Московский Авиационный Институт

(национальный исследовательский университет)

Кафедра 806

Курсовой проект по дисциплине

«Дискретный анализ»

на тему «Текстовый поиск»

Выполнил:

студент гр. М8о-307Б

Баранов Антон

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_

Москва. 2018

**Условие**

Необходимо разработать программу, реализующую текстовый поиск, т.е. выяснить вхождение какого-либо слова (или более сложного запроса) в базе текстов. В качестве базы данных использовать статьи из английской Википедии.

**Метод решения**

1. Изучение теоретических материалов
2. Изучение алгоритма работы поисковых систем (в упрощенном виде)
3. Реализация программы “crawler”
4. Реализация программы индексатора
5. Реализация программы обработки сложных запросов
6. Реализация программы взаимодействия с пользователем

**Описание алгоритма**

За основу данного алгоритма был взят алгоритм работы поисковых систем Google и Yandex, но в гораздо более простом виде.

Работа программы состоит из четырех этапов, за каждый из которых отвечает одна из подпрограмм:

1. Начало работы с пользователем и получение запроса
2. Выгрузка данных с Википедии в базу
3. Индексирование файлов в базе
4. Обработка запроса и генерация выходных данных

Взаимодействие между подпрограммами происходит в основном через файлы.

**Создание базы**

За этот этап отвечает кроулер. Он, используя библиотеку requests посылает запросы по необходимым адресам, далее парсит полученные данные с помощью библиотеки bs4 и функции BeautifulSoup. Итоговый текст записывается в файлы, находящиеся в базе. Для более удобного создания файловой системы базы используются библиотеки os и shutil.

**Индексирование базы**

За этот этап отвечает индексатор. Он получает список файлов в базе и начинает её обработку. С помощью структуры данных «множество», удобно предоставляемое языком Python, строится множество всех очищенных слов во всей базе. Далее для каждого слова из этого множества индексируется наличие в файлах базы. Слово и список его файлов хранятся в структуре данных «словарь».

**Обработка поступивших запросов**

Запросы поступают в виде логического выражения.

*Пример1:*

*word*

*Пример2:*

*word1 & (word2 | word3)*

Если в первом примере необходимо проверить лишь нахождение в базе одного слова, то во втором уже значительно более сложное выражение. Для его обработки используется так называемая обратная польская нотация. Сначала все запросы переводятся в постфиксную форму (альтернативное название), далее происходит выполнение логических действий. Для этого каждое слово заменяется вектором его вхождений в базу. Размерность этого вектора = (1 х кол-во статей в базе). Если слово находится в какой-либо статье, то значение на месте, соответствующем порядковому номеру файла в базе, в векторе равно единице, иначе ноль. Над векторами происходят логические операции умножения и сложения, также дополнительно реализованные. После полной обработки запроса остается список файлов, удовлетворяющих ему.

*Пример инфиксной (обычной) нотации:*

*a \* b + c*

*Пример постфиксной (обратной польской) нотации:*

*a b \* c +*

Эта нотация очень удобна при машинном решении выражений, также легко видоизменяется для логических операций, если учесть, что логическое «И» обладает свойствами умножения, а «ИЛИ» – сложения.

**Взаимодействие с пользователем**

Выполнение команд пользователя и управление проектом на более высоком уровне осуществляет подпрограмма dialog.py. С помощью неё можно выбрать режим работы (ручной или пакетный), также в ней происходит объединение работ всех подпрограмм в единое целое и генерируются выходные данные.

