Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа по курсу «Информационный поиск»

Студент: И. Д. Недосеков
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №1 «Добыча корпуса документов»

Необходимо подготовить корпус документов, который будет использован при выполнении остальных лабораторных работ:

- Скачать его к себе на компьютер. В отчёте нужно указать источник данных.
- Ознакомиться с ним, изучить его характеристики. Из чего состоит текст? Есть ли дополнительная мета-информация? Если разметка текста, какая она?
- Разбить на документы.
- Выделить текст.
- Найти существующие поисковики, которые уже можно использовать для поиска по выбранному набору документов (встроенный поиск Википедии, поиск Google с использованием ограничений на URL или на сайт). Если такого поиска найти невозможно, то использовать корпус для выполнения лабораторных работ нельзя!
- Привести несколько примеров запросов к существующим поисковикам, указать недостатки в полученной поисковой выдаче.

В результатах работы должна быть указаны статистическая информация о корпусе:

- Размер «сырых» данных.
- Количество документов.
- Размер текста, выделенного из «сырых» данных.
- Средний размер документа, средний объём текста в документе.

Лабораторная работа №2 «Булев индекс»

Требуется построить поисковый индекс, пригодный для булева поиска, по подготовленному в ЛР1 корпусу документов.
Требования к индексу:

- Самостоятельно разработанный, бинарный формат представления данных. Формат необходимо описать в отчёте, в побайтовом (или побитовом) представлении.

- Формат должен предполагать расширение, т.к. в следующих работах он будет меняться под требования новых лабораторных работ.
- Кроме обратного индекса, должен быть создан «прямой» индекс, содержащий в себе как минимум заголовки документов и ссылки на них (понадобятся для выполнения ЛР4, при генерации страницы поисковой выдачи).
- Для термов должна быть как минимум понижена капитализация.

В отчёте должно быть отмечено как минимум:

- Выбранное внутренне представление документов после токенизации.
- Выбранный метод сортировки, его достоинства и недостатки для задачи индексации.

Лабораторная работа №3 «Булев поиск»

Нужно реализовать ввод поисковых запросов и их выполнение над индексом, получение поисковой выдачи. Для демонстрации работы поисковой системы должен быть реализован веб-сервис, реализующий базовую функциональность поиска из двух страниц:

- Начальная страница с формой ввода поискового запроса.
- Страница поисковой выдачи, содержащая в себе форму ввода поискового запроса, 50 результатов поиска в виде текстов заголовков документов и ссылок на эти документы, а так же ссылку на получение следующих 50 результатов. дополнительная мета-информация? Если разметка текста, какая она?

Так же должна быть реализована утилита командной строки, загружающая индекс и выполняющая поиск по нему для каждого запроса на отдельной строчке входного файла.
В отчёте должно быть отмечено:

- Скорость выполнения поисковых запросов.
- Примеры сложных поисковых запросов, вызывающих длительную работу.
- Каким образом тестировалась корректность поисковой выдачи.

Лабораторная работа №4 «Ранжирование TF-IDF»

Необходимо сделать ранжированный поиск на основании схемы ранжирования TF-IDF. Теперь, если запрос содержит в себе только термины через пробелы, то его надо трактовать как нечёткий запрос, т.е. допускать неполное соответствие документа терминам запроса и т.п.

В отчёте нужно привести несколько примеров выполнения запросов, как удачных, так и не удачных.

1 Описание

Лабораторная работа №1 «Добыча корпуса документов»

Для подготовки корпуса было принято решение написать собсвенного робота для обхода сайта neolurk.org. Начиная с главной страницы, скачиваем страницу в формате html. На каждой страниый находим все ссылки, фильтруем (есть пути запрещенные сайтом и ссылки ведущие на другие сайты). После чего вытаскиваем только видимый текст статьи и передаем его в mystem. Эта библиотека производит лексический анализ и выдает значимые слова в приведенной форме. Сохраняем полученный текст и исходный документ в mongodb.

