МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Отчет по лабораторной работе №2

по предмету «Информационный поиск»

«Поисковый робот»

Группа: М8О-412Б-22

Студент(ка): Кайдалова А. А.

Оценка:

Дата сдачи:

Москва, 2025

Задание

Необходимо написать парсер на любом языке программирования.

- Написать поисковый робот — компоненты обкачки документов, используя любой язык программирования;
- Единственным аргументом поисковому роботу подаётся путь до yaml-конфига, содержащий:
  - Данные для базы данные в секции db;
  - Данные для робота в секции logic: задержка между обкачкой страницы;
  - Любые другие данные, необходимые для реализовации логики поискового робота.
- Сохранять в базе данных (например, MongoDB) документы со следующими полями:
  - url, нормализованный;
  - «сырой» html-текст документа;
  - название источника;
  - Дата обкачки документа в формате Unix time stamp.
- Поисковый робот можно остановить в любой момент и при повторном запуске робот должен начать с того документа, с которого он остановился;
- Периодически он должен уметь переобкачивать документы, которые уже есть в базе, но только в том случае, если они изменились.

Описание метода решения задачи

Я реализовала поискового робота на языке Python.

Хранение данных: PostgreSQL. Таблица documents содержит все требуемые поля, также метаданные: заголовок, автор, дата публикации, и служебные поля last_fetch_attempt, etag_header для управления обновлениями. DDL:

Начальный обход начинается с заданных стартовых URL сайта 7ya.ru, также используются sitemapы сайтов mama.ru и letidor.ru.

Обход документов реализован на основе очереди, в которую изначально добавляются стартовые URL и ссылки полученные из sitemapов. При обработке каждой ссылки из очереди программа:

1. Загружает HTML-документ с заданной задержкой (1 сек);
2. Извлекает из него внутренние ссылки;
3. Добавляет новые подходящие ссылки в конец очереди;

4. Фильтрует URL.

Сохранение состояния: при каждом цикле обхода состояние (очередь, набор уже добавленных URL, счётчик сохранённых документов) сериализуется в файл crawler_state.pkl.

Фильтрация URL: реализована функция is_valid_article_link(), которая допускает только статьи например, /article/ на 7ya.ru, /articles/ на mama.ru, исключая категории, форумы, профили.

Извлечение текста: с помощью BeautifulSoup удаляются навигационные блоки, извлекается только полезный текст статьи. Для некоторых сайтов используется структурированный JSON-LD.

Обновление документов: после первичного сбора документов запускается фаза обновления. Документы, не обновлявшиеся последние 7 дней, проверяются на изменения с помощью условных HTTP-запросов и хэширования по полям title, author, date, text.

Вежливость: соблюдение robots.txt с кэшированием на 24 часа, задержка 0.5 сек между запросами, корректный User-Agent.

Журнал выполнения задания

Реализация фильтрации URL. На mama.ru в sitemapах много ссылок на категории и форумы. Добавила проверку: только пути /articles/... без /category/. На letidor.ru разрешила только .html/.htm, на 7ya.ru — только /article/....

Нормализация URL. Изначально один и тот же документ мог сохраниться несколько раз из-за различий в http/https, www и конечных /. Реализовала функцию normalize_document_url(), приводящую все URL к единому виду.

Извлечение текста и метаданных. Написала отдельные блоки для каждого сайта, отличительные черты:

- 7ya.ru — текст в .articlebody;
- mama.ru — отдельно лид и основной блок;
- letidor.ru — дата в формате 12 декабря 2024 преобразована через словарь месяцев.

Обработка ошибок 404. Некоторые страницы возвращали статус 200, но содержали текст «страница не найдена». Реализовала функцию is_bad_content(), проверяющую <title>, <h1> и общий текст на наличие «страница не найдена» или «404».

Обеспечение отказоустойчивости. После каждого обработанного URL (для того чтобы можно было возобновить работу именно с ссылки. На которой закончили) состояние сохраняется в crawler_state.pkl. При Ctrl+C программа корректно завершает работу, при повторном запуске возобновляет обход.

Реализация фазы обновления. После сбора 54 543 документов запускается обновление:

1. выбор документов, не обновлявшихся 7+ дней;
2. условный запрос с If-Modified-Sinc, If-None-Match;
3. при изменении - пересчёт хэша по title, author, date, text и обновление записи.

