Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа по курсу «Информационный поиск»

Студент: А. К. Киреев
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 скрапера. В качестве сайта была выбрана английская википедия: https://en.wikipedia.org. Для настройки робота используем аргументы командной строки и данные из robots.txt. Начиная с главной страницы, скачиваем страницу в формате html. На каждой странице находим все ссылки, фильтруем эти ссылки (есть пути запрещенные сайтом и ссылки ведущие на другие сайты или посещенные ссылки). Одновременно с этим получаем видимый текст статьи и при помощи фреймворка nltk нормализуем его. Эта библиотека производит лексический анализ и выдает значимые слова в приведенной форме. Сохраняем полученный текст, ссылку и заголовок текста в mongodb.

Количество документов: 20000

Размер текста: 423 MB

Средний размер документа: 36 KB

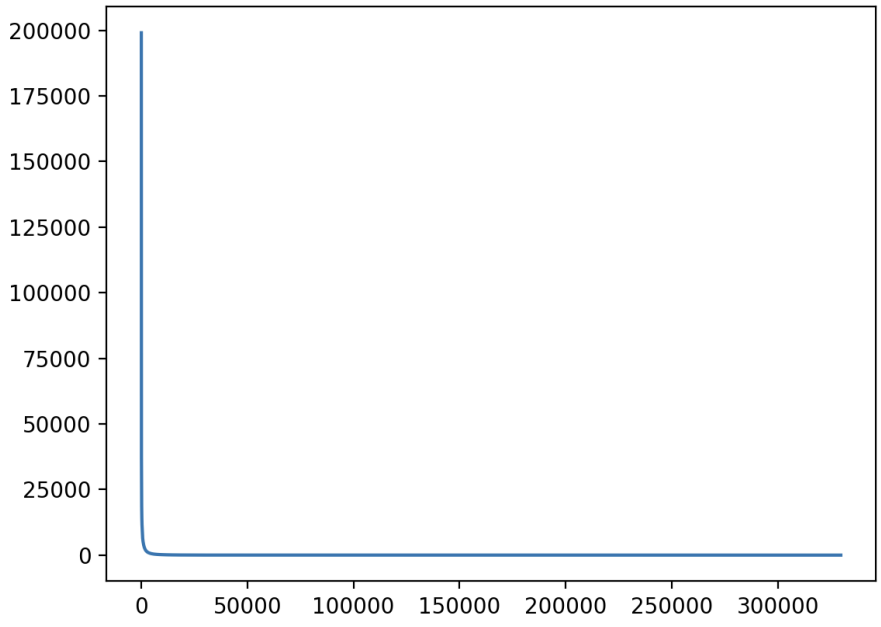

Рис. 1: Закон Ципфа для корпуса

2 Исходный код

scraper.py

```
import asyncio
import coloredlogs
import logging

from aiolimiter import AsyncLimiter
from httpx import AsyncClient, Response, codes
from bs4 import BeautifulSoup
from robots import RobotsParser
from typing import Tuple, Set, List
from functools import partial, wraps

from utils import Settings
from bucket_queue import BucketQueue
from db import save_documents, save_words, save_bigrams,\
  load_visited_urls, dump_visited_urls, load_pending_urls, dump_pending_urls
from text_enrich import enrich_text, get_text


class Scraper:
  logger = logging.getLogger(__name__)

  def __init__(self, settings: Settings):
    self.settings: Settings = settings
    Scraper.logger.setLevel(level=self.settings.log_level)
    coloredlogs.install(level=Scraper.logger.level)

    self.robots = RobotsParser.from_uri(uri=f"{self.settings.url_base}/robots.txt")
    if self.settings.rps > 50:
      Scraper.logger.warning("Your rps(%s) too large", self.settings.rps)
    self.throttler = AsyncLimiter(max_rate=self.settings.rps, time_period=1)

  async def _load_scraper_state(self):
    loaded_visited_urls: Set[str] = await load_visited_urls(Scraper.logger)
    if loaded_visited_urls:
      self.visited = loaded_visited_urls
    else:
      self.visited = {self.settings.start_url}
      await dump_visited_urls({self.settings.start_url}, Scraper.logger)

    self.queue = BucketQueue(settings=self.settings)
    loaded_pending_urls: Set[str] = await load_pending_urls(Scraper.logger)
```

```
42        if loaded_pending_urls:
43          self.queue.extend(loaded_pending_urls)
44        else:
45          self.queue.extend(self.visited)
46
47      def _filter_urls(self, urls: Set[str]) -> Set[str]:
48        filtered: Set[str] = {f'{self.settings.url_base}{url}' for url in urls
49                  if url.startswith('/wiki/')
50                  and self.robots.can_fetch("*", url)
51                  and ":" not in url
52                  and f'{self.settings.url_base}{url}' not in self.visited}
53        Scraper.logger.info("Discarded %s refs from current batch of documents",
          ↪  len(urls) - len(filtered))
54        return filtered
55
56      def _find_hrefs(self, sources: Tuple) -> Set[str]:
57        result: Set[str] = set()
58        for s in sources:
59          if not s:
60            continue
61          soup = BeautifulSoup(s.text, features="html.parser")
62          result |= {a['href'] for a in soup.find_all('a', href=True)}
63        Scraper.logger.info("Got %s refs from current batch of documents", len(result))
64        return result
65
66      @staticmethod
67      def _handle_response(task):