**Выполнение в консоли**

Пакетный режим:

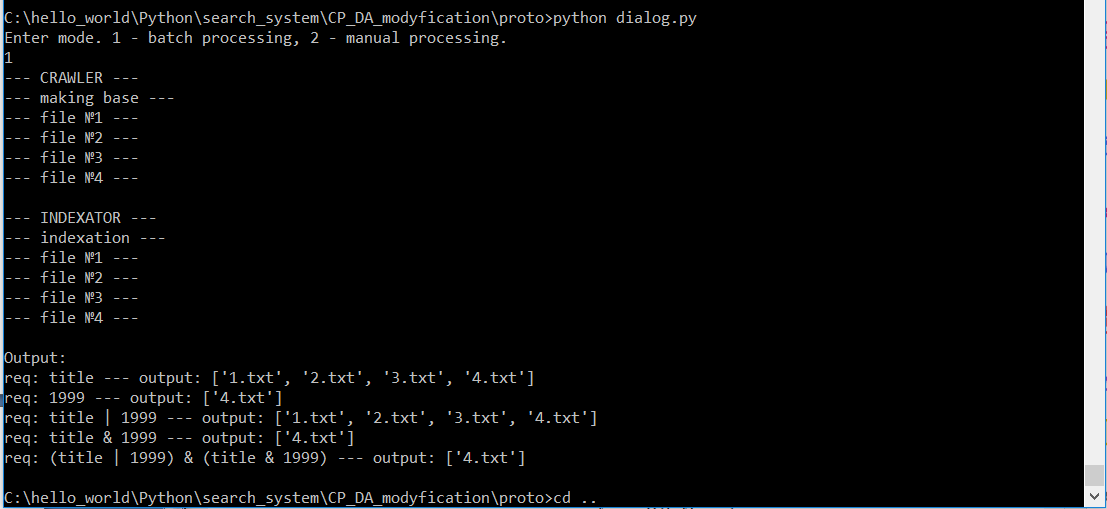


Рис.1 Выполнение в пакетном режиме

В файле requests.txt содержалось 4 названия статей. Кроулером была сгенерирована база из этих статей, выгруженных с Википедии. Индексатор проиндексировал все файлы. Далее в dialog.py были получены выходные данные, сохраненные в файл output.txt и продублированные на консоль.

Ручной режим:

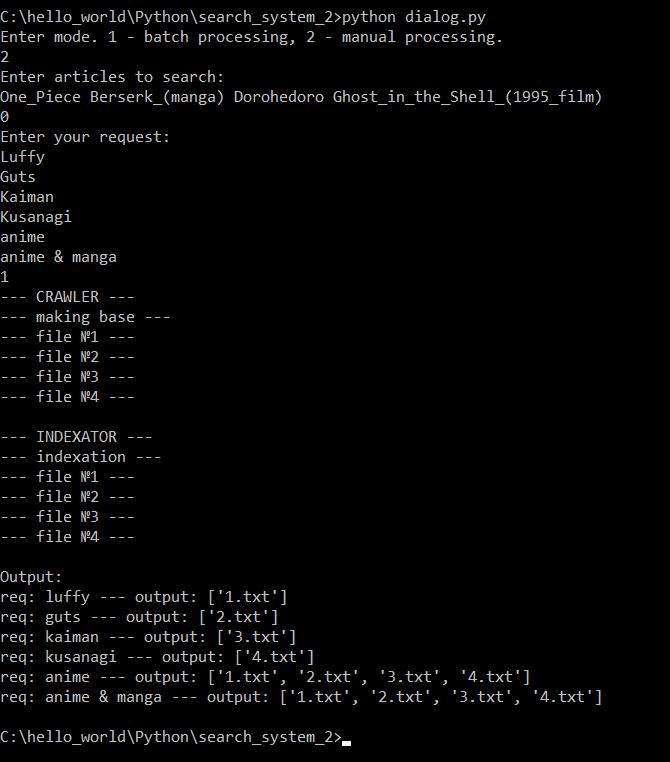


Рис.2 Выполнение в ручном режиме

Выполнение всех этапов поиска произошло аналогично пакетному режиму, за исключением того, что мы вручную определили запросы и базовые статьи.

