Nejc Povšič, Vid Ribič, Luka Bezovšek

3. seminar pri predmetu Iskanje in ekstrakcija podatkov s spleta

MENTORJA: prof. dr. Marko Bajec in doc. dr. Slavko Žitnik

1. Uvod

Za tretji seminar smo imeli nalogo ustvariti preprost inverzni indeks in implementirati iskanje. Na voljo smo imeli 1416 spletnih strani, iz katerih smo izluščili besedilo, ga sprocesirali, nato pa posamezne besede shranili v indeks. Ta indeks smo nato uporabili pri iskanju in na koncu preverili razliko med iskanjem z uporabo inverznega indeksa in s tradicionalnim iskanjem, kjer je algoritem odprl vsak dokument in iskal zadetke.

2. Postavitev delovnega okolja

Vsebino celotnega inverznega indeksa s katerim si pomagamo pri iskanju, hranimo v *sqlite* podatkovni bazi. Za pravilno delovanje iskanja, smo definirali dve tabeli. Prva je tabela *IndexWord*, ki hrani vse besede, ki se kadarkoli pojavijo v vseh dokumentih oz. se uvrščajo v nabor tistih besed, po katerih bomo iskali. Več o ustreznosti besed za izbor bomo spregovorili v naslednjih poglavjih. Tabela *IndexWord* vsebuje samo eno polje, ki se imenuje *word*. To je tipa TEXT in hrani vsako besedo posebej. Hkrati smo to polje označili tudi kot primarni ključ v tabeli, zato se vrednosti v tabeli ne ponavljajo, hkrati pa nam bo ta vrednost služila kot tuji ključ v naslednji tabeli. Druga tabela se imenuje Posting, v njej pa hranimo tiste podatke, ki nam dejansko pomagajo pri iskanju s pomočjo inverznega indeksa. Prvo polje imenovano *word,* je tuji ključ in referenca na tabelo *IndexWord.* Njegova vrednost nam pove, za katero besedo gre pri trenutni instanci zapisa. Naslednje polje se imenuje *documentName,* v njem pa je zapisano ime dokumenta v katerem se ta beseda pojavi. Za vsak dokument v katerem se pojavi specifična beseda, imamo tako v tabeli *IndexWord* en zapis, v tabeli *documentName* pa toliko zapisov, kolikor je takih dokumentov, ki vsebujejo dano besedo. Polje *frequency* je celoštevilskega tipa, njegova vrednost pa nam pove, kolikokrat se beseda *word* pojavi v dokumentu z imenom *documentName.* Zadnje polje v tej tabeli se imenuje *indexes.* V njem hranimo vse indekse pojavitev besede *word v* dokumentu z imenom *documentName.* Ker v podatkovni bazi tipa sqlite lahko hranimo samo osnovne podatkovne tipe - ne moremo hraniti npr. seznama, indekse pojavitev hranimo kot tekstovno vrednost, posamezne vrednosti pa med seboj ločimo z vejico. Če se na primer neka beseda v dokumentu pojavi na prvem, petem in desetem mestu, bomo to shranili kot *“0, 4, 9”*. Za vse operacije s podatkovno bazo v kateri hranimo inverzni indeks, smo ustvarili razred *Storage*. Pripadajoče metode razreda skrbijo za povezovanje s podatkovno bazo ter za

zapisovanje in brisanje podatkov. Na spodnji sliki je prikazan primer definicije razreda *Storage* in metoda imenovana *create_tables,* ki poskrbi za kreiranje zgoraj opisanih tabel.

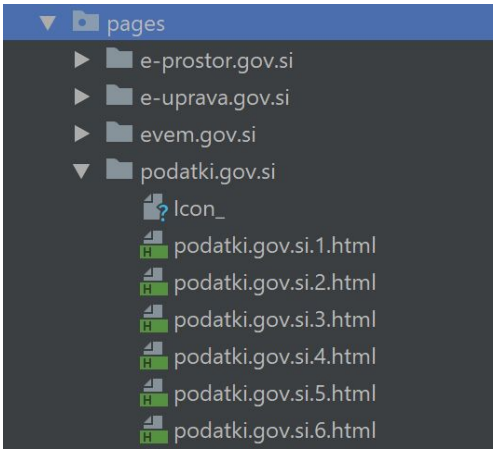
```
class Storage:
    def __init__(self, db_name):
        self.conn = sqlite3.connect(db_name)

    def create_tables(self):
        c = self.conn.cursor()

        c.execute('''
            CREATE TABLE IF NOT EXISTS IndexWord (
                word TEXT PRIMARY KEY
            );
        ''')

        c.execute('''
            CREATE TABLE IF NOT EXISTS Posting (
                word TEXT NOT NULL,
                documentName TEXT NOT NULL,
                frequency INTEGER NOT NULL,
                indexes TEXT NOT NULL,
                PRIMARY KEY (word, documentName),
                FOREIGN KEY (word) REFERENCES IndexWord(word)
            );
        ''')

        self.conn.commit()
```

Z metodo create_tables ustvarimo potrebni tabeli za delovanje indeksa

Kako smo se dejansko lotili gradnje indeksa, pa si bomo pogledali v poglavju Indeksiranje.

3. Procesiranje spletnih strani

Preden smo lahko sploh ustvarili indeks, smo morali vseh 1416 spletnih strani ustrezno procesirati. Za ta namen smo definirali razred PreProcessor. Preden si pogledamo, kako delujejo specifične metode le-tega, si poglejmo še, kako smo sploh prebrali vsebino vseh strani. Definirali smo razred Reader, ki vsebuje metodo *get_documents*. Metoda se sprehodi po celotni vsebini mape documents_folder, katere ime ji podamo pri inicializaciji in poišče vse podrejene mape v njej. Nato pregleda vsebino vsake podrejene mape, v katerih se nahajajo strani, po katerih bomo iskali v nalogi. Na spodnji sliki je prikazana struktura map in podmap, ki vsebujejo strani tipa html. Ko smo ustvarili instanco razreda Reader, smo mu za ime korenske mape tako podali vrednost ‘pages’.