Количество документов: 20000

Размер текста: 408 MB

Средний размер документа: 21 KB

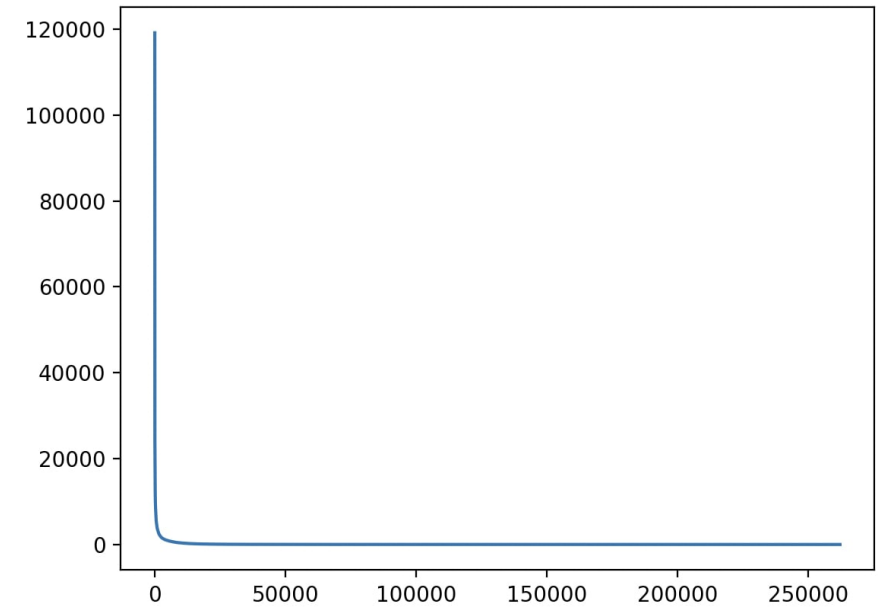

Рис. 1: Закон Ципфа для корпуса

Лабораторная работа №2 «Булев индекс»

Для реализации программы построения индекса, был выбран Rust. Индекс разделяется на прямой (doc_id: слова) и обратный (слово: doc_ids). Прямой индекс в свою очередь разделен на файл мета информации об индексе, и сам индекс. Запись в нем представляется в следующем виде:

< Doc_id 12 байт >< Количество слов в документе unt64 > [
< размер слова в байтах uint64 >< слово в байтах >,
...
]

В файле мета информации хранится ID документа, сдвиг от начала и размер записи в байтах.

Обратный индекс хранится аналогично прямому.

Для построения будем использовать SPIMInvert алгоритм.

Лабораторная работа №3 «Булев поиск»

Написан web интерфейс на FastApi.

На этой стороне проверяется корректность запроса и заменяются опечатки через биграммы.

Среднее время запроса: 6.2 сек.

Лабораторная работа №4 «Ранжирование TF-IDF»

Во время построения прямого индекса считается tf, idf считается после нахождения подходящих документов. Строится вектор запроса и найденных документов. Сортируем по величине косинуса и выдаем результат.

