**Міністерство освіти і науки України**

**Національний технічний університет України**

**"Київський політехнічний інститут імені Ігоря Сікорського"**

**Фізико-технічний інститут**

«Основи аналізу алгоритмів»

**Проект**

Виконали:

студент гр. ФБ-92

Курганський Л.С.

Кудряшов М.О.

Перевірив:

Чорний О. М.

Київ – 2021

**Варіант №8**: Колекція текстових документів із можливістю повнотекстового пошуку (1)

В даному варіанті передбачається, що по створеним колекціям будуються інвертовані індекси (inverted index), які і використовуються для прискорення виконання пошукових команд. Для побудови індексу програма повинна аналізувати документи, які додаються до колекцій, виділяти з них окремі слова та зберігати список позицій, де дане слово зустрічається в межах даного документу. Для простоти, в межах даного практикуму вважаємо, що “словом” в документі вважається довільна послідовність, яка складається із алфавітно-цифрових символів та символу **\_** (в термінах регулярних виразів - **[a-zA-Z0-9\_]+**). Решта символів, які зустрічаються в документах визначають межі слів та можуть ігноруватись.

Перелік команд, які необхідно реалізувати:

1. **CREATE collection\_name;** - створення нової колекції з назвою **collection\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Collection collection\_name has been created”**.  
     
   **Приклад:**  
   **CREATE wiki\_articles;**
2. **INSERT collection\_name “value”;** - додавання нового документу до колекції **collection\_name**. В якості результату команда повертати певне повідомлення для користувача, наприклад **“Document has been added to collection\_name”**.  
     
   **Приклад:**  
   **INSERT wiki\_articles “The word 'algorithm' has its roots in Latinizing the ... ”;**
3. **PRINT\_INDEX collection\_name;** - виведення на екран внутрішньої структури інвертованого індексу, побудованого для колекції **collection\_name**. Вивід повинен включати виділені ключові слова, документи, яким вони належать та позиції, в яких вони зустрічаються в даних документах.  
     
   **Приклад:**  
   Для індексу, що містить документи **“to be or not to be”** та **“to go or not to go”**, яким було присвоєно ідентифікатори **d1** та **d2** відповідно, вивід повинен бути схожим до наступного:  
     
   **"be":**

**d1 -> [2, 6]**

**"go":**

**d2 -> [2, 6]**

**"not":**

**d1 -> [4]**

**d2 -> [4]**

**"or":**

**d1 -> [3]**

**d2 -> [3]**

**"to":**

**d1 -> [1, 5]**

**d2 -> [1, 5]**

1. **SEARCH collection\_name [WHERE query];** - пошук документів по колекції **collection\_name**.

Якщо вказано ключове слово **WHERE** та слідуючий за ним пошуковий запит, то команда повинна включити до вибірки тільки документи, які задовольняють даний запит. В іншому випадку, необхідно повернути всі документи зі вказаної колекції. Визначення запиту **query** має наступний вигляд:   
 **query := “keyword”   
 | “prefix”\*  
 | “keyword\_1” <N> “keyword\_2”**   
Пояснення:

* **“keyword”** - до вибірки повинні бути включені документи, які містять слово **“keyword”**.
* **“prefix”\*** - префіксний пошук, тобто до вибірки повинні бути включені документи, які містять слово, що починається із **“prefix”**. Наприклад, якщо **query = “універ”\***, то у вибірку повинні бути включені всі документи, що містять слова **“університет”**, **“універсам”**, **“універсал”** тощо.
* **“keyword\_1” <N> “keyword\_2”** - до вибірки повинні бути включені документи, які містять слова **“keyword\_1”** та **“keyword\_1”** на відстані **N** слів одне від одного. Позиція в документі та взаємний порядок слів не важливі. Наприклад, документ **“The word 'algorithm' has its roots in Latinizing the nisba ...”** задовольняє таким запитам: **“word” <1> “algorithm”**, **“algorithm” <1> “word”**, **“word” <4> “roots”**, **“roots” <4> “word”** тощо.

Можна вважати, що **“keyword”**, **“prefix”**, **“keyword\_1”** та **“keyword\_2”** вище є коректним словами, відповідно до визначення даного на початку даного варіанту, тобто задовольняють регулярному виразу **[a-zA-Z0-9\_]+**.  
 Пошук документів, що задовольняють певному запиту повинен проводитись без урахування регістру документів та ключових слів у цих запитах. Тобто, наприклад, документ **“The word 'algorithm' has its roots in Latinizing the nisba ...”** задовольняє усім наступним пошуковим запитам: **“algorithm”**, **“ALGORithm”**, **“ALGo”\*** , **“WORD” <2> “has”** тощо.  
 В якості результату команда повинна повертати список знайдених документів. Порядок рядків може бути довільним.

**Приклади команд:**

**SEARCH wiki\_articles;**

**SEARCH wiki\_articles WHERE “algorithm”;**

**SEARCH wiki\_articles WHERE “latin”\*;**

**SEARCH wiki\_articles WHERE “has” <2> “roots”;**

# 

# 1. Парсер

Спочатку відбувається очищення введеної команди.

def cleaning\_text(text):

    # cleaning symbols after ;

    find\_end = lambda x: len(x) if x.find(";") == -1 else x.find(";")

    text = text[:find\_end(text)]

    # cleaning from space

    text = text.split()

    text = ' '.join([elem for elem in text])

    return text

Потім визначення команди:

def define\_command(command):

    key\_words = ["CREATE", "INSERT", "SEARCH", "REMOVE", "EXIT"]

    command = command.upper().split()