Struktura hrambe spletnih strani, ki smo jih morali obdelati

Vsebino vsake strani, ki smo jo našli v opisani strukturi smo nato prebrali in shranili v objekt, ki smo ga oblikovali po naslednji shemi:

{
“name”: <ime dokumenta>
“content”: <vsebina dokumenta>
}

S tako oblikovanimi objekti, ki jih vrnemo v seznamu vseh dokumentov, si nato v prihodnosti pomagamo pri obdelavi in grajenju indeksa. Spodnja slika prikazuje implementacijo avtomatskega pridobivanja vsebine vseh datotek v opisani strukturi.

```
class Reader:
    def __init__(self, folder):
        self.documents_folder = folder

    def get_documents(self):
        documents = []
        total_documents_counter = 0

        directory = [fol for fol in os.listdir(self.documents_folder) if
                     os.path.isdir(os.path.join(self.documents_folder, fol))]
        directory = sorted(directory)

        for folder in directory:
            documents_counter = 0

            documents_sub_folder = r'%s/%s' % (self.documents_folder, folder)
            sub_folder_content = [file for file in os.listdir(documents_sub_folder) if file.endswith('.html')]

            for document in sub_folder_content:

                html_file = r'%s/%s' % (documents_sub_folder, document)

                try:
                    documents.append({"name": document, "content": open(html_file, mode='r', encoding="utf8").read()})
                    total_documents_counter += 1
                    documents_counter += 1

                except Exception as error:
                    print('An exception occured while parsing page ', document)
                    print(error)

            print('_____\n')
            print(' FETCHED ' + str(documents_counter) + ' PAGES INSIDE ' + folder + ' FOLDER \n')

        print('_____\n')
        print('FETCHED ' + str(total_documents_counter) + ' TOTAL NEW PAGES\n')

        return documents
```

Pridobivanje in hramba vsebine strani

Vrnimo se zopet k razredu *PreProcessor.* Metoda *tokenize_documents* poskrbi, da se vsebina vsakega dokumenta v seznamu po prej opisanem postopku pridobivanja ustrezno pripravi na postopek indeksiranja. Ker gre za vsebino strani tipa html, ta ob prebiranju vsebuje html značke, ki jih moramo odstraniti in ohraniti samo vsebino, ki definira pomen in ne strukturo strani. To storimo z uporabo knjižnice BeautifulSoup, ki ne samo, da odstrani html značke iz vsebine, ampak poskrbi tudi za odstranitev javascript in css odsekov, ki prav tako ne povedo nič o sami vsebini strani. Nato je potrebno vsebino strani še ustrezno tokenizirati - tekst razdeliti na seznam posameznih besed. Tovrstna oblika vsebine nam bo pri izdelavi indeksa precej olajšala delo. Primer razreda *PreProcessor* z implementiranimi metodami za predhodno obdelavo vsebine, je prikazan na spodnji sliki.

```python
from bs4 import BeautifulSoup
from nltk.tokenize import word_tokenize
from stopwords import stop_words_slovene


class PreProcessor:
    def __init__(self):
        pass

    @staticmethod
    def tokenize_documents(documents):
        processed_documents = []

        for document in documents:
            document['content'] = PreProcessor.tokenize_document(document['content'])

            processed_documents.append(document)

        return processed_documents

    @staticmethod
    def remove_markups(content):
        soup = BeautifulSoup(content, "html.parser")

        for s in soup(['script', 'style']):
            s.decompose()

        return ' '.join(soup.stripped_strings)

    @staticmethod
    def tokenize_document(content):
        clean_doc = PreProcessor.remove_markups(content)

        return word_tokenize(clean_doc)

    @staticmethod
    def remove_stopwords(tokenized_content, convert_to_lower=False):
        filtered_content = []

        for word in tokenized_content:
            if word not in stop_words_slovene:
                filtered_content.append(word if not convert_to_lower else word.lower())

        return filtered_content
```

Implementacija razreda PreProcessor za obdelavo vsebine strani

4. Indeksiranje

Ko smo vsebino dokumentov ustrezno procesirali, smo se lotili izdelave indeksa. Kot smo že omenili v prejšnjem odstavku, smo vsebino dokumentov tokenizirali (zapisali v obliki seznamov besed), oblika zapisa posamezne instance dokumenta pa ostaja enaka opisani

(ime dokumenta in vsebina dokumenta). Izdelavo indeksa smo definirali s pomočjo razreda *IndexBuilder*. Ob inicializaciji instance razreda, se ustvari nova instanca razreda *Storage,* ki poskrbi za hrambo podatkov, njegovo strukturo pa smo že opisali v poglavju Postavitev delovnega okolja. Hkrati s klicem metode *create_tables* le-ta poskrbi za inicializacijo podatkovnih tabel in izbris starih podatkov, če ti že obstajajo, saj indeks vsakič zgradimo od začetka.

```
from Storage import Storage
from stopwords import stop_words_slovene
import numpy as np

DB_NAME = 'inverted-index.db'


class IndexBuilder:
    def __init__(self, documents):
        # create new db instance
        self.storage = Storage(DB_NAME)

        # create the tables if they do not exist
        self.storage.create_tables()

        self.documents = documents

        # create index from processed documents
        self.build_index()
```