2 Исходный код

engine/index_api/forward_index.rs

```rust
use std::{
  collections::HashMap,
  io::{Read, Seek, Write},
  mem::size_of,
  ops::AddAssign,
};

lazy_static! {
  static ref FORWARD_INDEX_RECORD_METADATA_SIZE: usize =
  bincode::serialized_size(&ForwardIndexRecordMetadata::default()).unwrap() as
↪  usize;
}

use futures::TryStreamExt;
use mongodb::bson::Document;

use std::fs::OpenOptions;

use serde::{de::IntoDeserializer, Deserialize, Serialize};

#[derive(Serialize, Deserialize, PartialEq, Debug)]
pub struct ForwardIdexRecord {
  pub id: [u8; 12],
  pub lemmas: Vec<Lemma>,
}

#[derive(Serialize, Deserialize, PartialEq, Debug)]
pub struct Lemma {
  pub frequency: f64,
  pub text: String,
}

impl ForwardIdexRecord {
  fn from_document(document: Document) -> Self {
    let lemmas: Vec<String> = document
    .get_array("lemmas")
    .unwrap()
    .into_iter()
    .map(|lemma_info| {
      lemma_info
      .as_document()
```

```
        .unwrap()
        .get_str("text")
        .unwrap()
        .to_string()
      })
      .collect();

    let mut lemmas_count = HashMap::new();
    let count_words = lemmas.len() as f64;
    for lemma in lemmas {
      lemmas_count.entry(lemma).or_insert(0).add_assign(1);
    }

    return Self {
      id: document.get_object_id("_id").unwrap().bytes(),
      lemmas: lemmas_count
      .into_iter()
      .map(|(lemma, count)| Lemma {
        frequency: count as f64 / count_words,
        text: lemma,
      })
      .collect(),
    };
  }
}

#[derive(Serialize, Deserialize, Debug, Default)]
pub struct ForwardIndexRecordMetadata {
  id: [u8; 12],
  offset_in_index_file: u64,
  size_of_record: u64,
}

pub async fn build_forward_index(cursor: &mut mongodb::Cursor<Document>) {
  let mut index_file = OpenOptions::new()
  .truncate(true)
  .create(true)
  .write(true)
  .open("./forward_index.bin")
  .unwrap();

  let mut index_metadata_file = OpenOptions::new()
  .truncate(true)
  .create(true)
```

```
    .write(true)
    .open("./forward_index_metadata.bin")
    .unwrap();

  while let Some(document) = cursor.try_next().await.unwrap() {
    let record = ForwardIdexRecord::from_document(document);
    let record_as_bytes = bincode::serialize(&record).unwrap();

    let current_metadata = ForwardIndexRecordMetadata {
      id: record.id,
      offset_in_index_file: index_file.metadata().unwrap().len(),
      size_of_record: record_as_bytes.len() as u64,
    };

    index_file.write(&record_as_bytes[..]).unwrap();

    let record_as_bytes = bincode::serialize(&current_metadata).unwrap();
    index_metadata_file.write(&record_as_bytes[..]).unwrap();
  }
}

use std::fs::File;
use std::io::SeekFrom;

type DocId = [u8; 12];
pub struct ForwardIndexApi {
  index_file: File,
  index_metadata: File,

  loaded_metadata: Option<HashMap<DocId, ForwardIndexMetaRecord>>,
}

struct ForwardIndexMetaRecord {
  offset_in_index_file: u64,
  size_of_record: u64,
}

impl ForwardIndexApi {
  pub fn new(index_path: &str, index_metadata_path: &str) -> Self {
    Self {
      index_file: File::open(&index_path).unwrap(),
      index_metadata: File::open(&index_metadata_path).unwrap(),

      loaded_metadata: None,
```

```
    }
  }

  pub fn next(&mut self) -> Result<ForwardIdexRecord, Box<dyn std::error::Error>> {
    let mut metadata_buf = vec![0; *FORWARD_INDEX_RECORD_METADATA_SIZE];
    self.index_metadata.read_exact(&mut metadata_buf)?;

    let next_record_info: ForwardIndexRecordMetadata =
    bincode::deserialize(&metadata_buf).unwrap();

    let mut record_buf = vec![0u8; next_record_info.size_of_record as usize];
    self.index_file.read_exact(&mut record_buf.as_mut_slice())?;

    let next_record: ForwardIdexRecord =
↪   bincode::deserialize(&record_buf.as_slice()).unwrap();

    return Ok(next_record);
  }

  fn _load_metadata_record(
  &mut self,
  ) -> Result<ForwardIndexRecordMetadata, Box<dyn std::error::Error>> {
    let mut metadata_buf = vec![0; *FORWARD_INDEX_RECORD_METADATA_SIZE];
    self.index_metadata.read_exact(&mut metadata_buf)?;
    let next_record_info: ForwardIndexRecordMetadata =
↪   bincode::deserialize(&metadata_buf)?;

    return Ok(next_record_info);
  }

  pub fn load_metadata(&mut self) {
    if let Some(_) = self.loaded_metadata {
      return;
    }

    let mut metadata = HashMap::new();

    while let Ok(next_record) = self._load_metadata_record() {
      metadata.insert(
      next_record.id,
      ForwardIndexMetaRecord {
        offset_in_index_file: next_record.offset_in_index_file,
        size_of_record: next_record.size_of_record,
      },
```

```
      );
    }
    self.loaded_metadata = Some(metadata);
  }

  pub fn load_record_by(&mut self, id: &DocId) -> ForwardIdexRecord {
    self.load_metadata();
    let loaded_meta = self.loaded_metadata.as_ref().unwrap();
    let meta_info = loaded_meta.get(id).unwrap();
    self.index_file
    .seek(SeekFrom::Start(meta_info.offset_in_index_file))
    .unwrap();
    let mut buf = vec![0u8; meta_info.size_of_record as usize];
    self.index_file.read_exact(&mut buf).unwrap();
    let record: ForwardIdexRecord = bincode::deserialize(&buf).unwrap();

    return record;
  }
}