Результаты

За время выполнения работы собрано 54 543 уникальных статей из трёх источников:

	source	article_count
1	7ya.ru	8 417
2	letidor.ru	40 685
3	mama.ru	5 441

Все документы прошли фильтрацию по содержимому и тематике. Программа успешно возобновляла работу после пяти принудительных остановок. Поограмма корректно искала обновившееся документы (для теста брала разницу в 10 минут).

Выводы

Работа выполнена в полном соответствии с требованиями. Реализован надёжный, вежливый и отказоустойчивый поисковый робот, способный собирать и обновлять корпус документов.

Недостатки и пути улучшения:

- Производительность: задержка 1 секунда, хоть и не большая, но сильно замедляет сбор. Улучшение: можно динамически регулировать задержку в зависимости от ответа сервера или использовать асинхронные запросы.
- Зависимость от структуры HTML: при изменении верстки сайта парсер может сломаться. Улучшение: добавить fallback-механизмы и логирование ошибок извлечения текста.

Несмотря на указанные недостатки, программа стабильно работает, корректно сохраняет состояние и обеспечивает хорошее качество собранного корпуса.

Приложения

Исходный код: crawler.py, config.yaml, .env (см. ниже).
Требования к запуску:

- PostgreSQL с созданной таблицей documents:

```
CREATE TABLE public.documents (
	id bigserial NOT NULL,
	url text NOT NULL,
	normalized_url text NOT NULL,
	"source" text NOT NULL,
	created_at timestamptz DEFAULT now() NULL,
	html text NULL,
	clean_text text NULL,
	title text NULL,
	author text NULL,
	publish_date text NULL,
	last_modified_header text NULL,
	etag_header text NULL,
	fetch_timestamp int8 NULL,
	last_fetch_attempt int8 NULL,
	CONSTRAINT documents_normalized_url_key UNIQUE (normalized_url),
	CONSTRAINT documents_pkey PRIMARY KEY (id)
);
CREATE INDEX idx_last_fetch ON public.documents USING btree (last_fetch_attempt);
CREATE UNIQUE INDEX idx_normalized_url ON public.documents USING btree
(normalized_url);
```

- Переменная окружения DB_PASSWORD

Инструкция по запуску: python crawler.py config.yaml

config.yaml:

```yaml
db:
  host: "localhost"
  port: 5432
  database: "search_corpus"
  user: "postgres"

logic:
  delay_seconds: 1
  user_agent: "MAI-SearchBot/1.0 (student project; contact: alexkayd@list.ru)"
  respect_robots_txt: true
  use_mama_sitemap: true
  use_letidor_sitemap: true
  max_documents: 60000

  sources:
    - name: "7ya"
      start_urls:
        - "https://www.7ya.ru/article/6-navykov-bezopasnosti-kotorym-nuzhno-nauchit-malysha-2-3-let/"
        - "https://www.7ya.ru/article/Mama-pyati-detej-kazhdyj-zasypaet-v-svoej-krovati/"
        - "https://www.7ya.ru/article/Kak-nauchit-rebenka-ostavatsya-bez-mamy/"
        - "https://www.7ya.ru/article/Kak-perevesti-rebenka-na-odin-dnevnoj-son-lajfhaki-dlya-mam/"
        - "https://www.7ya.ru/article/Kak-podgotovit-rebenka-k-detskomu-sadu-nachnite-s-sebya/"

    - name: "mama"
      start_urls: []

    - name: "letidor"
      start_urls: []
```

