проверяет наличие URL в таблице. Если URL есть в базе данных, проверяет наличие записей в таблице db\_wordLocation для этой страницы.

# 

# Рисунок 9. Функция addLinkRef.

# На рисунке 9 изображена функция addLinkRef которая добавляет информацию о ссылках между страницами. Получает идентификаторы для исходной и целевой страницы из таблицы db\_urlList. Добавляет запись в таблицу db\_linkBetweenURL, связывающую страницы. Сохраняет слова из текста ссылки в таблице db\_linkWord.

# 

# Рисунок 10. Функция crawl.

# На рисунке 10 изображена функция crawl - основная функция паука. Выполняет обход по списку URL до определенной глубины.

# 

# Рисунок 11. Функция getEntryId.

# На рисунке 11 изображена функция getEntryId - универсальная функция для получения ID записи в таблице или добавления новой записи, если её ещё нет. Возвращает идентификатор (ID) для использования в других таблицах.

# 

# Рисунок 12. Функция linkFilter.

# На рисунке 12 изображена функция linkFilter - извлекает и очищает ссылки на странице. Удаляет ненужные параметры из URL, добавляет базовый URL, если ссылка относительная, и фильтрует ненужные ссылки (например, на .doc файлы).

# 

# 

# Рисунок 13. Функции doubleArrayToString и textFilter.

# На рисунке 13 изображены функции doubleArrayToString и textFilter. doubleArrayToString - преобразует двумерный массив в одномерный, используется для обработки списка ссылок. textFilter - фильтрует текст, удаляя ненужные символы и преобразуя слова в леммы с помощью Mystem, также удаляет слова, содержащие цифры.

# 

# Рисунок 14. Функция main.

# На рисунке 14 изображена функция main которая инициализирует объект класса Crawler и запускает обход (crawling) веб-сайтов.

# Результат мониторинга процесса индексации

# Ниже на рисунке 15 изображен график поэтапного заполнения таблиц в базу данных. Желтая линия показывает количество записей в таблице wordlist, относительно шагов индексации. Она растет быстрее других, особенно после определенного времени, что указывает на резкий рост числа проиндексированных слов. Синяя линия показывает количество записей в таблице linkBetweenUrl, она растет более плавно относительно желтой. Красная линия представляет количество записей в таблице urlList. Рост этой линии самый медленный и практически линейный, что указывает на медленное добавление новых URL в базу данных по сравнению с другими таблицами.

# 

# Рисунок 15. График заполнения таблиц базы данных.

# Анализ проиндексированных таблиц

# Для того чтобы узнать количество записей в каждой таблице, двадцать наиболее часто проиндексированных доменов, двадцать наиболее часто встречающихся слов на странице были составлены Sql запросы в графическом клиенте DB Browser for SQLite.

# 

# Рисунок 16. Запрос количество записей в каждой таблице.

# 

# Рисунок 17. Запрос двадцать наиболее часто встречающихся слов на странице.

# 

# Рисунок 18. Запрос двадцать наиболее часто встречающихся слов на странице.

# Вывод

# По итогам проделанной работы был реализован паук и база данных для индексирования веб-страниц. Основные результаты работы:

# Спроектирована схема базы данных для хранения информации о проиндексированных страницах, найденных ссылках и словах на страницах

# Паук разработан в соответствии с рекомендациями. Реализован метод crawl(), который осуществляет обход страниц с определенного списка URL-адресов, а также заносит проиндексированные страницы в базу данных

# Проведен анализ содержимого таблиц базы данных. Были выявлены следующие результаты:

# Общее количество записей в каждой из таблиц: URL-адресов, слов, ссылок между страницами.

# Определены 20 наиболее часто проиндексированных доменов

# Выделены 20 наиболее часто встречающихся слов на страницах

# Разработка паука и базы данных для индексации веб-страниц позволила автоматизировать процесс сбора и анализа информации с веб-страниц. Полученные результаты подтвердили корректность работы паука и базы данных, а также предоставили полезные аналитические данные о структуре веб-контента.