```

engine/index_api/reversed_index.rs

```
#[macro_use]
extern crate lazy_static;

pub mod forward_index;
pub mod reversed_index;

```

engine/index_api/lib.rs

```
use std::{
  collections::{BTreeMap, HashMap},
  io::{Read, Write},
};

use crate::forward_index;

pub async fn spimi_invert(
stream: &mut forward_index::ForwardIndexApi,
) -> Result<(), Box<dyn std::error::Error>> {
  let mut dictionary = BTreeMap::new();
  let mut count_documents = 0;
  while let Ok(docs) = stream.next() {
```

```
      println!("{count_documents}");
      count_documents += 1;
      for lemma in docs.lemmas {
        dictionary
        .entry(lemma.text)
        .or_insert(Vec::default())
        .push(docs.id);
      }

  }

  ReversedIndexAPI::dump_to_file(dictionary, count_documents)?;

  Ok(())
}

#[derive(serde::Serialize, serde::Deserialize)]
struct ReversedIndexRecord {
word: String,
doc_ids: Vec<[u8; 12]>,
idf: f64,
}

#[derive(serde::Serialize, serde::Deserialize)]
struct ReVersedIndex {
index: Vec<ReversedIndexRecord>,
}

pub struct ReversedIndexAPI {
file: std::fs::File,
}

impl ReversedIndexAPI {
fn new(path: &str) -> Self {
  Self {
    file: std::fs::File::open(&path).unwrap(),
  }
}

fn dump_to_file(
dictionary: BTreeMap<String, Vec<[u8; 12]>>,
count_documents: u64,
) -> Result<(), Box<dyn std::error::Error>> {
  let mut file = std::fs::OpenOptions::new()
```

```
58    .truncate(true)
59    .create_new(true)
60    .write(true)
61    .open("./reversed_index.bin")?;
62
63    let index = dictionary
64    .into_iter()
65    .map(|(word, doc_ids)| ReversedIndexRecord {
66      word: word,
67      idf: f64::ln(count_documents as f64 / doc_ids.len() as f64),
68      doc_ids: doc_ids,
69    })
70    .collect();
71
72    let buf = bincode::serialize(&ReVersedIndex { index: index })?;
73
74    file.write(&buf.as_slice())?;
75
76    Ok(())
77  }
78
79  pub fn load_from_file(path: &str) -> (BTreeMap<String, Vec<[u8; 12]>>,
    ↪  HashMap<String, f64>) {
80    let mut file = std::fs::File::open(path).unwrap();
81    let buf_size = file.metadata().unwrap().len();
82
83    let mut buf = vec![0u8; buf_size as usize];
84
85    file.read_exact(&mut buf.as_mut_slice()).unwrap();
86    let index: ReVersedIndex = bincode::deserialize(&buf).unwrap();
87
88    let words_idf = HashMap::from_iter(
89    index
90    .index
91    .iter()
92    .map(|record| (record.word.clone(), record.idf)),
93    );
94
95    return (BTreeMap::from_iter(
96    index
97    .index
98    .into_iter()
99    .map(|record| (record.word, record.doc_ids)),
100   ), words_idf);
```

```
}
}
```