crawler.py:

```python
import sys
import os
import pickle
import time
import signal
import hashlib
import yaml
import requests
import json
import re
from urllib.parse import urljoin, urlparse, urlunparse
from urllib.robotparser import RobotFileParser
from bs4 import BeautifulSoup
from dotenv import load_dotenv
import psycopg2
from collections import deque

load_dotenv()

def connect_to_database(config):
    db_conf = config['db']
    return psycopg2.connect(
        host=db_conf['host'],
        port=db_conf['port'],
        database=db_conf['database'],
        user=db_conf['user'],
        password=os.getenv('DB_PASSWORD')
    )

def is_valid_article_link(url):
```

```
    parsed = urlparse(url)
    path = parsed.path.lower()

    if "7ya.ru" in parsed.netloc and path.startswith("/article/"):
        return True
    if "mama.ru" in parsed.netloc and path.startswith("/articles/"):
        if "/category/" in path:
            return False
        return True
    if "letidor.ru" in parsed.netloc:
        if path.endswith('.htm') or path.endswith('.html'):
            return True
        return False
    return False

def get_source_name_from_url(url):
    parsed = urlparse(url)
    netloc = parsed.netloc.lower().rstrip('.')
    if netloc == '7ya.ru' or netloc.endswith('.7ya.ru'):
        return "7ya.ru"
    if netloc == 'mama.ru' or netloc.endswith('.mama.ru'):
        return "mama.ru"
    if netloc == 'letidor.ru' or netloc.endswith('.letidor.ru'):
        return "letidor.ru"
    return "unknown"

def store_document(conn, original_url, normalized_url, html_content, clean_text, title, author,
publish_date, source_name, fetch_time, last_mod, etag):
    with conn.cursor() as cur:
        cur.execute("""
            INSERT INTO documents (
                url, normalized_url, html, clean_text, title, author, publish_date,
                source, fetch_timestamp, last_modified_header, etag_header, last_fetch_attempt
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            ON CONFLICT (normalized_url) DO UPDATE
                SET
                    html = EXCLUDED.html,
                    clean_text = EXCLUDED.clean_text,
                    title = EXCLUDED.title,
                    author = EXCLUDED.author,
                    publish_date = EXCLUDED.publish_date,
                    source = EXCLUDED.source,
                    last_modified_header = EXCLUDED.last_modified_header,
                    etag_header = EXCLUDED.etag_header,
                    last_fetch_attempt = EXCLUDED.last_fetch_attempt
        """, (original_url, normalized_url, html_content, clean_text, title, author, publish_date,
              source_name, fetch_time, last_mod, etag, fetch_time))
        conn.commit()

def extract_internal_links(page_html, base_url):
    soup = BeautifulSoup(page_html, 'html.parser')
    found_links = set()
    for a_tag in soup.find_all('a', href=True):
        href = a_tag['href'].strip()
        if not href or href.startswith(('#', 'mailto:', 'tel:', 'javascript:', '/tags', '/search',
'/user', '/profile', '/comment')):
            continue
        full_url = urljoin(base_url, href)
        normalized = normalize_document_url(full_url)
        if not is_valid_article_link(normalized):
            continue
        if urlparse(normalized).netloc == urlparse(base_url).netloc:
            found_links.add(normalized)
    return found_links

def normalize_document_url(url, base_url=None):
    if base_url:
        url = urljoin(base_url, url)
    parsed = urlparse(url)
```

```
    scheme = parsed.scheme.lower()
    netloc = parsed.netloc.lower()
    path = parsed.path or '/'
    return urlunparse((scheme, netloc, path, '', '', ''))

def is_document_already_saved(conn, normalized_url):
    with conn.cursor() as cur:
        cur.execute("SELECT 1 FROM documents WHERE normalized_url = %s", (normalized_url,))
        return cur.fetchone() is not None

def compute_document_hash(title, author, publish_date, clean_text):
    combined = f"{title}|{author}|{publish_date}|{clean_text}"
    return hashlib.md5(combined.encode('utf-8', errors='ignore')).hexdigest()

def extract_article_text(html, source_url):
    soup = BeautifulSoup(html, 'html.parser')
    for tag in soup(["script", "style", "nav", "footer", "aside", "header"]):
        tag.decompose()

    try:
        for script in soup.find_all("script", type="application/ld+json"):
            data = json.loads(script.string)
            if isinstance(data, dict) and "articleBody" in data:
                return data["articleBody"].strip()
    except (ValueError, KeyError, TypeError):
        pass

    content = None

    if "7ya.ru" in source_url:
        content = soup.select_one('.articlebody')
    elif "mama.ru" in source_url:
        lead = soup.select_one('.article-lead--text.mt-2')
        lead_text = ""
        if lead:
            p_tag = lead.find('p')
            if p_tag:
                lead_text = p_tag.get_text(strip=True)