68        @wraps(task)
69        async def wrapper(self, url: str, client: AsyncClient):
70          retries: int = 0
71          max_retires: int = 5
72          response: Response = await task(self, url, client)
73
74          while ((response.status_code == codes.TOO_MANY_REQUESTS or
            ↪  response.status_code == codes.SERVICE_UNAVAILABLE)
75            and retries <= max_retires):
76            Scraper.logger.warning("Too many request for url: %s, retry: %s", url,
              ↪  retries)
77            await asyncio.sleep(1)
78            response = await task(self, url, url, client)
79            retries += 1
80
81          if retries > max_retires:
82            Scraper.logger.error("Can't scrape url: %s, max retries was reached", url)
```

```
          return None

        if codes.is_redirect(response.status_code):
          Scraper.logger.warning("Redirect on url: %s", url)
          response = await task(self, response.headers['Location'], client)

        if response.status_code != codes.OK:
          Scraper.logger.error("Response error on url: %s, code %s", url,
          ↪  response.status_code)
          return None

        return response

      return wrapper

    @_handle_response
    async def _scrape(self, url: str, client: AsyncClient) -> Response:
      async with self.throttler:
        try:
          return await client.get(url)
        except Exception as e:
          return None

    async def run(self):
      await self._load_scraper_state()
      async with AsyncClient() as client:
        task = partial(self._scrape, client=client)
        self.queue.set_task(task)
        for tasks in self.queue:
          results: Tuple = await asyncio.gather(*tasks)

          if len(self.visited) > self.settings.max_scraped_count:
            break

          enriched: List[List[str]] = list(map(lambda r: enrich_text(get_text(r)),
          ↪  results))
          await save_documents(enriched, results, Scraper.logger)
          await save_words(enriched, Scraper.logger)

          processed: Set[str] = self._filter_urls({result.url.path for result in
          ↪  results})
          await dump_visited_urls(processed, Scraper.logger)
          self.visited |= processed

```

```
124             found_refs: Set[str] = self._filter_urls(self._find_hrefs(results))
125             await dump_pending_urls(found_refs, Scraper.logger)
126             self.queue.extend(found_refs)
127 
128             Scraper.logger.info("Scraped: %s", len(self.visited))
129 
130         await save_bigrams()
```

Лабораторная работа №2 «Булев индекс»

Для реализации программы построения прямого и обратного индексов, был выбран Rust. Прямой индекс в свою очередь разделен на файл-заголовок с мета-информацией об индексе, и на файл с текстом и значениями tf для слов текста.

Запись в файле с мета-информацией представляется в следующем виде:

< Doc_id 12 байт >< отступ в файле с текстом u64 >
...

Запись в файле с текстом представляется в следующем виде:

< Количество слов в документе u64 > [
< размер слова в байтах u64 >< слово >< значение tf f64 >,
...
]

Обратный индекс также, как и прямой, разбит на два файла.

Запись в файле с мета-информацией представляется в следующем виде:

< размер слова в байтах u64 >< слово >
< отступ в файле с документами u64 >< значение idf f64 >
...

Запись в файле с документами представляется в следующем виде:

< размер слова в байтах u64 >
< слово >
< Количество документов для данного слова u64 > [
< Doc_id 12 байт >,
...
]

Для построения обратного индекса во внешней памяти будем использовать SPIMI алгоритм.