engine/indexer/main.rs

```rust
use serde::{Deserialize, Serialize};

use mongodb::bson::doc;
use mongodb::bson::oid::ObjectId;

#[derive(Debug, Serialize, Deserialize)]
struct Page {
  _id: ObjectId,
  path: String,
  lemmas: Vec<Lemma>,
}

use index_api::forward_index::*;
use index_api::reversed_index::*;

#[derive(Debug, Serialize, Deserialize)]
struct Lemma {
  text: String,
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
  let client = mongodb::Client::with_uri_str("mongodb://localhost").await?;

  let db = client.database("test");

  let test_collection = db.collection::<Page>("docs");

  let aggregate_pipeline = vec![doc! {
    "$sort": {
      "_id": 1
    }
  }];
  let mut cursor = test_collection.aggregate(aggregate_pipeline, None).await?;

  build_forward_index(&mut cursor).await;

  let mut fapi = ForwardIndexApi::new("./forward_index.bin",
↪  "./forward_index_metadata.bin");
```

```

  spimi_invert(&mut fapi).await?;

  Ok(())
}

```

engine/server/boolean_search.rs

```rust
use std::collections::BTreeMap;

// use futures::stream::Iter;

use std::cmp::Ordering;

#[derive(Debug)]
struct SearchState<'a> {
  ids: &'a Vec<[u8; 12]>,
  position: usize,
}

pub fn boolean_search(
request: &Vec<String>,
reversed_index: &BTreeMap<String, Vec<[u8; 12]>>,
) -> Vec<[u8; 12]> {
  let mut docs_containing_words = Vec::with_capacity(request.len());
  let mut maximum_doc_id = [u8::MIN; 12];

  for word in request {
    match reversed_index.get(word) {
      Some(docs) => {
        docs_containing_words.push(SearchState {
          ids: docs,
          position: 0,
        });
        maximum_doc_id = std::cmp::max(maximum_doc_id, docs[0])
      }
      None => continue,
    }
  }
  let mut result = Vec::new();

  'main_loop: loop {
    let mut some_cheked = false;
```

```
    let mut all_previous_equal = true;

    'word_cycle: for state in &mut docs_containing_words {
      'doc_id_cycle: while let Some(next_doc_id) = state.ids.get(state.position) {
        some_cheked = some_cheked || true;

        match next_doc_id.cmp(&maximum_doc_id) {
          Ordering::Less => {
            state.position += 1;
            continue 'doc_id_cycle;
          }
          Ordering::Equal => {
            all_previous_equal = all_previous_equal && true;
            break 'doc_id_cycle;
          }
          Ordering::Greater => {
            all_previous_equal = false;
            maximum_doc_id = next_doc_id.clone();
            continue 'main_loop;
          }
        }
      }
    }
    if all_previous_equal {
      result.push(maximum_doc_id);
      docs_containing_words
      .iter_mut()
      .for_each(|state| state.position += 1)
    }

    if !some_cheked {
      break 'main_loop;
    }
  }

  return result;
}

```

engine/server/main.rs

```
#[macro_use]
extern crate lazy_static;