        article_block = soup.select_one('.article-block')
        main_text = ""
        if article_block:
            for tag in article_block(["script", "style", "nav", "footer", "aside", "header"]):
                tag.decompose()
            main_text = ' '.join(article_block.get_text().split())
        full_text = ' '.join([lead_text, main_text]).strip()
        return full_text if full_text else ""
    elif "letidor.ru" in source_url:
        texts = soup.select('div[data-qa="text"]')
        if texts:
            full_text = ' '.join(t.get_text().strip() for t in texts)
            return full_text
    else:
        content = soup

    return ' '.join(content.get_text().split()) if content else ''

def extract_article_metadata(html, source_url):
    soup = BeautifulSoup(html, 'html.parser')
    title = ""
    author = ""
    publish_date = ""

    try:
        for script in soup.find_all("script", type="application/ld+json"):
            data = json.loads(script.string)
            if isinstance(data, dict):
                title = data.get("headline", "").strip()
                author_obj = data.get("author", {})
                if isinstance(author_obj, dict):
```

```
                    author = author_obj.get("name", "").strip()
                elif isinstance(author_obj, str):
                    author = author_obj.strip()
                dt = data.get("datePublished", "")
                if dt:
                    publish_date = dt.split("T")[0]
                if title and author and publish_date:
                    return title, author, publish_date
    except (ValueError, KeyError, TypeError):
        pass

    if "7ya.ru" in source_url:
        title_tag = soup.select_one('h1[itemprop="headline"]')
        author_tag = soup.select_one('div.type a[href*="club.7ya.ru"]')
        date_tag = soup.find('meta', {'itemprop': 'datePublished'})
        title = title_tag.get_text(strip=True) if title_tag else ""
        author = author_tag.get_text(strip=True) if author_tag else ""
        publish_date = date_tag['content'] if date_tag and date_tag.get('content') else ""

    elif "mama.ru" in source_url:
        title_tag = soup.select_one('h1[itemprop="headline"]')
        author_tag = soup.select_one('h5.fw-semibold.small-text-on-mobile')
        date_meta = soup.find('meta', {'itemprop': 'datePublished'})
        publish_date = ""
        if date_meta and date_meta.get('content'):
            publish_date = date_meta['content'].split('T')[0]
        title = title_tag.get_text(strip=True) if title_tag else ""
        author = author_tag.get_text(strip=True) if author_tag else ""
    elif "letidor.ru" in source_url:
        title_tag = soup.select_one('h1[data-qa="lb-topic-header-texts-title"]')
        title = title_tag.get_text(strip=True) if title_tag else ""
        author_tag = soup.select_one('div.jsx-1196406913 a.link')
        author = author_tag.get_text(strip=True) if author_tag else ""
        date_div = soup.select_one('div.J57431KZ.M7gu8GJc')
        if date_div:
            date_text = date_div.get_text().strip()
            match = re.search(r'(\d{1,2})\s+([а-яА-Я]+)\s+(\d{4})', date_text)
            if match:
                day, month_ru, year = match.groups()
                months = {
                    "января": "01", "февраля": "02", "марта": "03", "апреля": "04",
                    "мая": "05", "июня": "06", "июля": "07", "августа": "08",
                    "сентября": "09", "октября": "10", "ноября": "11", "декабря": "12"
                }
                month = months.get(month_ru, "01")
                publish_date = f"{year}-{month}-{day.zfill(2)}"
    return title, author, publish_date

def is_bad_content(html):
    soup = BeautifulSoup(html, 'html.parser')
    title_tag = soup.find('title')

    if title_tag:
        title_text = title_tag.get_text().lower()
        if "404" in title_text or "страница не найдена" in title_text:
            return True
    h1_tag = soup.find('h1')

    if h1_tag:
        h1_text = h1_tag.get_text().lower()
        if "404" in h1_text or "страница не найдена" in h1_text:
            return True
    body_text = soup.get_text()
    words = body_text.split()

    if len(words) < 10:
        if "404" in body_text.lower() or "страница не найдена" in body_text.lower():
            return True
    return False
```

```python
_robots_cache = {}
_robots_cache_expires = {}
_ROBOTS_CACHE_TTL = 86400

def is_allowed_by_robots(url, user_agent):
    parsed = urlparse(url)
    scheme = parsed.scheme or 'https'
    netloc = parsed.netloc
    if not netloc:
        return False
    domain_base = f"{scheme}://{netloc}"
    now = time.time()
    if domain_base in _robots_cache_expires:
        if now - _robots_cache_expires[domain_base] > _ROBOTS_CACHE_TTL:
            _robots_cache.pop(domain_base, None)
            _robots_cache_expires.pop(domain_base, None)
    if domain_base not in _robots_cache:
        try:
            parser = RobotFileParser()
            parser.set_url(f"{domain_base}/robots.txt")
            parser.read()
            _robots_cache[domain_base] = parser
            _robots_cache_expires[domain_base] = now
        except Exception as e:
            print(f"Не удалось загрузить robots.txt для {domain_base}: {e}")
            _robots_cache[domain_base] = None
    rp = _robots_cache[domain_base]
    if rp is None:
        return True
    return rp.can_fetch(user_agent, url)

def load_sitemap_urls(sitemap_url, session, timeout=10):
    try:
        resp = session.get(sitemap_url, timeout=timeout)
        if resp.status_code != 200:
            return []
        soup = BeautifulSoup(resp.content, 'xml')
        urls = []
        for loc in soup.find_all('loc'):
            url = loc.get_text().strip()
            if url:
                urls.append(url)
        return urls
    except Exception as e:
        print(f"Ошибка при загрузке sitemap {sitemap_url}: {e}")
        return []

def main(config_path):
    with open(config_path, 'r', encoding='utf-8') as f:
        config = yaml.safe_load(f)

    logic = config['logic']
    user_agent = logic['user_agent']
    delay_sec = logic['delay_seconds']
    max_docs_limit = logic.get('max_documents', 100)
    state_file_path = "crawler_state.pkl"
    db_connection = None

    def handle_interrupt(sig, frame):
        print("\nСостояние сохранено.")
        if db_connection:
            db_connection.close()
        sys.exit(0)
    signal.signal(signal.SIGINT, handle_interrupt)

    try:
        db_connection = connect_to_database(config)
        print("Подключение к бд успешно")
```

```
    except Exception as e:
        print(f"Ошибка подключения к бд: {e}")
        sys.exit(1)

    session = requests.Session()
    session.headers.update({'User-Agent': user_agent})

    with db_connection.cursor() as cur:
        cur.execute("SELECT COUNT(*) FROM documents")
        actual_count = cur.fetchone()[0]

    if actual_count >= max_docs_limit:
        print(f"\nЗапускаю обновление")
        run_update_phase(db_connection, max_docs_limit, user_agent, delay_sec)
        db_connection.close()
        print("\nОбновление завершено.")
        return

    document_queue = deque()
    seen_in_queue = set()
    saved_count = 0

    if os.path.exists(state_file_path):
        print("\nВосстанавливаю состояние")
        with open(state_file_path, 'rb') as f:
            state = pickle.load(f)
            document_queue = deque(state['queue'])
            seen_in_queue = set(state.get('seen_in_queue', []))
            saved_count = state['saved']
        print(f"Очередь: {len(document_queue)} URL, сохранено: {saved_count}")
    else:
        for source in logic['sources']:
            for url in source['start_urls']:
                url = url.strip()
                if not url:
                    continue
                normalized = normalize_document_url(url)
                if not is_allowed_by_robots(normalized, user_agent):
                    print(f"Запрещено robots.txt: {normalized}")
                    continue
                if is_document_already_saved(db_connection, normalized):
                    print(f"Уже есть в бд: {normalized}")
                    continue
                if normalized not in seen_in_queue:
                    document_queue.append((normalized, source['name']))
                    seen_in_queue.add(normalized)

        if logic.get('use_mama_sitemap', False):
            sitemap_urls = load_sitemap_urls("https://mama.ru/sitemap.xml", session)
            if not sitemap_urls:
                print("Не удалось загрузить основной sitemap mama.ru.")
            else:
                post_sitemaps = [u for u in sitemap_urls if 'post-sitemap' in u]
                added = 0
                for sitemap_url in post_sitemaps:
                    for url in load_sitemap_urls(sitemap_url, session):
                        norm = normalize_document_url(url)
                        if is_valid_article_link(norm) and norm not in seen_in_queue:
                            document_queue.append((norm, "mama.ru"))
                            seen_in_queue.add(norm)
                            added += 1
                print(f"Добавлено из sitemap mama.ru: {added} URL")

        if logic.get('use_letidor_sitemap', False):
            sitemap_urls = load_sitemap_urls("https://letidor.ru/sitemap.xml", session)
            if not sitemap_urls:
                print("Не удалось загрузить основной sitemap letidor.ru.")
            else:
                article_sitemaps = []
```

```
                for u in sitemap_urls:
                    if 'sitemap' in u and ('main' in u or 'article' in u or 'post' in u):
                        article_sitemaps.append(u)
                added = 0
                for sitemap_url in article_sitemaps:
                    for url in load_sitemap_urls(sitemap_url, session):