Inicializacija razreda IndexBuilder

Ko je podatkovna baza pripravljena, lahko pričnemo z vpisovanjem podatkov. Pokličemo metodo *build_index*, ki smo jo implementirali tako, da se sprehodi po seznamu vseh predhodno obdelanih dokumentov in vsako besedo, ki ustreza določenim kriterijem uvrsti v indeks. Najprej celotno vsebino dokumenta pretvorimo v male črke. Zakaj tega nismo storili že v procesu predobdelave? Predobdelano vsebino dokumentov ne uporabljamo samo v procesu izdelave indeksa, ampak tudi v procesu iskanja. Ker moramo ob prikazu zadetkov iskanja prikazati originalno vsebino dokumenta, po procesu predobdelave še vedno hranimo originalne velikosti posameznih znakov v besedah. Iz istega razloga v seznamih besed hranimo tudi posebne znake, ločila in ‘nepomembne besede’ (ang. stopwords). Po pretvorbi besed se nato sprehodimo po celotnem seznamu vsebine dokumenta in za vse besede, ki ne spadajo med ‘nepomembne besede’, zapišemo v podatkovno bazo.

Seznam ‘nepomembnih besed’ smo uporabili predefiniran seznam slovenskih besed iz knjižnice nltk.corpus, ki smo ga razširili še z nekaterimi drugimi slovenskimi besedami, med katere smo uvrstili tudi ločila.

```
# -*- coding: UTF-8 -*-
import nltk
from nltk.corpus import stopwords

# download stopwords in case not present localy
nltk.download('stopwords')
nltk.download('punkt')

stop_words_slovene = set(stopwords.words('slovenian')).union(
    {'ter', 'nov', 'novo', 'nova', 'zato', 'ĹAe', 'zaradi', 'a', 'ali', 'april', 'avgust', 'b', 'bi', 'bil', 'bila',
     'bile', 'bili', 'bilo', 'biti', 'blizu', 'bo', 'bodo', 'bojo', 'boli', 'bom', 'bomo', 'boste', 'bova', 'boĹA',
     'brez', 'c', 'cel', 'cela', 'celi', 'celo', 'd', 'da', 'daleÄ', 'dan', 'danes', 'datum', 'december', 'deset',
     'deseta', 'deseti', 'deseto', 'devet', 'deveta', 'deveti', 'deveto', 'do', 'dober', 'dobra', 'dobri', 'dobro',
     'dokler', 'dol', 'dolg', 'dolga', 'dolgi', 'dovolj', 'drug', 'druga', 'drugi', 'drugo', 'dva', 'dve', 'e', 'eden',
     'en', 'ena', 'ene', 'eni', 'enkrat', 'eno', 'etc.', 'f', 'februar', 'g', 'g.', 'ga', 'ga.', 'gor', 'gospa',
     'gospod', 'h', 'halo', 'i', 'idr.', 'ii', 'iii', 'in', 'iv', 'ix', 'iz', 'j', 'januar', 'jaz', 'je', 'ji', 'jih',
     'jim', 'jo', 'julij', 'junij', 'jutri', 'k', 'kadarkoli', 'kaj', 'kajti', 'kako', 'kakor', 'kamor', 'kamorkoli',
     'kar', 'karkoli', 'katerikoli', 'kdaj', 'kdo', 'kdorkoli', 'ker', 'ki', 'kje', 'kjer', 'kjerkoli', 'ko', 'koder',
     'koderkoli', 'koga', 'komu', 'kot', 'kratek', 'kratka', 'kratke', 'kratki', 'l', 'lahka', 'lahke', 'lahki',
     'lahko', 'le', 'lep', 'lepa', 'lepe', 'lepi', 'lepo', 'leto', 'm', 'maj', 'majhen', 'majhna', 'majhni', 'malce',
     'malo', 'manj', 'marec', 'me', 'med', 'medtem', 'mene', 'mesec', 'mi', 'midva', 'midve', 'mnogo', 'moj', 'moja',
     'moje', 'mora', 'morajo', 'moram', 'moramo', 'morate', 'moraĹA', 'morem', 'mu', 'n', 'na', 'nad', 'naj', 'najina',
     'najino', 'najmanj', 'naju', 'najveÄ', 'nam', 'narobe', 'nas', 'nato', 'nazaj', 'naĹA', 'naĹAa', 'naĹAe', 'ne',
     'nedavno', 'nedelja', 'nek', 'neka', 'nekaj', 'nekatere', 'nekateri', 'nekatero', 'nekdo', 'neke', 'nekega',
     'neki', 'nekje', 'neko', 'nekoga', 'nekoÄ', 'ni', 'nikamor', 'nikdar', 'nikjer', 'nikoli', 'niÄ', 'nje', 'njega',
     'njegov', 'njegova', 'njegovo', 'njej', 'njemu', 'njen', 'njena', 'njeno', 'nji', 'njih', 'njihov', 'njihova',
     'njihovo', 'njiju', 'njim', 'njo', 'njun', 'njuna', 'njuno', 'no', 'nocoj', 'november', 'npr.', 'o', 'ob', 'oba',
     'obe', 'oboje', 'od', 'odprt', 'odprta', 'odprti', 'okoli', 'oktober', 'on', 'onadva', 'one', 'oni', 'onidve',
     'osem', 'osma', 'osmi', 'osmo', 'oz.', 'p', 'pa', 'pet', 'peta', 'petek', 'peti', 'peto', 'po', 'pod', 'pogosto',
     'poleg', 'poln', 'polna', 'polni', 'polno', 'ponavadi', 'ponedeljek', 'ponovno', 'potem', 'povsod', 'pozdravljen',
     'pozdravljeni', 'prav', 'prava', 'prave', 'pravi', 'pravo', 'prazen', 'prazna', 'prazno', 'prbl.', 'precej',
     'pred', 'prej', 'preko', 'pri', 'pribl.', 'približno', 'primer', 'pripravljen', 'pripravljena', 'pripravljeni',
     'proti', 'prva', 'prvi', 'prvo', 'r', 'ravno', 'redko', 'res', 'reÄ', 's', 'saj', 'sam', 'sama', 'same', 'sami',
     'samo', 'se', 'sebe', 'sebi', 'sedaj', 'sedem', 'sedma', 'sedmi', 'sedmo', 'sem', 'september', 'seveda', 'si',
     'sicer', 'skoraj', 'skozi', 'slab', 'smo', 'so', 'sobota', 'spet', 'sreda', 'srednja', 'srednji', 'sta', 'ste',
     'stran', 'stvar', 'sva', 't', 'ta', 'tak', 'taka', 'take', 'taki', 'tako', 'takoj', 'tam', 'te', 'tebe', 'tebi',
     'tega', 'teĺžak', 'teĺžka', 'teĺžki', 'teĺžko', 'ti', 'tista', 'tiste', 'tisti', 'tisto', 'tj.', 'tja', 'to',
     'toda', 'torek', 'tretja', 'tretje', 'tretji', 'tri', 'tu', 'tudi', 'tukaj', 'tvoj', 'tvoja', 'tvoje', 'u', 'v',
     'vaju', 'vam', 'vas', 'vaĹA', 'vaĹAa', 'vaĹAe', 've', 'vedno', 'velik', 'velika', 'veliki', 'veliko', 'vendar',
     'ves', 'veÄ', 'vi', 'vidva', 'vii', 'viii', 'visok', 'visoka', 'visoke', 'visoki', 'vsa', 'vsaj', 'vsak', 'vsaka',
     'vsakdo', 'vsake', 'vsaki', 'vsakomur', 'vse', 'vsega', 'vsi', 'vso', 'vÄasih', 'vÄeraj', 'x', 'z', 'za',
     'zadaj', 'zadnji', 'zakaj', 'zaprta', 'zaprti', 'zaprto', 'zdaj', 'zelo', 'zunaj', 'Ä', 'Äe', 'Ästo',
     'Ästrta', 'Ästrtek', 'Ästrti', 'Ästrto', 'Äez', 'Äigav', 'ĹA', 'ĹAest', 'ĹAesta', 'ĹAesti', 'ĹAesto',
     'ĹAtiri', 'Íž', 'Íže', 'svoj', 'jesti', 'imeti', '\u0161e', 'iti', 'kak', 'www', 'km', 'eur', 'paÄ', 'del',
     'kljub', 'ĹAele', 'prek', 'preko', 'znova', 'morda', 'kateri', 'katero', 'katera', 'ampak', 'lahek', 'lahka',
     'lahko', 'morati', 'torej', '.', ',', ';', ':', '-', '_', '*', '+', '?', '!', '"', '#', '=', '/', '$', '%', '|',
     '@', '', '&', '', '(', ')', '--', '~', '•', '[', ']', '{', '}', '\'', '\'\'', '>', '<', '©', '~', '€', 'f'])
```

