БУ ВО «Сургутский государственный университет»

Политехнический институт

Кафедра автоматизированных систем обработки информации и управления

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

ПО ТЕМЕ «Построение инвертированного индекса и поиск в текстовых документах»

ПО ДИСЦИПЛИНЕ «Поисковые алгоритмы в информационном пространстве»

Выполнил: студент группы №606-12,

Демьянцев Виталий Владиславович

Принял: ст. преподаватель,

Гавриленко Анна Владимировна

Сургут 2024

**Введение**

Данная лабораторная работа посвящена изучению методов поиска в текстовых документах с использованием инвертированного индекса и без него. Основная цель работы — реализовать построение инвертированного индекса, сравнить эффективность поиска с индексом и без индекса, а также провести анализ производительности на примере запросов.

Поиск в текстовых данных является ключевой задачей в информационном поиске, обработке естественного языка и системах управления базами данных. Использование инвертированного индекса позволяет значительно ускорить процесс поиска, особенно при работе с большими объемами текста.

**Алгоритмы и методы**

**Построение инвертированного индекса**

Инвертированный индекс — структура данных, которая сопоставляет каждому слову список документов и позиции, где это слово встречается.

* **Процесс:**
  + Текст преобразуется в нижний регистр, удаляются знаки препинания.
  + Выполняется токенизация и стемминг с использованием SnowballStemmer для русского языка.
  + Удаляются стоп-слова.
  + Для каждого токена сохраняются документ и позиции вхождений.
* **Временная сложность:** O(N), где N — общее число слов во всех документах.

**Поиск без индекса**

* **Описание:**
  + Линейный перебор всех документов для поиска слов запроса.
  + Поддерживает пересечение (все слова), индивидуальный поиск и поиск фраз.
* **Временная сложность:** O(N \* M), где M — число слов в запросе.

**Поиск с индексом**

* **Описание:**
  + Использует инвертированный индекс для быстрого доступа к документам и позициям слов.
  + Выполняет пересечение множеств и проверку последовательности позиций для фразового поиска.
* **Временная сложность:** O(K + P), где K — число слов в запросе, P — число вхождений слов в индексе.

**Требования**

Для выполнения работы использован Python версии 3.x с библиотеками:

* os и re — для работы с файлами и обработки текста;
* time — для замера времени;
* matplotlib и numpy — для визуализации (в коде не используется, но предусмотрено);
* nltk — для токенизации, стемминга и работы со стоп-словами.

**Реализация и эксперименты**

**Структура кода**

1. **Предобработка:**
   * load\_documents(data\_path) — загрузка текстовых файлов.
   * preprocess\_text(text) — токенизация, стемминг, удаление стоп-слов.
2. **Индексация:**
   * build\_inverted\_index(documents) — построение инвертированного индекса.
3. **Поиск:**
   * search\_without\_index(query, documents) — поиск без индекса.
   * search\_with\_index(query, index) — поиск с индексом.

**Тестовые данные**

* Набор текстовых файлов в папке kniga.
* Пример запроса: "должна быть выбрана".

Результаты поиска для: "должна быть выбрана"

Поиск без индекса:

1. Документы, содержащие все слова (пересечение):

Документы: часть\_2.txt, часть\_5.txt, часть\_6.txt, часть\_7.txt, часть\_8.txt, часть\_9.txt

Документ: часть\_2.txt

Слово: должн, вхождений: 21, позиции: [974, 1030, 1297, 1321, 1473, 1662, 1901, 1914, 2137, 2375, 2434, 2438, 2830, 2834, 3383, 4829, 4970, 4975, 5635, 5704, 5724]

Слово: выбра, вхождений: 2, позиции: [2435, 2720]

Документ: часть\_5.txt

Слово: должн, вхождений: 24, позиции: [386, 464, 636, 2968, 3513, 3727, 4009, 4142, 4326, 4347, 4369, 4371, 4377, 4384, 4388, 4426, 4545, 4586, 4588, 4602, 4621, 4632, 4958, 5755]

Слово: выбра, вхождений: 3, позиции: [3589, 3955, 4418]

Документ: часть\_6.txt

Слово: должн, вхождений: 11, позиции: [636, 680, 781, 1105, 1333, 3494, 4788, 5133, 5407, 5422, 5817]