                        norm = normalize_document_url(url)
                        if is_valid_article_link(norm) and norm not in seen_in_queue:
                            document_queue.append((norm, "letidor.ru"))
                            seen_in_queue.add(norm)
                            added += 1
                print(f"Добавлено из sitemap letidor.ru: {added} URL")

    print(f"\nНачинаю обход")

    while document_queue and saved_count < max_docs_limit:
        normalized_url, source_name = document_queue.popleft()
        seen_in_queue.discard(normalized_url)

        if not is_allowed_by_robots(normalized_url, user_agent):
            print(f"Запрещено robots.txt: {normalized_url}")
            continue

        print(f"Обрабатываю ({saved_count + 1}/{max_docs_limit}): {normalized_url}")

        try:
            response = session.get(normalized_url, timeout=10, allow_redirects=True)
        except Exception as e:
            print(f"Ошибка при запросе: {e}")
            continue

        if response.status_code == 404:
            print(f"Страница не найдена (404): {normalized_url}")
            continue
        elif response.status_code != 200:
            print(f"Пропущен (HTTP {response.status_code}): {normalized_url}")
            continue

        final_url = response.url
        new_normalized_url = normalize_document_url(final_url)

        if not is_valid_article_link(final_url):
            print(f"Редирект на недопустимый источник: {final_url}")
            continue

        if new_normalized_url != normalized_url:
            if is_document_already_saved(db_connection, new_normalized_url):
                print(f"Редирект на уже существующий документ: {normalized_url} на
{new_normalized_url}")
                continue

        if is_document_already_saved(db_connection, new_normalized_url):
            print(f"Уже есть после редиректа: {new_normalized_url}")
            continue

        fetch_timestamp = int(time.time())
        last_modified = response.headers.get('Last-Modified')
        etag_value = response.headers.get('ETag')

        clean_text = extract_article_text(response.text, final_url)
        title, author, publish_date = extract_article_metadata(response.text, final_url)
        new_source_name = get_source_name_from_url(final_url)

        if is_bad_content(response.text):
            print(f"Пропущена (плохой контент): {new_normalized_url}")
            continue

        if not clean_text.strip():
            print(f"Пропущена (пустой текст): {new_normalized_url}")
```

```
            continue

        store_document(db_connection, normalized_url, new_normalized_url, response.text, clean_text,
                       title, author, publish_date, new_source_name, fetch_timestamp, last_modified,
etag_value)
        saved_count += 1

        if saved_count < max_docs_limit:
            new_links = extract_internal_links(response.text, final_url)
            for link in new_links:
                if is_allowed_by_robots(link, user_agent):
                    if not is_document_already_saved(db_connection, link) and link not in
seen_in_queue:
                        document_queue.append((link, get_source_name_from_url(link)))
                        seen_in_queue.add(link)

        with open(state_file_path, 'wb') as state_file:
            pickle.dump({
                'queue': list(document_queue),
                'seen_in_queue': list(seen_in_queue),
                'saved': saved_count
            }, state_file)

        time.sleep(delay_sec)

    print(f"\nСохранено {saved_count} документов.")
    print(f"\nЗапускаю обновление")
    run_update_phase(db_connection, max_docs_limit, user_agent, delay_sec)

    db_connection.close()
    if os.path.exists(state_file_path):
        os.remove(state_file_path)
    print("\nОбновление завершено.")


# обновление
def run_update_phase(db_connection, max_docs_limit, user_agent, delay_sec):
    refresh_interval_sec = 604800
    batch_idx = 0
    total_updated = 0

    while True:
        current_time = int(time.time())
        with db_connection.cursor() as cur:
            cur.execute("""
                SELECT url, normalized_url, last_modified_header, etag_header,
                       clean_text, title, author, publish_date, fetch_timestamp
                FROM documents
                WHERE last_fetch_attempt < %s
                ORDER BY last_fetch_attempt ASC
                LIMIT 100
            """, (current_time - refresh_interval_sec,))
            documents_for_refresh = cur.fetchall()

        if not documents_for_refresh:
            break

        batch_idx += 1
        print(f"\nПакет {batch_idx}")

        updated_in_batch = 0
        for doc in documents_for_refresh:
            orig_url, norm_url, saved_last_mod, saved_etag, current_clean_text, current_title,
current_author, current_publish_date, original_fetch_time = doc

