

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 1

з дисципліни «Програмне забезпечення інформаційно-пошукових систем. Частина 2. Методи організації пошуку інформації»

тема «Реалізація теоретико-множинної моделі подання документів»

|  |  |  |
| --- | --- | --- |
| Виконав  Студент IV курсу  групи КП-01  Беліцький Олександр Сергійович |  |  |

Київ 2024

**Мета**

Реалізувати теоретико-множинну (стандартну булеву) модель подання документів, таким чином ознайомившись з найбільш поширеними моделями та набувши практичні навички з їх реалізації.

**Індивідуальне завдання**

1. Для реалізації програмного забезпечення в ході лабораторної роботи може використовуватись будь-який стек (мова програмування, фреймворк і так далі) технологій.

2. Програмне забезпечення може мати будь-який з перелічених інтерфейсів користувача: консольний, веб, мобільний, настільний.

3. Розроблене програмне забезпечення має використовувати стандартну

булеву модель подання документів.

4. Робота з розробленим програмним забезпеченням має бути поділена на наступні етапи:

a. Введення множини індексних термів.

i. Користувач повинен мати можливість ввести будь-яку невід’ємну кількість індексних термів на свій вибір (хоча б один індексний терм є обов’язковим).

ii. Допускається (на вибір студента) реалізація введення множини індексних термів за допомогою її зчитування з файлу будь-якого формату. У такому випадку, на данному етапі користувач повинен вводити шлях до цього файлу.

iii. Після закінчення введення множини індексних термів, користувач повинен автоматично перейти на наступний етап.

b. Введення колекції документів.

i. Користувач повинен мати можливість ввести будь-яку невід’ємну кількість документів (хоча б один документ у колекції є обов’язковим).

ii. Допускається (на вибір студента) реалізація введення колекції документів за допомогою її зчитування з окремих текстових файлів, розміщених у певній директорії (один текстовий файл = один документ). У такому випадку, на даному етапі користувач вводить шлях до директорії з файлами.

iii. Після закінчення введення, користувач автоматично переходить на наступний етап.

c. Виконання пошукових запитів.

i. На даному етапі користувач вводить пошуковий запит і переглядає його результати.

ii. Пошуковий запит вводиться у нормальній кон’юнктивній або диз’юнктивній формі, відповідно до варіанту студента.

iii. Після виконання пошукового запиту, користувач повинен залишатися на цьому ж етапі, з можливістю ввести новий пошуковий запит.

Варіант визначається за порядковим номером студента у списку його групи. Якщо номер студента є непарним числом, то пошуковий запит повинен вводитися у нормальній кон’юнктивній формі. Якщо номер студента є парним числом, то пошуковий запит повинен вводитися у нормальній диз’юнктивній формі.

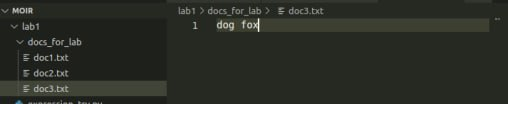
**Хід роботи**

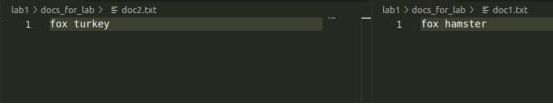
Спочатку визначимось із варіантами індивідуального завдання. Роботу будемо виконувати в IDE Visual Studio Code, використовуючи мову програмування Python. Розробимо консольний інтерфейс, через який будемо задавати шляхи до файлу термів, де кожен терм пишеться з нового рядка, та до директорії з документами. Далі користувач вводить запити у нормальній диз’юнктивній формі.

Код програми:

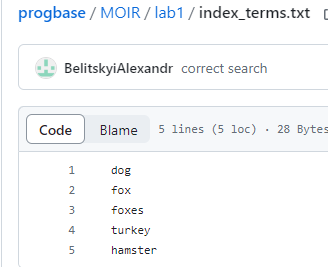
|  |
| --- |
| main.py |
| # index terms  def read\_index\_terms\_from\_file(file\_path):  with open(file\_path, 'r', encoding='utf-8') as file:  index\_terms = file.readlines()  index\_terms = [term.strip() for term in index\_terms]  return index\_terms  #file\_path = input("Enter path to the file with index terms: ")  file\_path = 'lab1/index\_terms.txt'  index\_terms = read\_index\_terms\_from\_file(file\_path)  print("Set of the index terms:", index\_terms)  #docs  import os  def read\_documents\_from\_directory(directory\_path):  documents = []  for filename in os.listdir(directory\_path):  if filename.endswith('.txt'):  with open(os.path.join(directory\_path, filename), 'r') as file:  document\_content = file.read()  documents.append(document\_content)  return documents  #directory\_path = input("Enter path to derictory with docs: ")  directory\_path = "lab1/docs\_for\_lab"  documents = read\_documents\_from\_directory(directory\_path)  print("Number of docs:", len(documents))  #search  '''  import re  # search  def execute\_search\_query(index\_terms, documents):  while True:  query = input("Enter the search query or 'exit' to quit: ")  if query.lower() == 'exit':  break    # dis  disjunctions = query.split('^')    relevant\_documents = set()  for disjunction in disjunctions:  conjunctions = disjunction.split('\*')  relevant\_docs\_for\_disjunction = set(range(1, len(documents) + 1))  for conjunction in conjunctions:  # terms with ! and without  terms\_with\_negation = re.findall(r'(!?\w+)', conjunction)  required\_terms = set()  excluded\_terms = set()  for term in terms\_with\_negation:  if term.startswith('!'):  excluded\_terms.add(term[1:]) # Відсутні терми  else:  required\_terms.add(term) # Терми, які повинні бути в документі  # search  relevant\_docs\_for\_conjunction = set()  for i, document in enumerate(documents):  # terms in doc  if all(term in document for term in required\_terms):  # !terms not in doc  if not any(term in document for term in excluded\_terms):  relevant\_docs\_for\_conjunction.add(i + 1)  relevant\_docs\_for\_disjunction &= relevant\_docs\_for\_conjunction  relevant\_documents |= relevant\_docs\_for\_disjunction    # print  if relevant\_documents:  print("Relevant documents for the query '{}':".format(query))  for doc\_index in relevant\_documents:  print("Document #{}:".format(doc\_index))  print("Relevant terms:", query)  print("Document content:", documents[doc\_index - 1])  else:  print("No relevant documents found for the query '{}'.\n".format(query))  execute\_search\_query(index\_terms, documents)  '''  #search  def execute\_search\_query(index\_terms, documents):  while True:  query = input("Enter the search query or 'exit' to quit: ")  if query.lower() == 'exit':  print("Exiting...")  break    # query for disjunc parts  disjunctions = query.split('^')    relevant\_documents = set()  for disjunction in disjunctions:  conjunctions = disjunction.split('\*')  relevant\_docs\_for\_disjunction = set(range(1, len(documents) + 1))  for conjunction in conjunctions:  # search in each part of disjuncs  relevant\_docs\_for\_conjunction = set()  for i, document in enumerate(documents):  if all(term.strip() in document for term in conjunction.split()):  relevant\_docs\_for\_conjunction.add(i + 1)  relevant\_docs\_for\_disjunction &= relevant\_docs\_for\_conjunction  relevant\_documents |= relevant\_docs\_for\_disjunction    # print  if relevant\_documents:  print("---\nRelevant documents for the query '{}':".format(query))  for doc\_index in relevant\_documents:  print("Document #{}:".format(doc\_index))  print("Relevant terms:", query)  print("Document content:", documents[doc\_index - 1])  print("---\n")  else:  print("No relevant documents found for the query '{}'.\n".format(query))  execute\_search\_query(index\_terms, documents) |

Створимо вхідні дані. На скріншотах нижче видно перелік документів, пошук яких буде виконуватись, та текст всередині них.

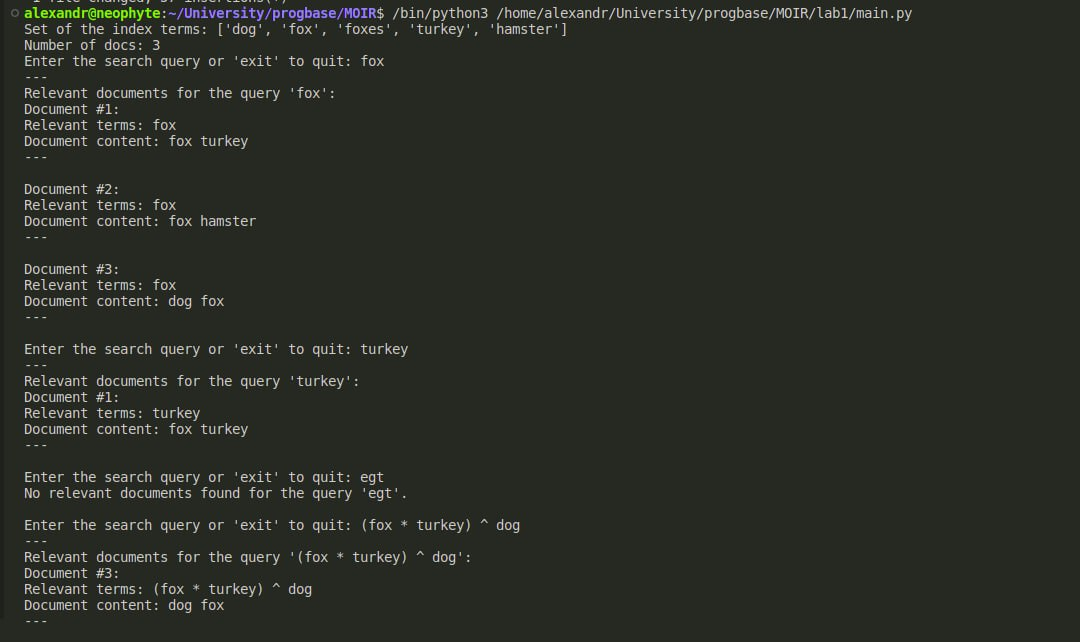




Файл з термами:



Продемонструємо роботу застосунку:



**Висновки**

У результаті лабораторної роботи ми здійснили реалізацію теоретико-множинної, а саме стандартної булевої, моделі подання документів за допомогою мови програмування Python.

Ми використали IDE Visual Studio Code для написання програми та реалізації консольного інтерфейсу. За допомогою цієї програми ми змогли читати дані з файлів термів та документів, а також виконувати пошук за запитами користувача у нормальній диз'юнктивній формі.

Основні кроки реалізації включали в себе читання вхідних даних з файлів, створення необхідних структур даних для представлення та обробки інформації, а також виконання пошуку за запитами користувача та виведення результатів.