```

```
4   lazy_static! {
5     static ref REVERSED_INDEX: BTreeMap<String, Vec<[u8; 12]>> =
6     ReversedIndexAPI::load_from_file("../indexer/reversed_index.bin").0;
7     static ref WORDS_IDF: HashMap<String, f64> =
8     ReversedIndexAPI::load_from_file("../indexer/reversed_index.bin").1;
9   }
10
11  use std::collections::{BTreeMap, HashMap};
12
13  use index_api::{forward_index, reversed_index::ReversedIndexAPI};
14  use serde::{Deserialize, Serialize};
15
16  pub mod boolean_search;
17
18  #[derive(Serialize, Deserialize, Debug)]
19  struct RequestSchema {
20    words: Vec<String>,
21  }
22
23  #[derive(Serialize, Deserialize, Debug)]
24  struct ResponseSchema {
25    doc_ids: Vec<mongodb::bson::oid::ObjectId>,
26  }
27
28  use actix_web::{get, post, App, HttpResponse, HttpServer, Responder};
29
30  #[get("/")]
31  async fn hello() -> impl Responder {
32    HttpResponse::Ok().body("Hello world!")
33  }
34
35  use itertools::Itertools;
36  use std::cmp::Ordering;
37
38
39  fn calculate_cos(v1: &Vec<f64>, v2: &Vec<f64>) -> f64 {
40    let numerator = v1.iter().zip(v2.iter()).map(|(a, b)| a * b).sum::<f64>();
41    let denumerator =
42    f64::sqrt(v1.iter().map(|a| a * a).sum()) * f64::sqrt(v2.iter().map(|b| b *
↪   b).sum());
43
44    return numerator / denumerator;
45  }
46
```

```rust
fn rank_docs(request: &Vec<String>, docs: &Vec<[u8; 12]>) -> Vec<[u8; 12]> {
  let mut forward_api = forward_index::ForwardIndexApi::new(
  "../indexer/forward_index.bin",
  "../indexer/forward_index_metadata.bin",
  );

  let request_words_count = request.len() as f64;
  let request_vector = request
  .iter()
  .map(|word| WORDS_IDF.get(word).unwrap_or(&0f64) * 1f64 / request_words_count)
  .collect_vec();

  let result = docs.iter().cloned().sorted_by_key( |doc| {
    let forward_record = forward_api.load_record_by(doc);
    let searching_words: Vec<forward_index::Lemma> = forward_record
    .lemmas
    .into_iter()
    .filter(|lemma| request.contains(&lemma.text))
    .sorted_by_key(|lemma| lemma.text.clone())
    .collect_vec();

    let document_vector = searching_words
    .iter()
    .map(|lemma| lemma.frequency * WORDS_IDF.get(&lemma.text).unwrap())
    .collect();

    return (calculate_cos(&request_vector, &document_vector) * 1000f64) as i64;
  }
  ).rev().collect();

  return result;
}

#[post("/search")]
async fn search(req_body: String) -> impl Responder {
  println!("{req_body}");
  let data: RequestSchema = serde_json::from_str(&req_body).unwrap();
  let result_ids = boolean_search::boolean_search(&data.words, &REVERSED_INDEX);
  let ranked_results = rank_docs(&data.words, &result_ids);
  let result = ResponseSchema {
    doc_ids: ranked_results.into_iter().map(|doc_id|
↪ mongodb::bson::oid::ObjectId::from_bytes(doc_id)).collect(),
  };

```

```rust
  return HttpResponse::Ok().body(serde_json::to_string(&result).unwrap());
}

#[tokio::main]
async fn main() -> std::io::Result<()> {
  HttpServer::new(|| App::new().service(hello).service(search))
  .bind(("localhost", 8080))?
  .run()
  .await
}

```

scrapper/scrapy.py

```python
import asyncio
from collections import deque
from itertools import chain, islice
from typing import Iterable, TypeVar

from aiolimiter import AsyncLimiter
from bs4 import BeautifulSoup
from httpx import AsyncClient, Response, codes
from loguru import logger

T = TypeVar('T')


class ChankedQueue(deque[tuple[T, ...]]):
def __init__(self, chunk_size: int = 50, *arg, **kwargs) -> None:
self.chunk_size = chunk_size
super(ChankedQueue, self).__init__(*arg, **kwargs)

def extend(self, iterable: Iterable[T]) -> None:  # type: ignore
real_dequeue = super(ChankedQueue, self)
start = 0
if real_dequeue.__bool__():
last_chunk = real_dequeue.pop()
stop = self.chunk_size - len(last_chunk)
else:
last_chunk = tuple()
stop = self.chunk_size

while True:
next_chunk = tuple(chain(last_chunk, islice(iterable, start, stop)))
```

```
    if not next_chunk:
    break

    real_dequeue.append(next_chunk)
    last_chunk = tuple()
    start, stop = stop, stop + self.chunk_size


    def get_all_url_from_html(source: str) -> set[str]:
    soup = BeautifulSoup(source, features="html.parser")
    return {a['href'] for a in soup.find_all('a', href=True)}


    async def make_request(url: str, session: AsyncClient, throttler: AsyncLimiter) ->
    ↪  Response:
    async with throttler:
    logger.debug('scrapy {}', url)
    return await session.get(url)


    async def scrape(url: str, session: AsyncClient, throttler: AsyncLimiter) ->
    ↪  Response | None:
    response = await make_request(url, session, throttler)

    if response.status_code == codes.OK:
    return response

    if response.status_code == codes.TOO_MANY_REQUESTS or response.status_code ==
    ↪  codes.SERVICE_UNAVAILABLE:
    logger.warning('retry later by {} on url {}', response.status_code, url)
    await asyncio.sleep(1.5)
    response = await make_request(url, session, throttler)

    if codes.is_redirect(response.status_code):
    logger.warning('redirect on {} {}', url, response.status_code)
    response = await make_request(response.headers['Location'], session, throttler)

    if response.status_code != codes.OK:
    logger.error('cannot scrape {} code {}', url, response.status_code)
    return None



    return response
```

```