Razširitev seznama nepomembnih besed

Ker želimo zapisovanje indeksa pojavitve besede za vsak dokument minimizirati, ob prvi pojavitvi besede v dokumentu poiščemo tudi vse ostale pojavitve besede v tem dokumentu. Tako lahko iz števila najdenih besed določimo frekvenco besede v dokumentu, hkrati pa zgradimo tekstovni niz, ki podaja indekse pojavitve besede v dokumentu.

```
def build_index(self):
    for document in self.documents:

        # transform all words in document to lowercase before indexing
        lowercase_content = [x.lower() for x in document['content']]

        for word in lowercase_content:
            if word not in stop_words_slovene:
                # find all occurences of the word in the document
                occurrence_index = np.where(np.array(lowercase_content) == word)[0]

                frequency = len(occurrence_index)

                self.storage.insert_index_word(word, document['name'], frequency,
                                               self.indexes_to_string(occurrence_index))

    self.storage.close_connection()

# transform array of indexes to comma separated string
def indexes_to_string(self, indexes):
    index_string = ''

    for index in np.nditer(indexes):
        index_string += '{},'.format(str(index))

    # remove last comma character
    index_string = index_string[:-1]

    return index_string
```

Implementacija metode za uvrščanje besed v indeks in grajenje tekstovnega niza pojavitve besede v dokumentu

Zapis posamezne besede v podatkovno bazo smo implementirali s pomočjo enega SQL ukaza. Ta vstavi novo besedo v tabelo *IndexWord* samo v primeru, če ta beseda v tabeli še ne obstaja.

```
def insert_index_word(self, word, document_name, frequency, occurence_indexes):
    c = self.conn.cursor()

    c.execute('''
        INSERT INTO IndexWord (word)
        SELECT ?
        WHERE NOT EXISTS(SELECT 1 FROM IndexWord WHERE word = ?);
    ''', (word, word))

    self.insert_posting(word, document_name, frequency, occurence_indexes)

    self.conn.commit()
    # self.conn.close()
```