# Приложение

# from bs4 import BeautifulSoup

# import requests

# import sqlite3

# import pymorphy3

# from pymystem3 import Mystem

# import re

# # Заполняем БД

# def createTables(dbFileName):

# cur = dbFileName.cursor()

# cur.execute("CREATE TABLE db\_urlList("

# "id INTEGER PRIMARY KEY AUTOINCREMENT, "

# "url TEXT)"

# )

# cur.execute("CREATE TABLE db\_wordList("

# "id INTEGER PRIMARY KEY AUTOINCREMENT, "

# "word TEXT)"

# )

# cur.execute("CREATE TABLE db\_wordLocation("

# "id INTEGER PRIMARY KEY AUTOINCREMENT, "

# "fk\_word\_id INTEGER, "

# "fk\_url\_id INTEGER, "

# "location INTEGER)"

# )

# cur.execute("CREATE TABLE db\_linkWord("

# "id INTEGER PRIMARY KEY AUTOINCREMENT, "

# "fk\_word\_id INTEGER, "

# "fk\_link\_id INTEGER)"

# )

# cur.execute("CREATE TABLE db\_linkBetweenURL("

# "id INTEGER PRIMARY KEY AUTOINCREMENT, "

# "fk\_fromURL\_id INTEGER, "

# "fk\_toURL\_id INTEGER)"

# )

# dbFileName.commit()

# return

# # Удаляем БД

# def deleteTables(dbFileName):

# cur = dbFileName.cursor()

# cur.execute("DROP TABLE db\_urlList")

# cur.execute("DROP TABLE db\_wordList")

# cur.execute("DROP TABLE db\_wordLocation")

# cur.execute("DROP TABLE db\_linkWord")

# cur.execute("DROP TABLE db\_linkBetweenURL")

# dbFileName.commit()

# return

# # Паук

# class Crawler:

# # 0. Конструктор Инициализация паука с параметрами БД

# def \_\_init\_\_(self, dbFileName, mainUrl):

# self.db = sqlite3.connect(dbFileName)

# createTables(self.db)

# self.mainUrl = mainUrl

# self.functors\_pos = {'INTJ', 'PRCL', 'CONJ', 'PREP'}

# self.m = Mystem()

# self.morth = pymorphy3.MorphAnalyzer()

# return

# # 0. Деструктор

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

# # deleteTables(self.db)

# return

# # 1. Индексирование одной страницы

# def addIndex(self, soup, url):

# if (self.isIndexed(url)): # если страница уже проиндексирована, то ее не индексируем isIndexed

# return

# print(f"Indexing: {url}")

# textList = self.getTextOnly(soup)

# if textList == None:

# return

# url\_id = self.getEntryId('db\_urlList', 'url', url) # Получаем идентификатор URL getEntryId

# location = 0

# for word in textList:

# location = + 1

# word\_id = self.getEntryId('db\_wordList', 'word', word)

# word\_location\_id = self.getEntryId('db\_wordLocation', 'fk\_url\_id', url\_id, 'fk\_word\_id', word\_id,

# 'location', location)

# return

# # 2. Получение текста страницы

# def getTextOnly(self, soup):

# textList = []

# if (soup.find('div', class\_='post\_\_content') == None):

# return

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

# for data in pMain:

# if data.text == '':

# continue

# textList.append(data.text)

# for i in range(len(textList)):

# textList[i] = self.textFilter(textList[i])

# newTextList = self.doubleArrayToSingle(textList)

# return newTextList

# # Удалить все кроме букв и цифр

# def clearText(self, text):

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

# # 3. Разбиение текста на слова

# def separateWords(self, text):

# newClearText = self.clearText(text)

# return newClearText.split()

# # 4. Проиндексирован ли URL (проверка наличия URL в БД)

# def isIndexed(self, url):

# cur = self.db.cursor()

# request = cur.execute(f"SELECT id FROM db\_urlList WHERE url = '{url}' ").fetchone()

# if (request != None):

# request = cur.execute(f"SELECT fk\_word\_id FROM db\_wordLocation WHERE fk\_url\_id = '{request}' ").fetchone()

# if (request != None):

# return True

# return False

# # 5. Добавление ссылки с одной страницы на другую

# def addLinkRef(self, urlFrom: str, urlTo: str, linkText: str):

# cur = self.db.cursor()

# urlFrom\_id = self.getEntryId('db\_urlList', 'url', urlFrom)

# urlTo\_id = self.getEntryId('db\_urlList', 'url', urlTo)

# link\_id = self.getEntryId('db\_linkBetweenURL', 'fk\_fromURL\_id', urlFrom\_id, 'fk\_toURL\_id', urlTo\_id)

# filteredTextLink = self.textFilter(linkText)

# for word in filteredTextLink:

# word\_id = self.getEntryId('db\_wordList', 'word', word)

# link\_word\_id = self.getEntryId('db\_linkWord', 'fk\_link\_id', link\_id, 'fk\_word\_id', word\_id)

# self.db.commit()

# # 6. Непосредственно сам метод сбора данных.

# # Начиная с заданного списка страниц, выполняет поиск в ширину

# # до заданной глубины, индексируя все встречающиеся по пути страницы

# def crawl(self, urlList, maxDepth=1):

# for currDepth in range(0, maxDepth):

# # # Вар.1. обход каждого url на текущей глубине

# newUrlList = []

# for url in urlList:

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