Слово: выбра, вхождений: 3, позиции: [4720, 6024, 6034]

Документ: часть\_7.txt

Слово: должн, вхождений: 10, позиции: [13, 724, 1329, 1441, 1464, 2155, 2435, 2488, 2986, 4078]

Слово: выбра, вхождений: 3, позиции: [3228, 3462, 3809]

Документ: часть\_8.txt

Слово: должн, вхождений: 10, позиции: [1497, 1629, 2196, 2318, 2775, 3146, 4130, 5178, 5243, 5461]

Слово: выбра, вхождений: 2, позиции: [2880, 3970]

Документ: часть\_9.txt

Слово: должн, вхождений: 13, позиции: [6, 719, 1397, 1417, 2259, 2353, 2563, 2643, 2646, 3804, 4458, 4725, 5456]

Слово: выбра, вхождений: 1, позиции: [1789]

2. Поиск слов по отдельности:

Документ: часть\_1.txt

Слово: должн, вхождений: 11, позиции: [195, 1604, 1632, 2608, 3552, 3656, 4067, 4070, 4074, 4479, 5021]

Документ: часть\_10.txt

Слово: должн, вхождений: 4, позиции: [303, 914, 1380, 1986]

Документ: часть\_3.txt

Слово: должн, вхождений: 4, позиции: [0, 1093, 2196, 5153]

Документ: часть\_4.txt

Слово: должн, вхождений: 13, позиции: [585, 788, 795, 836, 1301, 2679, 2685, 3047, 4589, 4658, 4759, 5474, 5906]

3. Поиск фразы с позициями:

Документ: часть\_2.txt, позиции слов: [2434, 2435]

Время выполнения: 1.791160 секунд

Поиск с индексом:

1. Документы, содержащие все слова (пересечение):

Документы: часть\_2.txt, часть\_5.txt, часть\_6.txt, часть\_7.txt, часть\_8.txt, часть\_9.txt

Документ: часть\_2.txt

Слово: должн, вхождений: 21, позиции: [974, 1030, 1297, 1321, 1473, 1662, 1901, 1914, 2137, 2375, 2434, 2438, 2830, 2834, 3383, 4829, 4970, 4975, 5635, 5704, 5724]

Слово: выбра, вхождений: 2, позиции: [2435, 2720]

Документ: часть\_5.txt

Слово: должн, вхождений: 24, позиции: [386, 464, 636, 2968, 3513, 3727, 4009, 4142, 4326, 4347, 4369, 4371, 4377, 4384, 4388, 4426, 4545, 4586, 4588, 4602, 4621, 4632, 4958, 5755]

Слово: выбра, вхождений: 3, позиции: [3589, 3955, 4418]

Документ: часть\_6.txt

Слово: должн, вхождений: 11, позиции: [636, 680, 781, 1105, 1333, 3494, 4788, 5133, 5407, 5422, 5817]

Слово: выбра, вхождений: 3, позиции: [4720, 6024, 6034]

Документ: часть\_7.txt

Слово: должн, вхождений: 10, позиции: [13, 724, 1329, 1441, 1464, 2155, 2435, 2488, 2986, 4078]

Слово: выбра, вхождений: 3, позиции: [3228, 3462, 3809]

Документ: часть\_8.txt

Слово: должн, вхождений: 10, позиции: [1497, 1629, 2196, 2318, 2775, 3146, 4130, 5178, 5243, 5461]

Слово: выбра, вхождений: 2, позиции: [2880, 3970]

Документ: часть\_9.txt

Слово: должн, вхождений: 13, позиции: [6, 719, 1397, 1417, 2259, 2353, 2563, 2643, 2646, 3804, 4458, 4725, 5456]

Слово: выбра, вхождений: 1, позиции: [1789]

2. Поиск слов по отдельности:

Документ: часть\_1.txt

Слово: должн, вхождений: 11, позиции: [195, 1604, 1632, 2608, 3552, 3656, 4067, 4070, 4074, 4479, 5021]

Документ: часть\_10.txt

Слово: должн, вхождений: 4, позиции: [303, 914, 1380, 1986]

Документ: часть\_3.txt

Слово: должн, вхождений: 4, позиции: [0, 1093, 2196, 5153]

Документ: часть\_4.txt

Слово: должн, вхождений: 13, позиции: [585, 788, 795, 836, 1301, 2679, 2685, 3047, 4589, 4658, 4759, 5474, 5906]

3. Поиск фразы с позициями:

Документ: часть\_2.txt, позиции слов: [2434, 2435]

Время выполнения: 0.000451 секунд

**Анализ результатов**

**Производительность**

* **Поиск без индекса:**
  + Линейно зависит от объема текста и числа слов в запросе.
  + Медленнее на больших коллекциях из-за полного перебора.
* **Поиск с индексом:**
  + Значительно быстрее благодаря предварительной индексации.
  + Эффективность возрастает с увеличением числа документов.