**Недостатки**

Недостатками, которые возможно исправить, что улучшит данный проект, являются скорость выполнения индексации и поиска, нагруженность оперативной памяти. Можно было бы ввести постраничное отображение больших документов, не держать весь словарь индексации в оперативной памяти, оптимизировать алгоритм поиска. Также считаю, что к данной программе можно сделать хороший графический интерфейс, для большего удобства пользователя. Но так как работа учебная и основная её цель выполнена, то это не такие уж и критические недостатки, тем более что пути их решения видны уже сейчас.

**Выводы**

Данная работа является достаточно практичной и помогает отточить те навыки, которые будут наиболее нужны при дальнейшей работе в этой сфере. Выполнение этого курсового проекта было достаточно интересно и я узнал много нового о парсинге и онлайн-запросах. Язык Python очень хорошо подходит для моего задания, т.к. предоставляет мощные и при этом простые инструменты для работы с текстом, массивами и, как оказалось, с интернет-запросами.

Также пришлось столкнуться и с проблемами, на первый взгляд не связанными с моим заданием, например, парсинг входных запросов. Это оказалось неожиданно трудоёмко, так как пришлось реализовывать обратную польскую нотацию. Но это даже хорошо, так как я отточил свои навыки и в этом вопросе тоже.

*Приложение 1*

**Исходный код**

ссылка на git-репозиторий: <https://github.com/NoruNoruBim/Python/tree/master/searching_system_2_0>

**dialog.py**

import crawler

import indexator

import calc\_expr

def start():

noa = crawler.make\_base()

index = indexator.indexation(noa)

requests = []

output = []

with open("requests.txt", 'r', encoding="utf8") as file:# достаем запросы из файла

for line in file:

requests += [line.lower()]

for request in requests:# обрабатываем каждый запрос

tmp = calc\_expr.calc(request, index, noa)# итог, но в бинарном виде

tmp2 = []

for i in range(len(tmp)):# переводим в пользовательский вид

if tmp[i]: tmp2 += [str(i + 1) + ".txt"]

output += [tmp2]

# печатаем итоговый итог

print("\nOutput:")

with open("output.txt", 'w') as file:

for i in range(len(requests)):

print("req: " + requests[i].strip() + " --- output: " + str(output[i]))

file.write("req: " + requests[i].strip() + " --- output: " + str(output[i]) + '\n')

def main():

print("Enter mode. 1 - batch processing, 2 - manual processing.")

tmp = input()

if tmp == '2':

articles = []

print("Enter articles to search:")

while tmp != '0':

tmp = input()

articles += tmp.split()

with open("articles.txt", 'w') as fi:

for i in range(len(articles) - 1):

fi.write(articles[i] + '\n')

print("Enter your request:")

requests = []

while tmp != '1':

tmp = input()

requests += [tmp]

with open("requests.txt", 'w') as req:

for i in range(len(requests) - 1):

req.write(requests[i] + '\n')

start()

main()

**crawler.py**

import requests

from bs4 import BeautifulSoup

import os, shutil

def make\_base():

print("--- CRAWLER ---")

url\_array = []

urls\_new = set()

with open("articles.txt", 'r') as url\_s:# make array of articles url

for url in url\_s:

url\_array += ["https://en.wikipedia.org/wiki/" + url.strip()]

shutil.rmtree("data", True)

os.mkdir("data")

print("--- making base ---")

for i in range(len(url\_array)):

print("--- file №" + str(i + 1) + " ---")

request = requests.get(url\_array[i])# реквестируем конкретную ссылку

soup = BeautifulSoup((request.text), "lxml")# преобразуем в текст

main\_information = soup.find('div', {'id' : 'bodyContent'})# выискиваем то что нужно

filename = str(i + 1) + '.txt'

with open("data/" + filename, 'w', encoding='utf8') as file:# записываем выгруженную с википедии статью в нашу базу

file.write(main\_information.text)

return len(url\_array)

**indexator.py**

def clear(word):

while len(word) > 1:

if word[0].isalnum() and word[-1].isalnum(): break

if not word[0].isalnum(): word = word[1:]

if not word[-1].isalnum(): word = word[:-1]