3 Исходный код

inverted_index.rs

```rust
pub async fn build(&mut self) -> Result<(), Box<dyn std::error::Error>> {
      let dbase = db::connect_to_docs_database().await?;
      let mut cur = dbase.get_cursor().await?;

      let mut inverted_idx_block = BTreeMap::new();

      let mut tasks = Vec::new();

      while let Some(result) = cur.next().await {
          let doc: db::Doc = bson::from_document(result?)?;
          let mut used = HashSet::new();
          for word in doc.words {
              used.insert(word.to_owned());
              if !inverted_idx_block.contains_key(&word) {
                  inverted_idx_block.insert(word, vec![doc._id]);
              } else {
                  if let Some(posting_list) = inverted_idx_block.get_mut(&word) {
                      posting_list.push(doc._id);
                  }
              }

              self.cur_map_size += 1;

              if self.cur_map_size >= self.max_map_size {
                  print!("Block {:?} dumped\n", self.count_of_written_blocks);
```

```
                    tasks.push(InvertedIndex::write_block_to_disk(
                        inverted_idx_block,
                        self.count_of_written_blocks,
                        self.blocks_directory,
                    ));
                    self.count_of_written_blocks += 1;
                    inverted_idx_block = BTreeMap::new();
                    self.cur_map_size = 0;
                }
            }

            for word in used {
                self.head
                    .entry(word)
                    .and_modify(|pair| *pair = (pair.0, pair.1 + 1.0))
                    .or_insert((0, 1.0));
            }
        }

        if inverted_idx_block.len() > 0 {
            print!("EXTRA Block {:?} dumped\n", self.count_of_written_blocks);
            tasks.push(InvertedIndex::write_block_to_disk(
                inverted_idx_block,
                self.count_of_written_blocks,
                self.blocks_directory,
            ));
            self.count_of_written_blocks += 1;
        }

        let results = join_all(tasks).await;
        for res in results {
            res?;
        }
        print!("Joined\n");

        self.merge_blocks().await?;

        let mut resulting_block_file = self.blocks_directory.to_owned();
        resulting_block_file.push_str(&format!(
            "/block_{}.inv_idx.bin",
            self.count_of_written_blocks - 1
        ));
        copy(resulting_block_file, self.index_content_file).await?;
        print!("Inverted index's content file was built\n");
```

```
        let count_of_documents = dbase.get_count_of_documents().await?;
        for pair in self.head.values_mut() {
            *pair = (pair.0, f64::log10(count_of_documents as f64 / pair.1));
        }

        self.set_offsets().await?;
        self.write_head_to_disk().await?;
        print!("Inverted index was built\n");

        Ok(())
    }
```

Лабораторная работа №3 «Булев поиск»

Написан web-сервис на Flask, реализовано общение с поисковым движком при помощи httpx. Также в рамках данной работы реализовано исправление ошибок при помощи биграмм и алгоритма Дамерау-Левенштейна.

Среднее время запроса: 10 сек.

4 Исходный код

app.py

```
from flask import Flask, render_template, request, url_for, flash, redirect

from search_helper import BigramIndex
from request_enrich import get_enriched_words
from db import get_documents

from utils import Settings
from typing import List, Dict

import coloredlogs
import logging
import asyncio
import httpx
import time


```

```
17  loop = asyncio.get_event_loop()
18
19  logger = logging.getLogger(__name__)
20  logger.setLevel(level=logging.DEBUG)
21  coloredlogs.install(level=logging.DEBUG)
22
23  app = Flask(__name__)
24  app.config['SECRET_KEY'] = Settings().secret
25
26
27  @app.route('/', methods=('GET', 'POST'))
28  def search():
29    if request.method == 'POST':
30      search_request = request.form['request']
31      enriched_request: List[str] = get_enriched_words(search_request)
32
33      bi: BigramIndex = BigramIndex(enriched_request)
34      loop.run_until_complete(bi.build(logger))
35
36      search_dict: Dict = bi.get_search_dict()
37      logger.debug("Supposed request structure: %s", search_dict)
38
39      search_engine_request: List = [supposed[0] for supposed in
        ↪  search_dict.values()]
40      logger.debug("Request for search engine: %s", search_engine_request)
41
42      start = time.time()
43      search_engine_response = httpx.post('http://localhost:8080/search',
44                          json={'words': search_engine_request},
45                          timeout=None)
46      end = time.time()
47      logger.debug("Search engine request time [sec]: %s", end - start)
48
49      doc_ids: List = search_engine_response.json()["doc_ids"]
50      logger.debug("Got %s documents in search engine", len(doc_ids))
51      results = loop.run_until_complete(get_documents(doc_ids))
52      logger.debug("Response: %s", results)
53
54      front_request: List[Dict] = []
55      for given, changed in zip(enriched_request, search_engine_request):
56        if given != changed:
57          front_request.append({"word": changed, "color": "red"})
58        else:
59          front_request.append({"word": changed, "color": "black"})
```

```
60
61        if not search_request:
62          flash('Request is required!')
63        else:
64          return render_template('result.html', results=results,
            ↪  changed_request=front_request)
65      return render_template('search.html')
66
67
68    if __name__ == '__main__':
69      app.run(debug=False, use_reloader=False)
```