```

scrapper/scrapy.py

```python
import asyncio
from itertools import chain

from httpx import AsyncClient, Timeout
from loguru import logger

from documents import upload_documents
from scrapy import ChankedQueue, get_all_url_from_html, scrape
from config import Config
from postprocess import build_bigram_words_dictionary

def get_new_urls(paths: set[str], visited: set[str]) -> set[str]:
return {
  f'{Config.url}{path}'
  for path in paths
  if path.startswith('/wiki/') and path not in visited
}


async def empty_coroutine() -> None:
return


async def run():
visited_paths = {f'{Config.url}{Config.start_path}'}
urls_to_process = ChankedQueue()
urls_to_process.append((f'{Config.url}{Config.start_path}',))

upload_documents_task = empty_coroutine()

session = AsyncClient(timeout=Timeout(100, connect=60))

while len(visited_paths) < Config.count_documents:

url_chunk = urls_to_process.popleft()
scrape_tasks = [scrape(url, session=session, throttler=Config.throttler) for url in
↪  url_chunk]
_, *results = await asyncio.gather(upload_documents_task, *scrape_tasks)

visited_paths.update(url_chunk)
```

```
    paths_in_results = set(
    chain.from_iterable(
    get_all_url_from_html(result.text)
    for result in results
    if result
    )
    )
    urls_to_process.extend(get_new_urls(paths_in_results, visited_paths))

    upload_documents_task = upload_documents(results)

    await upload_documents_task
    await session.aclose()
    logger.debug('finished downloading, start build dictionary and bigrams')

    await build_bigram_words_dictionary()


    logger.debug("finished scraping")




    if __name__ == "__main__":
    asyncio.run(run())

```

web-api/main.py

```
    import json

    import httpx
    from fastapi import FastAPI, Request, Form
    from fastapi.templating import Jinja2Templates
    from bson import ObjectId

    from helpers.mongodb_connector import mongo_client

    from loguru import logger

    from enrich import get_enrich_query

    app = FastAPI()
    templates = Jinja2Templates(directory="templates/")
```

```
16
17  EXAMPLE_DOCS = ['http://neolurk.org/wiki/Заглавная_страница',
    ↪  'http://neolurk.org/wiki/Заглавная_страница']
18
19
20  @app.get('/')
21  def read_form():
22  return 'hello world'
23
24
25  @app.get("/form")
26  def form_post(request: Request):
27  result = ""
28  return templates.TemplateResponse('form.html', context={'request': request,
    ↪  'result': result})
29
30
31  async def get_search_results(query: list[str]) -> list[str]:
32  ids = await get_search_results_ids(query)
33  ids = [ObjectId(id) for id in ids]
34
35  results = mongo_client.test.docs.aggregate(
36  [
37  {'$match': {'_id': {'$in': ids}}},
38  {'$project': {'path': 1}}
39  ]
40  )
41  res_url = []
42  async for result in results:
43  res_url.append(f'http://neolurk.org{result["path"]}')
44
45  return res_url
46
47
48  async def get_search_results_ids(query: list[str]) -> list[str]:
49  async with httpx.AsyncClient() as client:
50  response = await client.post(
51  'http://localhost:8080/search',
52  headers={'Content-Type': 'application/x-www-form-urlencoded'},
53  content=f'{{"words": { [word for word in sorted(query)] }}}'.replace("'", '"')
54  )
55
56  if response.status_code != httpx.codes.OK:
57  logger.error('something went wrong', response)
```

```
  raise Exception('asd')

  return [doc['$oid'] for doc in response.json()['doc_ids']]


  @app.post("/form")
  async def form_post(request: Request, statement: str = Form(default='')):
  result = EXAMPLE_DOCS if statement else []
  enrich_query = await get_enrich_query(statement)
  query = [
  enrich_query.fuzzy_mapping[lemma][0]
  if lemma in enrich_query.fuzzy_mapping
  else lemma
  for lemma in enrich_query.query
  ]
  search_result = await get_search_results(query)
  logger.debug('Enrich query {}  mapping {}', enrich_query.query,
  ↪  enrich_query.fuzzy_mapping)
  return templates.TemplateResponse('form.html', context={'request': request,
  ↪  'results': search_result})