Implementacija zapisovanja nove besede v tabelo IndexWord

Na enak način smo implementirali tudi zapisovanje v tabelo *Posting.* Ker se frekvenca in indeksi pojavitve besede v dokumentu izračunajo že ob prvi pojavitvi besede v dokumentu, z enim SQL stavkom poskrbimo, da zapis ustreza vsem pojavitvam besede v dokumentu. Kriterij za zapis nove instance v tabelo je v tem primeru še neobstoječi zapis z vrednostima *word* in *documentName.*

```
def insert_posting(self, word, document_name, frequency, indexes):
    c = self.conn.cursor()

    c.execute('''
        INSERT INTO Posting (word, documentName, frequency, indexes)
        SELECT ?, ?, ?, ?
        WHERE NOT EXISTS(SELECT 1 FROM Posting WHERE word = ? AND documentName = ?);
    ''', (word, document_name, frequency, indexes, word, document_name))

    self.conn.commit()
    # self.conn.close()
```

Implementacija zapisovanja v tabelo Posting

Velikost podatkovne baze inverznega indeksa, ki smo ga zgradili za vse dokumente po opisanih metodah, je na koncu znašala pribl. 39MB, pri čemer je število besed v tabeli *IndexWord* doseglo število 47650, tabela *Posting* pa vsebuje 396063 zapisov.

5. Iskanje z uporabo inverznega indeksa

Ko smo imeli zgrajen indeks, smo lahko začeli z iskanjem. Ustvarili smo razred *DataRetrieval*, ki vsebuje celotno logiko iskanja, združevanja najdenih rezultatov in izpisovanja.

Za iskanje smo že imeli določene tri nize, sami pa smo dodali še tri, ki so vidni na spodnji sliki.

```
search_queries = [
    "predelovalne dejavnosti",
    "trgovina",
    "social services",
    "Sistem SPOT",
    "EU Parlament",
    "VLADA REPUBLIKE SLOVENIJE"
]
```

Iskalni nizi, ki smo jih uporabili

Preden sploh lahko začnemo z iskanjem, moramo niz pretvoriti v majhne črke. Če je niz sestavljen iz večih besed, ga razbijemo na posamezne besede in nato nad njimi izvajamo iskanje, kot da bi iskali samo eno besedo.

Za vsako besedo, ki smo jo izluščili iz niza, najdemo vse ustrezne vnose iz tabele “Postings” in jih združimo. Tako dobimo vse možne vnose za besede iskalnega niza. Vnosi vsebujejo stran, na kateri se ta beseda nahaja, pozicijo pojavitev in frekvenco.

```
def search(self, query, max_results_to_show=None):
    start_time = datetime.now()

    query_words = query.lower().split(" ")

    # includes all the postings for all the words in the query
    postings = []

    for word in query_words:
        # finds all the postings for the word and concatenates the two arrays
        postings += self.storage.find_word_postings(word)

    merged_results = self.merge_postings(postings)

    if max_results_to_show is not None:
        merged_results = merged_results[0:max_results_to_show]

    self.get_snippets(merged_results)

    end_time = datetime.now()

    time_difference = end_time - start_time

    self.print_output(query, time_difference, merged_results)
```

Metoda search, kjer lahko vidimo delovanje iskanja z uporabo indeksa

Celoten seznam, ki smo ga dobili za iskalni niz, nato razvrstimo po straneh z uporabo slovarja. V slovar si shranimo vse *postinge* (dolžina seznama je enaka številu besed v iskalnem nizu), naslov strani in skupno frekvenco za vse besede iskalnega niza, dodamo pa tudi polje *snippets*, v katerega bomo shranili kratke izseke besedila iz strani:

{
“postings”: <seznam pojavitev za vsako besedo>,
“site”: <stran, kateri postingi pripadajo>,
“total_frequencies”: <število pojavitev vseh besed v iskalnem nizu>,
“snippets”: <izseki iz izvornega besedila>
}

Slovar nato pretvorimo nazaj v seznam, da lahko strani uredimo po padajočem številu skupnih frekvenc.

```python
def merge_postings(self, postings):
    postings_per_site = {}

    # group all the postings by site
    for posting in postings:
        site = posting[1]
        frequency = posting[2]

        if site in postings_per_site:
            postings_per_site[site]["postings"].append(posting)
            postings_per_site[site]["total_frequencies"] += frequency
        else:
            postings_per_site[site] = {
                "postings": [posting],
                "site": site,
                "snippets": None,
                "total_frequencies": frequency
            }

    list_to_sort = []

    for site in postings_per_site:
        list_to_sort.append(postings_per_site[site])

    # sort the postings that are grouped by site by the total frequencies of words
    sorted_list = sorted(list_to_sort, key=lambda d: d["total_frequencies"], reverse=True)

    return sorted_list
```

Metoda merge_postings, ki združi postinge glede na stran in jih na koncu uredi po velikosti števila pojavitev

Preostane nam še to, da prikažemo izseke iz izvornega besedila (ang. “snippets”). To storimo tako, da iz vseh *postingov* pridobimo pozicije besed, jih uredimo po velikosti, nato pa jih primerjamo z razčlenjeno (tokenizirano) stranjo, ki smo jo, kot smo omenili, shranili na disk. Stran preberemo vrstico po vrstico in besedimo shranimo v seznam. Izsek smo omejili na tri besede pred in po najdeni besedi.Seveda tu pride do problema, ker v razčlenjeni strani ne hranimo presledkov, zato ne moremo točno vedeti, kje so bili presledki postavljeni. Trenutno presledek izpišemo vsepovsod razen pred piko in vejico, kar ni popolno, a že zelo polepša izgled.

Za lažji pregled smo število strani, za katere prikazujemo rezultate, omejili na pet. Na spodnji sliki vidimo primer izpisa za iskalni niz “predelovalne dejavnosti”.