# soup = BeautifulSoup(html\_doc, "html.parser") # использовать парсер для работа тегов

# filteredLinks, textLinks = self.linkFilter(soup)

# for i in range(len(filteredLinks)):

# print(f"{url} -> {filteredLinks[i]}")

# self.addLinkRef(url, filteredLinks[i], textLinks[i])

# self.addIndex(soup, url) # вызвать функцию класса Crawler для добавления содержимого в индекс

# newUrlList.append(filteredLinks)

# urlList = self.doubleArrayToSingle(newUrlList)

# return

# # 7. Вспомогательная функция для получения идентификатора

# # добавления записи, если такой еще нет

# def getEntryId(self, tableName, fieldNameFirst, valueFirst, fieldNameSecond=False, valueSecond=False,

# fieldNameThird=False, valueThird=False):

# cur = self.db.cursor()

# fieldNameList = [fieldNameSecond, fieldNameThird]

# valueList = [valueSecond, valueThird]

# requestGet = f"SELECT id FROM '{tableName}' WHERE {fieldNameFirst} = '{valueFirst}'"

# for i in range(len(valueList)):

# if valueList[i]:

# requestGet += f" AND {fieldNameList[i]} = '{valueList[i]}'"

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

# if request\_id != None:

# return request\_id[0]

# requestInsert = f"INSERT INTO '{tableName}' ({fieldNameFirst}"

# for i in range(len(valueList)):

# if valueList[i]:

# requestInsert += f", {fieldNameList[i]}"

# requestInsert += f") VALUES ('{valueFirst}'"

# for i in range(len(valueList)):

# if valueList[i]:

# requestInsert += f", '{valueList[i]}'"

# requestInsert += f")"

# cur.execute(f"{requestInsert}")

# self.db.commit()

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

# return request\_id[0]

# # Форматируем ссылку - Очищаем ссылку от ненужный хуков и тд

# def linkFilter(self, soup):

# try:

# links = soup.find('main').find\_all('a', href=True)

# except:

# return [], []

# filteredLinks = []

# textLinks = []

# newfilteredLinks = []

# newtextLinks = []

# for data in links:

# filteredLinks.append(data.get("href"))

# for data in links:

# textLinks.append(data.text)

# for i in range(len(filteredLinks)):

# if textLinks[i] == '\n\n':

# continue

# if textLinks[i] == '':

# continue

# if '.doc' in filteredLinks[i]:

# continue

# if 'https' not in filteredLinks[i]:

# filteredLinks[i] = self.mainUrl + filteredLinks[i]

# if '?' in filteredLinks[i]:

# filteredLinks[i] = filteredLinks[i].split('?', 1)[0]

# newfilteredLinks.append(filteredLinks[i])

# newtextLinks.append(textLinks[i])

# return newfilteredLinks, newtextLinks

# # Двумерный массив в одномерный

# def doubleArrayToSingle(self, doubleArray):

# singleArray = []

# for arr in doubleArray:

# for element in arr:

# singleArray.append(element)

# return singleArray

# def textFilter(self, text):

# newText = []

# textList = []

# clearText = self.clearText(text)

# lemmas = self.m.lemmatize(clearText)

# for lem in lemmas:

# if ' ' not in lem:

# textList.append(lem)

# for word in textList:

# try:

# # Удаляем слова, содержащие цифры

# if not any(char.isdigit() for char in word): #просматривает полностью слово и удаляет цифры если они там содержаться

# newText.append(word)

# except Exception as e:

# continue

# return newText

# def main():

# urlLenta = 'https://lenta.ru'

# urlListLenta = [urlLenta]

# for i in range(1, 3):

# url\_page = urlLenta + '/page/' + str(i)

# urlListLenta.append(url\_page)

# # URL для wikipedia.org

# urlWikipedia = 'https://wikipedia.org'

# urlListWikipedia = [urlWikipedia]

# crawler = Crawler('database.db', urlLenta) # инициализация базы данных

# crawler.crawl(urlListLenta, maxDepth=2) # Сначала обходим lenta.ru

# # После обхода lenta.ru, обходим wikipedia.org

# crawler.crawl(urlListWikipedia, maxDepth=2)

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

# main()