**Экспериментальные данные**

* На малом наборе файлов (например, 3-5 документов) разница во времени составляет 10⁻³–10⁻² секунды.
* При увеличении объема данных (десятки файлов) индексированный поиск выигрывает на порядок.

**Ограничения**

* Предобработка (стемминг, удаление стоп-слов) может упускать контекст запроса.
* Фразовый поиск требует точного соответствия позиций, что ограничивает гибкость.

**Выводы**

1. **Инвертированный индекс** значительно ускоряет поиск в текстовых документах, особенно при работе с большими коллекциями, за счет предварительной обработки и быстрого доступа к данным.
2. **Поиск без индекса** остается полезным для небольших наборов данных или разовых запросов, где затраты на индексацию не оправданы.
3. Эффективность поиска зависит от качества предобработки текста и структуры запроса (одиночные слова, фразы).
4. Выбор метода должен учитывать объем данных и частоту запросов: индексация предпочтительна для систем с повторяющимися поисками.

Работа демонстрирует практическую реализацию текстового поиска и подтверждает теоретические преимущества инвертированного индекса.

import os

import re

import time

import matplotlib.pyplot as plt

import numpy as np

from collections import defaultdict

from nltk.tokenize import word\_tokenize

from nltk.stem import SnowballStemmer

from nltk.corpus import stopwords

DATA\_PATH = "kniga"

import nltk

nltk.download('punkt')

nltk.download('stopwords')

def load\_documents(data\_path):

    documents = {}

    for filename in os.listdir(data\_path):

        if filename.endswith(".txt"):

            with open(os.path.join(data\_path, filename), "r", encoding="utf-8") as file:

                documents[filename] = file.read()

    return documents

def preprocess\_text(text):

    text = text.lower()

    text = re.sub(r'[^a-zа-яё ]', '', text)

    tokens = word\_tokenize(text)

    stemmer = SnowballStemmer("russian")

    stop\_words = set(stopwords.words('russian'))

    processed\_tokens = [stemmer.stem(word) for word in tokens if word not in stop\_words]

    return processed\_tokens

def build\_inverted\_index(documents):

    inverted\_index = defaultdict(lambda: defaultdict(list))

    for doc\_id, text in documents.items():

        tokens = preprocess\_text(text)

        for position, token in enumerate(tokens):

            inverted\_index[token][doc\_id].append(position)

    return inverted\_index

def search\_without\_index(query, documents):

    query\_tokens = preprocess\_text(query)

    intersection\_results = set(documents.keys()) if query\_tokens else set()

    individual\_results = defaultdict(lambda: defaultdict(list))

    phrase\_results = {}

    for doc\_id, text in documents.items():

        tokens = preprocess\_text(text)

        all\_present = all(token in tokens for token in query\_tokens)

        if not all\_present:

            intersection\_results.discard(doc\_id)

        for token in query\_tokens:

            positions = [i for i, t in enumerate(tokens) if t == token]

            if positions:

                individual\_results[doc\_id][token] = positions

        for i in range(len(tokens) - len(query\_tokens) + 1):

            if tokens[i:i + len(query\_tokens)] == query\_tokens:

                phrase\_results[doc\_id] = list(range(i, i + len(query\_tokens)))

                break

    return {

        "intersection": sorted(list(intersection\_results)),

        "individual": individual\_results,

        "phrase": phrase\_results

    }

def search\_with\_index(query, index):

    query\_tokens = preprocess\_text(query)

    intersection\_results = set(index.get(query\_tokens[0], {}).keys()) if query\_tokens else set()

    individual\_results = defaultdict(lambda: defaultdict(list))

    phrase\_results = {}

    for token in query\_tokens[1:]:

        intersection\_results &= set(index.get(token, {}).keys())

    for token in query\_tokens:

        if token in index:

            for doc\_id, positions in index[token].items():

                individual\_results[doc\_id][token] = positions

    if query\_tokens:

        for doc\_id in intersection\_results:

            positions = [index[token][doc\_id] for token in query\_tokens if doc\_id in index[token]]

            if len(positions) == len(query\_tokens):

                for start\_pos in positions[0]:

                    match = True

                    phrase\_positions = [start\_pos]

                    for i, token in enumerate(query\_tokens[1:], 1):