```

web-api/fuzzy.py

```
from helpers.mongodb_connector import mongo_client
from helpers.bigrams import get_bigrams_of_word
from config import Config



class Dimension:

def __init__(self, word: str) -> None:
self.word = word
self.words_bigrams = set(get_bigrams_of_word(word))

def distance_to(self, candidate: str) -> float:
if abs(len(self.word) - len(candidate)) > 3:
return 0
words_bigrams = self.words_bigrams
candidates_bigrams = set(get_bigrams_of_word(candidate))
distance = abs(len(words_bigrams) - len(words_bigrams & candidates_bigrams))
return abs(1 - distance / len(words_bigrams | candidates_bigrams))

```

```

async def get_fuzzy_words(word: str) -> list[str]:
bigrams = get_bigrams_of_word(word)
word_dimension = Dimension(word)
fuzzy_words_rating: set[tuple[str, float]] = set()
async for bigram_with_words in mongo_client.test.bigram_words.find({'bigram': {'$in':
↪  bigrams}}):
fuzzy_words_rating.update(
(candidate, word_dimension.distance_to(candidate))
for candidate in bigram_with_words['words']
if word_dimension.distance_to(candidate) >= Config.minimum_distance_for_fuzzy_words
)

return [
word
for word, _ in reversed(sorted(fuzzy_words_rating, key=lambda row: row[1]))
]

async def is_dictionary_word(word: str) -> bool:
return bool(
await mongo_client.test.word_bigrams.count_documents({'word': word})
)
```

web-api/enrich.py

```
import asyncio

from helpers.mystem import system
from helpers.stemmer import stemmer

from typing import NamedTuple

from fuzzy import is_dictionary_word, get_fuzzy_words


def get_lexemes(text: str) -> list[str]:
return [
stemmer.stem(lexem)
for lexem in system.lemmatize(text)
if lexem.strip()
]


class EnrichQuery(NamedTuple):
```

```
    query: list[str]
    fuzzy_mapping: dict[str, str]


    async def process_lexeme(lexeme: str) -> tuple[str, str]:
    if await is_dictionary_word(lexeme):
    return lexeme, lexeme
    else:
    return lexeme, await get_fuzzy_words(lexeme) or lexeme


    async def get_enrich_query(text: str) -> EnrichQuery:
    lexemes = get_lexemes(text)
    query: list[str] = []
    fuzzy_mapping: dict[str, str] = {}
    process_lexeme_tasks = (process_lexeme(lexeme) for lexeme in lexemes)

    for lexeme, processed_lexeme in await asyncio.gather(*process_lexeme_tasks):
    query.append(lexeme)
    if lexeme != processed_lexeme:
    fuzzy_mapping[lexeme] = processed_lexeme

    return EnrichQuery(query, fuzzy_mapping)


```

3 Выводы

Выполнив лабораторную работу по курсу «Информационный поиск», я написал веб паука, научился строить индекс для поиска. Изучил алгоритмы ранжирования, лемматизации и основы работы поисковых движков.

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))