# something else

return word

def make\_set(filename):

s = set()

with open("data/" + str(filename) + ".txt", 'r', encoding='utf8') as article:

for line in article:

for word in line.strip().split():

word = clear(word)

s.update({word.lower()})

return s

def indexation(noa):

print("\n--- INDEXATOR ---")

all = set()

part = set()

index = dict()

for i in range(1, noa + 1):# make set of all words

all.update(make\_set(i))

print("--- indexation ---")

for i in range(1, noa + 1):# indexation

print("--- file №" + str(i) + " ---")

part = make\_set(i)

for word in all:

if word in part:

if word not in index.keys():

index.update({word : [i]})

else:

index[word] += [i]

return index

**calc\_expr.py**

from pythonds.basic.stack import Stack

#import operator

import numpy as np

def arr(word, index, noa):

tmp2 = np.zeros(noa)

if word in index.keys():

tmp1 = index[word]

for i in tmp1:# vector with ones and zeros

tmp2[i - 1] = 1

return tmp2

def fix(line):

count = 0

for char in line:

count += 1

if char == '(':

line = line[:count] + ' ' + line[count:]

count += 1

if char == ')':

line = line[:count - 1] + ' ' + line[count - 1:]

count += 1

return line

def infixToPostfix(infixexpr):

infixexpr = fix(infixexpr)

prec = {}

#prec['\*'] = 3

prec['~'] = 4

prec['&'] = 3

#prec['/'] = 3

#prec['+'] = 2

prec['|'] = 2

#prec['-'] = 2

prec['('] = 1

opStack = Stack()

postfixList = []

tokenList = infixexpr.split()

for token in tokenList:

#if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" or token in "0123456789":

if token.isalnum():

postfixList.append(token)

elif token[0] == '(':

opStack.push(token)

elif token == ')':

topToken = opStack.pop()

while topToken != '(':

postfixList.append(topToken)

topToken = opStack.pop()

else:

while (not opStack.isEmpty()) and \

(prec[opStack.peek()] >= prec[token]):

postfixList.append(opStack.pop())

opStack.push(token)

while not opStack.isEmpty():

postfixList.append(opStack.pop())

return " ".join(postfixList)

'''

print(infixToPostfix("A \* B + C \* D"))

print(infixToPostfix("(A | B) & C | (D | E) & (F | G)"))'''

def log\_mul(a, b):

return a \* b

def log\_sum(a, b):

for i in range(len(a)):

if a[i] + b[i] >= 1: a[i] = 1

return a

def log\_not(a):

for i in range(len(a)):

if a[i] == 1: a[i] = 0

else: a[i] = 1

def calc(expr, index, noa):

expr = infixToPostfix(expr)

#OPERATORS = {'+': operator.add, '-': operator.sub, '\*': operator.mul, '/': operator.truediv}

OPERATORS = {'|': log\_sum, '&': log\_mul, '~': log\_not}

stack = []

for token in expr.split(" "):

if token in OPERATORS:

op2, op1 = stack.pop(), stack.pop()

stack.append(OPERATORS[token](op1, op2))

else:

stack.append(arr(token, index, noa))

return stack.pop()

**Пример служебных файлов:**

**articles.txt**

One\_Piece

Berserk\_(manga)

Dorohedoro

Ghost\_in\_the\_Shell\_(1995\_film)

**requests.txt**

Luffy

Guts

Kaiman

Kusanagi

anime

anime & manga

**output.txt**

req: luffy --- output: ['1.txt']

req: guts --- output: ['2.txt']

req: kaiman --- output: ['3.txt']

req: kusanagi --- output: ['4.txt']

req: anime --- output: ['1.txt', '2.txt', '3.txt', '4.txt']

req: anime & manga --- output: ['1.txt', '2.txt', '3.txt', '4.txt']

База хранится в папке data в виде файлов number.txt, где number – порядковый номер файла в базе.