Лабораторная работа №4 «Ранжирование TF-IDF»

Во время построения прямого индекса считается tf, idf считается при построении обратного индекса. Для запроса также считаются значения tf, потом по полученным значениям строятся вектор запроса и векторы найденных документов. Полученные векторы нормируем и считаем косинусы. Для получения ранжированной выборки сортируем по значению косинуса (по убыванию).

5 Исходный код

engine.rs

```rust
impl Engine {
    pub async fn search(
        &self,
        query: Vec<String>,
    ) -> Result<Vec<ObjectId>, Box<dyn std::error::Error>> {
        let qvec = self.query_into_vec(&query);

        let mut result_of_bool_search: Option<Vec<Vec<u8>>> = None;

        for word in &query {
            let exists = self.inverted_index.head.get(word);
            match exists {
                Some(_) => {}
                None => {
                    continue;
                }
            }
            match result_of_bool_search {
                Some(intersec) => {
                    result_of_bool_search = Some(self.intersec(&intersec, word).await?);
                }
                None => {
                    result_of_bool_search = Some(self.get_posting_list(word).await?);
                }
            }
        }

        match result_of_bool_search {
            Some(documents) => {
                return Ok(self.get_best_documents(&qvec, documents).await?);
            }
            None => return Ok(vec![]),
        }
    }

    async fn get_best_documents(
        &self,
        qvec: &HashMap<String, f64>,
        documents: Vec<Vec<u8>>,
    ) -> Result<Vec<ObjectId>, Box<dyn std::error::Error>> {
```

```
        let mut result = Vec::default();
        let mut heap = BinaryHeap::new();

        for doc in documents {
            let dvec = self.doc_into_vec(&doc).await?;
            let mut cosine = 0.0;

            for word in qvec.keys() {
                if dvec.contains_key(word) {
                    cosine += qvec[word] * dvec[word];
                }
            }

            heap.push((NotNan::new(cosine).unwrap(), doc));
        }

        for _ in 0..self.top_n {
            if let Some((_, doc_id)) = heap.pop() {
                result.push(ObjectId::from_bytes(doc_id.try_into().unwrap()));
            } else {
                break;
            }
        }

        Ok(result)
    }

    async fn intersec(
        &self,
        curr: &Vec<Vec<u8>>,
        word: &String,
    ) -> Result<Vec<Vec<u8>>, Box<dyn std::error::Error>> {
        let mut result = Vec::default();

        let other = self.get_posting_list(word).await?;

        let (mut i, mut j) = (0, 0);

        while i < curr.len() && j < other.len() {
            if curr[i] == other[j] {
                result.push(curr[i].clone());
                i += 1;
                j += 1;
            } else if curr[i] < other[j] {
```

```
                i += 1;
            } else {
                j += 1;
            }
        }

        Ok(result)
    }

    async fn get_posting_list(
        &self,
        word: &String,
    ) -> Result<Vec<Vec<u8>>, Box<dyn std::error::Error>> {
        let mut result = Vec::default();

        let offset = self.inverted_index.head[word].0;

        let content_file = File::open(self.inverted_index.index_content_file).await?;
        let mut reader = BufReader::new(content_file);
        reader.seek(SeekFrom::Start(offset)).await?;

        let posting_list_len = reader.read_u64().await?;

        for _ in 0..posting_list_len {
            let mut bytes = vec![0u8; 12];
            reader.read_exact(&mut bytes).await?;
            result.push(bytes);
        }

        Ok(result)
    }

    fn query_into_vec(&self, query: &Vec<String>) -> HashMap<String, f64> {
        let mut tf = HashMap::new();
        for word in query {
            tf.entry(word.to_owned())
                .and_modify(|counter| *counter += 1.0)
                .or_insert(1.0);
        }
        for val in tf.values_mut() {
            *val = 1.0 + f64::log10(*val);
        }

        let keys = tf.keys().cloned().collect::<Vec<_>>();
```

```
129
130         for word in keys {
131             let default = (0, 0.0);
132             let idf = self.inverted_index.head.get(&word).unwrap_or(&default).1;
133             tf.entry(word).and_modify(|tf| *tf *= idf);
134         }
135
136         let len = f64::sqrt(tf.values().map(|&val| val * val).sum());
137         for val in tf.values_mut() {
138             *val /= len;
139         }
140
141         tf
142     }
143
144     async fn doc_into_vec(
145         &self,
146         doc_id: &Vec<u8>,
147     ) -> Result<HashMap<String, f64>, Box<dyn std::error::Error>> {
148         let mut tf = HashMap::new();
149
150         let offset = self.forward_index.head[doc_id];
151
152         let content_file = File::open(self.forward_index.index_content_file).await?;
153         let mut reader = BufReader::new(content_file);
154         reader.seek(SeekFrom::Start(offset)).await?;
155
156         let words_count = reader.read_u64().await?;
157
158         for _ in 0..words_count {
159             let len = reader.read_u64().await?;
160             let mut bytes = vec![0u8; len as usize];
161             reader.read_exact(&mut bytes).await?;
162             let tf_val = reader.read_f64().await?;
163             tf.insert(String::from_utf8(bytes).unwrap(), tf_val);
164         }
165
166         let keys = tf.keys().cloned().collect::<Vec<_>>();
167
168         for word in keys {
169             let default = (0, 0.0);
170             let idf = self.inverted_index.head.get(&word).unwrap_or(&default).1;
171             tf.entry(word).and_modify(|tf| *tf *= idf);
172         }
```

```
173
174         let len = f64::sqrt(tf.values().map(|&val| val * val).sum());
175         for val in tf.values_mut() {
176             *val /= len;
177         }
178
179         Ok(tf)
180     }
181 }
```