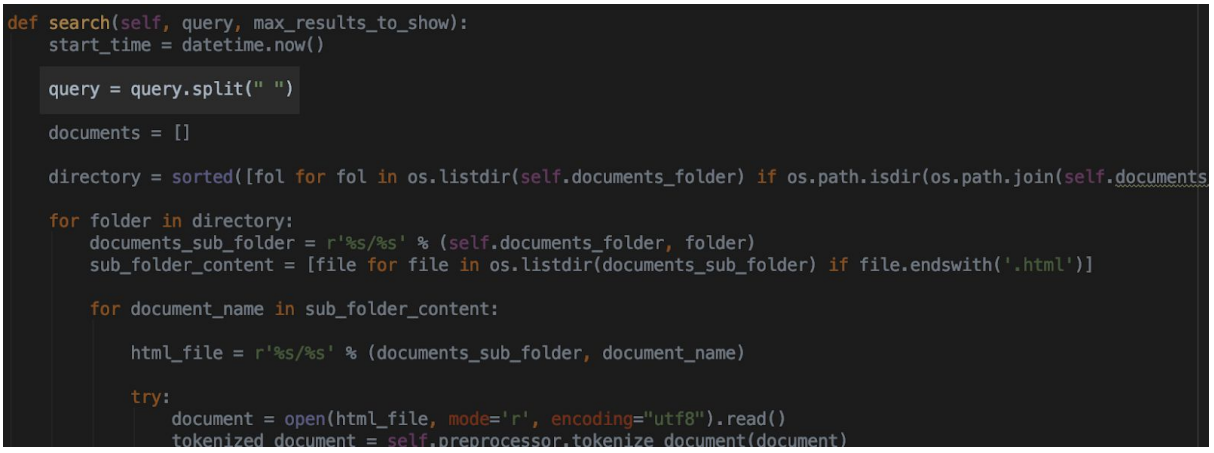
```
[SEARCH WITH INVERTED INDEX] Results for a query: "predelovalne dejavnosti"

    Results found in 76ms

    Frequencies    Document                              Snippet
    -------------- ------------------------------------- ------------------------------------------------------------
    1273           evem.gov.si.371.html                  iskanje ustrezne šifre dejavnosti /storitve in ...  pogojih za opravljanje dejavnosti. V
    75             evem.gov.si.377.html                  Defektolog v zdravstveni dejavnosti Dekan oziroma ...  Dietetik v zdravstveni dejavnosti
    35             evem.gov.si.452.html                  Druge storitvene dejavnosti, drugje ...  › Druge storitvene dejavnosti, drugje ...  ) Drug
    31             evem.gov.si.653.html                  Dovoljenje za opravljanje dejavnosti specializirane prodajalne ...  radijske ali televizi
    29             evem.gov.si.398.html                  usmerjene na opravljanje dejavnosti ( npr ...  za namene opravljanja dejavnosti ipd. ...
```

Primer izpisa za niz “predelovalne dejavnosti”

6. Iskanje brez uporabe indeksa

Iskanje brez uporabe indeksa smo implementirali v razredu *ClassicSearch*. V inicializacijski metodi smo povedali, v katerem direktoriju se nahajajo html strani za iskanje. Iskanje smo pognali s klicem metode *search*, ki vzame dva parametra: iskalni niz *(query)* in število prikazanih zadetkov *(max_results_to_show)*.

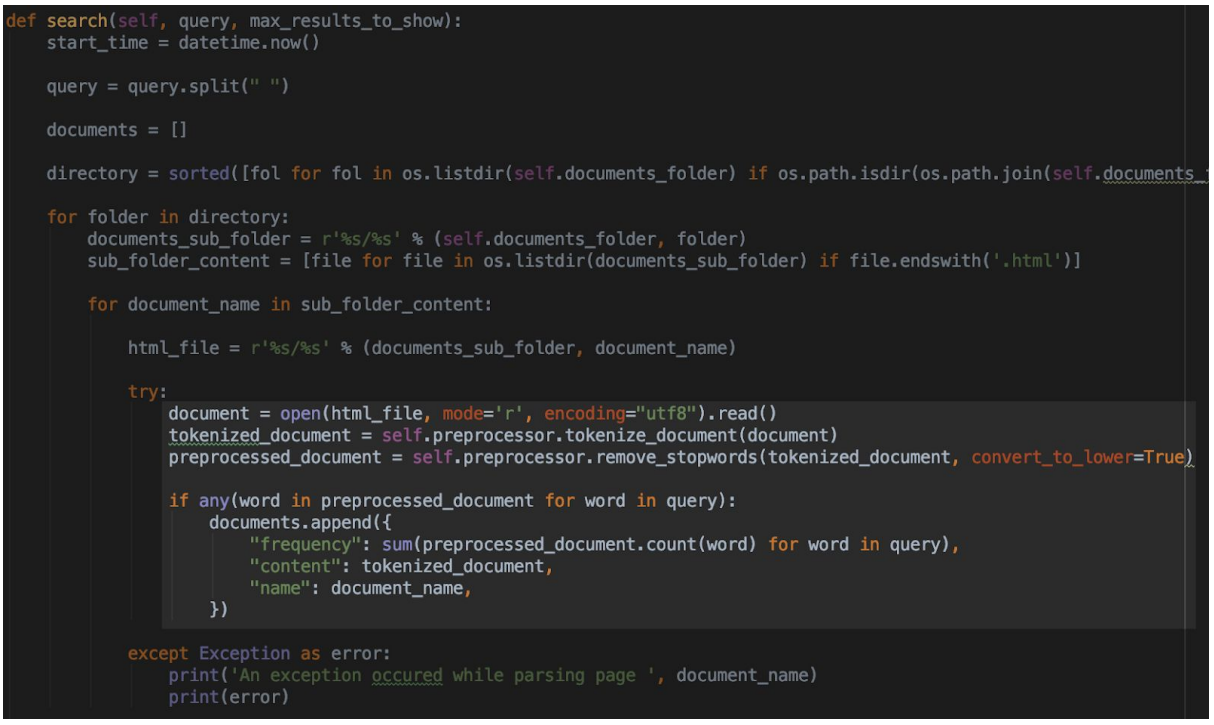
Metoda search najprej razbije query s funkcijo *split* v seznam, tako da lahko iščemo tako z eno kot večimi besedami.



```
def search(self, query, max_results_to_show):
    start_time = datetime.now()

    query = query.split(" ")

    documents = []

    directory = sorted([fol for fol in os.listdir(self.documents_folder) if os.path.isdir(os.path.join(self.documents

    for folder in directory:
        documents_sub_folder = r'%s/%s' % (self.documents_folder, folder)
        sub_folder_content = [file for file in os.listdir(documents_sub_folder) if file.endswith('.html')]

        for document_name in sub_folder_content:

            html_file = r'%s/%s' % (documents_sub_folder, document_name)

            try:
                document = open(html_file, mode='r', encoding="utf8").read()
                tokenized_document = self.preprocessor.tokenize_document(document)
```