    # empy command or only command without parameters

    if not len(command) or (len(command) < 2 and command[0] != "EXIT"):

        return " ", -7

    if command[0] in key\_words:

        return command[0], 100

    else:

        return " ", -7

Далі для кожної команди відбувається свій процес розбору:

* **CREATE**

def parse\_create(command):

    command = command.split()

    name = command[1]

    # incorrect name

    if re.search("[a-zA-Z]", name[0]) == None: # first letter

        return name, -1

    if re.search("[a-zA-Z0-9\_]\*", name) == None: # other symbols

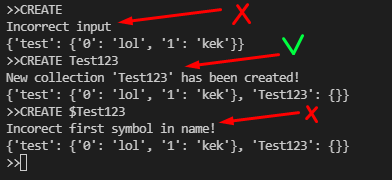
        return name, -2

    # too many parameters

    if len(command) > 2:

        return name, -4

    return name, 100



* **INSERT:**

def parse\_insert(command):

    command = command.split()

    name = command[1]

    text = ""

    if len(command) < 3: # without text

        return name, text, -12

    index = 0

    i = 0

    while (i < len(command)):

        if command[i][-1] == "\"":

            index = i

        i += 1

    i -= 1

    # incorrect ""

    if command[2][0] != "\"" or command[-1][-1] != "\"": # missing double quote

        return name, text, -6

    # too many parameters

    if i != index:

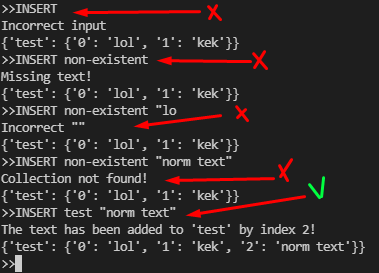
        return name, text, -4

    # getting text

    text = ' '.join([elem for elem in command[2:index+1]])

    text = text.strip("\"")

    return name, text, 100



* **REMOVE:**

def parse\_remove(command):

    command = command.split()

    case = 0

    name = command[1]

    index = 0

    if len(command) == 2:

        case = 1

    elif len(command) == 3:

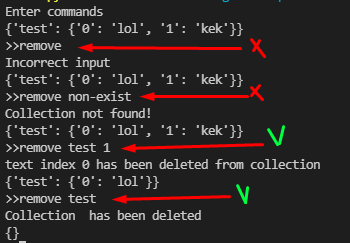
        case = 2

        index = command[2]

    else:

        return case, name, index, -4

    return case, name, index, 100



* **SEARCH:**

def parse\_search(command):

    command = command.split()

    name = command[1]

    condition = ""

    # finding WHERE

    if len(command) >= 3:

        # incorrect WHERE

        if command[2].upper() != "WHERE":

            return name, condition, -8

        # missing condition

        if len(command) == 3:

            return name, condition, -9

        # getting condition

        condition = command[3:]

        condition = ' '.join([elem for elem in condition])

    # without WHERE  
    return name, condition, 100

def parse\_condition(condition):

    if not len(condition): # without WHERE

        case = 0

    # keyword

    elif condition[0] == "\"" and condition[-1] == "\"" \

        and condition.count("\"") == 2:

        condition = condition[1:-1]

        case = 1

    # keyword + '\*'

    elif condition[0] == "\"" and condition[-1] == "\*":

        condition = condition[1:-2]

        case = 2

    # keyword1 <N> keyword2

    elif condition[0] == "\"" and condition[-1] == "\"" \

        and condition.count("\"") == 4 and condition.index("<") < condition.index(">"):

        keyword1 = condition[1:condition.index("\"", 1,-1)]

        keyword2 = condition[condition.index("\"", condition[1:].find("\"")+2, -1)+1:-1]

        N = condition[condition.index("<")+1:condition.index(">")]

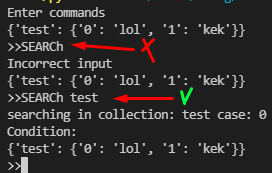
        condition = [keyword1, keyword2, int(N)]

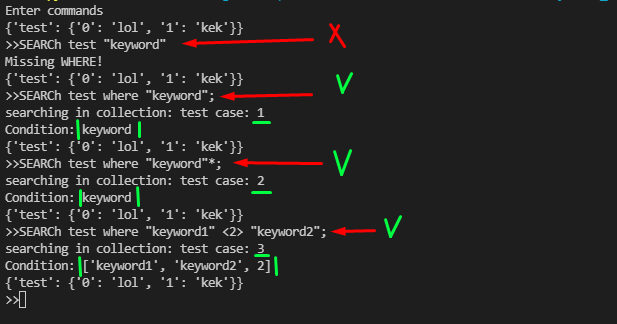
        case = 3

    else:

        return -1, condition, -10

    return case, condition, 100





Функція визначає три випадки пошуку згідно варіанту:

1. "keyword";
2. "prefix"\*;
3. "keyword" <N> "keyword";

Крім того була створена функція для обробки помилок:

def check\_status(status, name ="", index = 0):

    statuses = {

        4: f"text index {index} has been deleted from collection {name}",

        3: f"Collection {name} has been deleted",

        2: f"The text has been added to '{name}' by index {index}!",

        1: f"New collection '{name}' has been created!",

        0: "Command not found!",

        -1: "Incorect first symbol in name!",

        -2: "Incorect symbol in name!",

        -3: "The name is used!",

        -4: "Too many parameters!",

        -5: "Collection not found!",

        -6: "Incorrect \"\"",

        -7: "Incorrect input",

        -8: "Missing WHERE!",

        -9: "Missing condition!",

        -10: "Incorrect syntax!",

        -11: "Incorrect index",

-12: "Missing text!"

    }

    if status != 100:

        print(statuses[status])