main.rs

```
1   #[macro_use]
2   extern crate lazy_static;
3   extern crate futures;
4
5   lazy_static! {
6     static ref ENGINE: Mutex<Option<engine::Engine>> = Mutex::new(None);
7   }
8
9   pub mod db;
10  pub mod engine;
11  pub mod index;
12  pub mod inverted_index;
13
14  use actix_web::{post, App, HttpResponse, HttpServer, Responder};
15  use serde::{Deserialize, Serialize};
16  use std::sync::Mutex;
17
18  #[derive(Debug, Serialize, Deserialize)]
19  struct Request {
20    words: Vec<String>,
21  }
22
23  #[derive(Debug, Serialize, Deserialize)]
24  struct Response {
25    doc_ids: Vec<[u8; 12]>,
26  }
27
28  #[post("/search")]
29  async fn search(req_body: String) -> impl Responder {
30    println!("Request: {req_body}");
31    let data: Request = serde_json::from_str(&req_body).unwrap();
32    let mut result =
↪   ENGINE.lock().unwrap().as_mut().unwrap().search(data.words).await.unwrap();
```

```
    result.dedup();

    print!("Found doc_ids:\n");
    for oid in &result {
      print!("{:?}\n", oid.to_hex());
    }

    let response = Response {
      doc_ids: result.iter().map(|oid| oid.bytes()).collect(),
    };

    return HttpResponse::Ok().body(serde_json::to_string(&response).unwrap());
  }

  #[tokio::main]
  async fn main() -> std::io::Result<()> {
    *ENGINE.lock().unwrap() = Some(engine::init_engine().await.unwrap());
    HttpServer::new(|| App::new().service(search))
      .bind(("localhost", 8080))?
      .run()
      .await
  }

```

6 Выводы

Выполнив первую лабораторную работу по курсу «Информационный поиск», я научился писать веб-скрапер на Python, обрабатывать документы и познакомился с MongoDB. Во второй лабораторной работе научился строить прямой и обратный индексы на Rust для поиска. При написании третьей лабораторной работы я изучил фреймворки Flask и httpx и написал с их помощью веб-сервис для поисковой выдачи и общения с поисковым движком. В четвертой лабораторной я изучил алгоритмы ранжирования, а также нечеткий поиск.

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.))