Iskalni niz smo najprej razbili v seznam besed.

Nato se sprehodi čez podmape glavnega direktorija in v vsaki podmapi odpre dokument. S pomočjo predprocesorja ga tokenizira, tako pridobljen seznam besed pa potem dodatno očisti -- odstrani nepotrebne besede in velike začetnice zamenja z malimi. Dokument je sedaj pripravljen za iskanje: s pythonsko funkcijo *any* pogledamo, če se katerakoli izmed besed v iskalnem nizu nahaja v dokumentu. V primeru, da se, v ločen seznam dokumentov zabeležimo podatek o frekvenci, vsebini in imenu dokumenta.



```
def search(self, query, max_results_to_show):
    start_time = datetime.now()

    query = query.split(" ")

    documents = []

    directory = sorted([fol for fol in os.listdir(self.documents_folder) if os.path.isdir(os.path.join(self.documents_

    for folder in directory:
        documents_sub_folder = r'%s/%s' % (self.documents_folder, folder)
        sub_folder_content = [file for file in os.listdir(documents_sub_folder) if file.endswith('.html')]

        for document_name in sub_folder_content:

            html_file = r'%s/%s' % (documents_sub_folder, document_name)

            try:
                document = open(html_file, mode='r', encoding="utf8").read()
                tokenized_document = self.preprocessor.tokenize_document(document)
                preprocessed_document = self.preprocessor.remove_stopwords(tokenized_document, convert_to_lower=True)

                if any(word in preprocessed_document for word in query):
                    documents.append({
                        "frequency": sum(preprocessed_document.count(word) for word in query),
                        "content": tokenized_document,
                        "name": document_name,
                    })

            except Exception as error:
                print('An exception occured while parsing page ', document_name)
                print(error)
```

Vsak dokument smo odprli, tokenizirali, odstranili nepotrebne besede in spremenili velike v male začetnice. Če je vseboval iskalni niz, smo ga zapisali v ločen seznam dokumentov.

Ko imamo seznam najdenih dokumentov, pripravimo rezultate. Dokumente sortiramo po frekvenci (za sortiranje smo uporabili pythonsko metodo *sorted*) in se sprehodimo čez toliko dokumentov, kot jih moramo prikazati (upoštevamo parameter *max_results_to_show*). Za vsak dokument ustvarimo izvleček (snippet) v ločeni metodi, ki pogleda, kje se nahaja iskani niz in v snippet zabeleži 3 predhodnje in 3 naslednje besede.

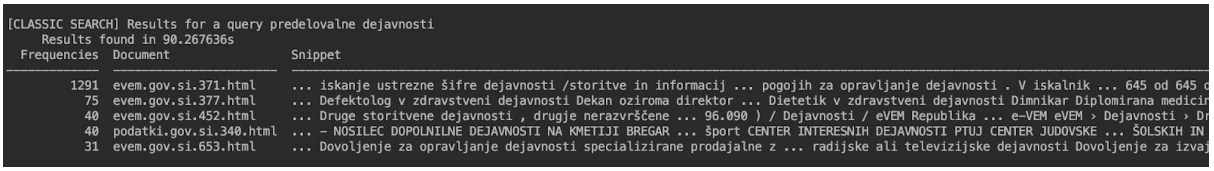
```
def search(self, query, max_results_to_show):
    start_time = datetime.now()

    query = query.split(" ")

    documents = []

    directory = sorted([fol for fol in os.listdir(self.documents_folder) if os.path.isdir(os.path.join(self.documents

    for folder in directory:
        documents_sub_folder = r'%s/%s' % (self.documents_folder, folder)
        sub_folder_content = [file for file in os.listdir(documents_sub_folder) if file.endswith('.html')]

        for document_name in sub_folder_content:

            html_file = r'%s/%s' % (documents_sub_folder, document_name)

            try:
                document = open(html_file, mode='r', encoding="utf8").read()
                tokenized_document = self.preprocessor.tokenize_document(document)
                preprocessed_document = self.preprocessor.remove_stopwords(tokenized_document, convert_to_lower=True)

                if any(word in preprocessed_document for word in query):
                    documents.append({
                        "frequency": sum(preprocessed_document.count(word) for word in query),
                        "content": tokenized_document,
                        "name": document_name,
                    })

            except Exception as error:
                print('An exception occured while parsing page ', document_name)
                print(error)

    results = []

    for document in sorted(documents, key=lambda k: k["frequency"], reverse=True)[:max_results_to_show]:
        results.append([document["frequency"], document["name"], self.create_snippet(query, document["content"])])

    end_time = datetime.now()
```