            headers = {'User-Agent': user_agent}
            if saved_last_mod:
                headers['If-Modified-Since'] = saved_last_mod
```

```
            if saved_etag:
                headers['If-None-Match'] = saved_etag

            try:
                resp = requests.get(orig_url, headers=headers, timeout=10, allow_redirects=True)
            except Exception as e:
                print(f"Ошибка при проверке обновления: {e}")
                with db_connection.cursor() as cur:
                    cur.execute("UPDATE documents SET last_fetch_attempt = %s WHERE normalized_url =
%s", (current_time, norm_url))
                db_connection.commit()
                time.sleep(delay_sec)
                continue

            if resp.status_code == 304:
                with db_connection.cursor() as cur:
                    cur.execute("UPDATE documents SET last_fetch_attempt = %s WHERE normalized_url =
%s", (current_time, norm_url))
                db_connection.commit()
                print(f"Не изменилась: {norm_url}")
                time.sleep(delay_sec)
                continue

            if resp.status_code != 200:
                with db_connection.cursor() as cur:
                    cur.execute("UPDATE documents SET last_fetch_attempt = %s WHERE normalized_url =
%s", (current_time, norm_url))
                db_connection.commit()
                time.sleep(delay_sec)
                continue

            final_url = resp.url
            new_norm_url = normalize_document_url(final_url)

            if not is_valid_article_link(final_url):
                print(f"Редирект на недопустимый источник при обновлении: {final_url}")
                with db_connection.cursor() as cur:
                    cur.execute("UPDATE documents SET last_fetch_attempt = %s WHERE normalized_url =
%s", (current_time, norm_url))
                db_connection.commit()
                time.sleep(delay_sec)
                continue

            new_clean_text = extract_article_text(resp.text, final_url)
            new_title, new_author, new_publish_date = extract_article_metadata(resp.text, final_url)
            new_source_name = get_source_name_from_url(final_url)

            old_hash = compute_document_hash(current_title, current_author, current_publish_date,
current_clean_text)
            new_hash = compute_document_hash(new_title, new_author, new_publish_date, new_clean_text)

            if old_hash == new_hash:
                with db_connection.cursor() as cur:
                    cur.execute("UPDATE documents SET last_fetch_attempt = %s WHERE normalized_url =
%s", (current_time, norm_url))
                db_connection.commit()
                print(f"Не изменилась: {norm_url}")
                time.sleep(delay_sec)
                continue

            if new_norm_url != norm_url:
                if is_document_already_saved(db_connection, new_norm_url):
                    print(f"Редирект при обновлении: {norm_url} на {new_norm_url} (уже существует)")
                    with db_connection.cursor() as cur:
                        cur.execute("UPDATE documents SET last_fetch_attempt = %s WHERE normalized_url
= %s", (current_time, norm_url))
                    db_connection.commit()
                    time.sleep(delay_sec)
                    continue
```

```
            new_fetch_time = int(time.time())
            new_last_mod = resp.headers.get('Last-Modified')
            new_etag = resp.headers.get('ETag')

            with db_connection.cursor() as cur:
                cur.execute("""
                    UPDATE documents SET
                        url = %s,
                        normalized_url = %s,
                        html = %s,
                        clean_text = %s,
                        title = %s,
                        author = %s,
                        publish_date = %s,
                        source = %s,
                        fetch_timestamp = %s,
                        last_modified_header = %s,
                        etag_header = %s,
                        last_fetch_attempt = %s
                    WHERE normalized_url = %s
                """, (orig_url, new_norm_url, resp.text, new_clean_text, new_title, new_author,
new_publish_date,
                      new_source_name,
                      new_fetch_time,
                      new_last_mod, new_etag, new_fetch_time, norm_url))
            db_connection.commit()

            print(f"Обновлено: {norm_url}")
            updated_in_batch += 1
            time.sleep(delay_sec)

        total_updated += updated_in_batch
        print(f"Пакет {batch_idx} завершён: обновлено {updated_in_batch} из
{len(documents_for_refresh)}")

    if total_updated > 0:
        print(f"\nВсего документов обновлено: {total_updated}")
    else:
        print("\nНет устаревших документов для обновления.")


if __name__ == "__main__":
    if len(sys.argv) != 2:
        sys.exit(1)
    main(sys.argv[1])
```