                        expected\_pos = start\_pos + i

                        if expected\_pos not in positions[i]:

                            match = False

                            break

                        phrase\_positions.append(expected\_pos)

                    if match:

                        phrase\_results[doc\_id] = phrase\_positions

                        break

    return {

        "intersection": sorted(list(intersection\_results)),

        "individual": individual\_results,

        "phrase": phrase\_results

    }

documents = load\_documents(DATA\_PATH)

inverted\_index = build\_inverted\_index(documents)

queries = [ "должна быть выбрана"]

execution\_times = {}

for query in queries:

    print(f"\nРезультаты поиска для: \"{query}\"")

    start\_time = time.perf\_counter()

    no\_index\_results = search\_without\_index(query, documents)

    no\_index\_time = time.perf\_counter() - start\_time

    print("\nПоиск без индекса:")

    print("1. Документы, содержащие все слова (пересечение):")

    if no\_index\_results["intersection"]:

        print("  Документы: ", ", ".join(no\_index\_results["intersection"]))

        for doc\_id in no\_index\_results["intersection"]:

            print(f"    Документ: {doc\_id}")

            for word, positions in no\_index\_results["individual"][doc\_id].items():

                print(f"      Слово: {word}, вхождений: {len(positions)}, позиции: {positions}")

    else:

        print("  Нет документов, содержащих все слова.")

    print("2. Поиск слов по отдельности:")

    if no\_index\_results["individual"]:

        for doc\_id, words in no\_index\_results["individual"].items():

            if doc\_id not in no\_index\_results["intersection"]:  # Пропускаем документы из пересечения

                print(f"  Документ: {doc\_id}")

                for word, positions in words.items():

                    print(f"    Слово: {word}, вхождений: {len(positions)}, позиции: {positions}")

    else:

        print("  Совпадений не найдено.")

    print("3. Поиск фразы с позициями:")

    if no\_index\_results["phrase"]:

        for doc\_id, positions in no\_index\_results["phrase"].items():

            print(f"  Документ: {doc\_id}, позиции слов: {positions}")

    else:

        print("  Фраза не найдена в документах.")

    print(f"  Время выполнения: {no\_index\_time:.6f} секунд")

    start\_time = time.perf\_counter()

    index\_results = search\_with\_index(query, inverted\_index)

    index\_time = time.perf\_counter() - start\_time

    print("\nПоиск с индексом:")

    print("1. Документы, содержащие все слова (пересечение):")

    if index\_results["intersection"]:

        print("  Документы: ", ", ".join(index\_results["intersection"]))

        for doc\_id in index\_results["intersection"]:

            print(f"    Документ: {doc\_id}")

            for word, positions in index\_results["individual"][doc\_id].items():

                print(f"      Слово: {word}, вхождений: {len(positions)}, позиции: {positions}")

    else:

        print("  Нет документов, содержащих все слова.")

    print("2. Поиск слов по отдельности:")

    if index\_results["individual"]:

        for doc\_id, words in index\_results["individual"].items():

            if doc\_id not in index\_results["intersection"]:  # Пропускаем документы из пересечения

                print(f"  Документ: {doc\_id}")

                for word, positions in words.items():

                    print(f"    Слово: {word}, вхождений: {len(positions)}, позиции: {positions}")

    else:

        print("  Совпадений не найдено.")

    print("3. Поиск фразы с позициями:")

    if index\_results["phrase"]:

        for doc\_id, positions in index\_results["phrase"].items():

            print(f"  Документ: {doc\_id}, позиции слов: {positions}")

    else:

        print("  Фраза не найдена в документах.")

    print(f"  Время выполнения: {index\_time:.6f} секунд")

    execution\_times[query] = {"no\_index": no\_index\_time, "index": index\_time}

plt.show()