Rezultate smo sortirali in shranili v seznam results

Ko imamo rezultate pripravljene, ustavimo štoparico in jih s pomočjo zunanje knjižnice *tabulate* (več o njej v rubriki *Uporabljene knjižnice*) prikažemo.

```
    results = []

    for document in sorted(documents, key=lambda k: k["frequency"], reverse=True)[:max_results_to_show]:
        results.append([document["frequency"], document["name"], self.create_snippet(query, document["content"])])

    end_time = datetime.now()

    time_difference = end_time - start_time

    print("[CLASSIC SEARCH] Results for a query " + " ".join(query))

    if len(documents) <= 0:
        print("     The search took {}s".format(time_difference.total_seconds()))
        print("     No results found :(\n")
    else:
        print("     Results found in {}s".format(time_difference.total_seconds()))
        print(tabulate(results, headers=["Frequencies", "Document", "Snippet"]))

    print("\n\n")
```

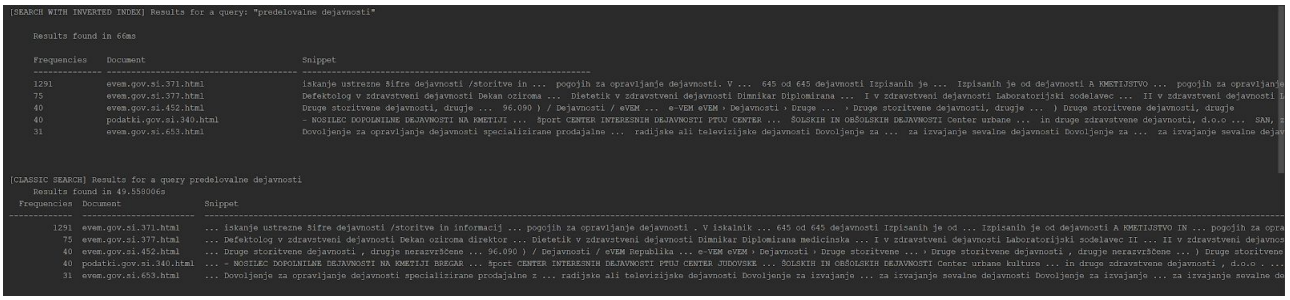
Z zunanjo knjižnico *Tabulate* smo prikazali dobljene rezultate v tabelarični obliki



Primer izpisa za iskalni niz “predelovalne dejavnosti”

7. Primerjava

Iskanje z inverznim indeksom je neprimerljivo hitrejše od klasičnega iskanja. Na priloženi sliki lahko vidimo, da je iskanje z uporabo indeksa za iskalni niz “**predelovalne dejavnosti**” potrebovalo približno 66 milisekund, medtem ko je klasično iskanje potrebovalo okoli 49 sekund, kar je več kot 740-krat več.



8. Uporabljene knjižnice

Za našo rešitev smo potrebovali nekatere zunanje knjižnice, s katerimi smo si zelo olajšali delo.

1) **NLTK**
   NLTK je pythonsko ogrodje za naravni jezik. S pomočjo korpusa iz knjižnice NLTK smo definirali nepotrebne besede v slovenščini (angl. stopwords) in dodali nekaj svojih primerov. Uporabili smo tudi modul *punkt* za zaznavanje ločil. Ključen paket tega ogrodja, ki smo ga uporabili, pa je *tokenize*, s pomočjo katerega smo predprocesirali dokumente. Uporabili smo metodo *word_tokenize*, ki tokenizira nize besed (stringe).

2) **NumPy**
   NumPy je knjižnica, ki doda podporo za zelo velike sezname in matrike z več dimenzijami, poleg pa ogromno matematičnih operacij za delo s temi matrikami. Knjižnico smo uporabili pri iskanju indeksov posameznih besed v dokumentu, saj iteracije z uporabo le-te porabijo manj časa kot navadne for zanke, vsebina dokumentov pa je v nekaterih primerih lahko zelo velika. Primer uporabe - iskanje vseh indeksov pojavitev besede v dokumentu:
   `occurrence_index = np.where(np.array(lowercase_content) == word)[0]`

3) **Tabulate**
   Pythonska knjižnica za lepši izpis tabelaričnih podatkov. Uporaba je preprosta: kot parameter metode *tabulate* podamo podatke in nazive za glavo tabele. Primer: `tabulate(search_results, headers=["Frequencies", "Document", "Snippet"])`.

4) **Beautiful Soup**
   Knjižnico Beautiful Soup smo spoznali že v prejšnjih nalogah. Tudi tokrat smo si z njo pomagali pri čiščenju besedila. Iz HTML dokumentov smo z njeno pomočjo odstranili *script* in *style* značke.

9. Zaključek

S tretjo nalogo smo zaključili, kar smo začeli s spletnim pajkom in nadaljevali z ekstrakcijo strukturiranih podatkov iz spletnih strani. Implementirali smo metodo za iskanje po ključnih besedah in s preprostim eksperimentom preverili razliko v hitrosti, ki nam jo tak pristop omogoči.

10. Viri

1. Predavanja IEPS o inverznem indeksu (potrebuje prijavo v FRI spletno učilnico): https://ucilnica.fri.uni-lj.si/pluginfile.php/98677/mod_label/intro/Inverted%20index.pdf?time=1558543731938
2. Navodila za tretjo nalogo: http://zitnik.si/teaching/wier/PA3.html
3. Paket NLTK tokenize: https://www.nltk.org/api/nltk.tokenize.html
4. Tabulate: https://pypi.org/project/tabulate/
5. BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/