

Μάθημα: Ανάκτηση Πληροφορίας

Τελική εργαστηριακή άσκηση

Ονοματεπώνυμο: Χρήστος Μπίτας

**Ερώτηση 1:** Περιγράψτε συνοπτικά τι είναι η Lucene και τι μπορεί να κάνει.

Η Lucene είναι μία μεγάλης αποδοτικότητας βιβλιοθήκη ανάκτησης πληροφοριών (IR) για μηχανές αναζήτησης. Η ανάκτηση πληροφορίας (IR) αφορά την διαδικασία αναζήτησης εγγράφων, πληροφοριών στο περιεχόμενο των εγγράφων. Η Lucene δίνει δυνατότητες προσθήκης αναζήτησης για την εφαρμογή. Είναι ανοικτός κώδικας Java ο οποίος είναι δημιούργημα του Apache Software Foundation και αποτελεί την πιο δημοφιλή βιβλιοθήκη. Η συγκεκριμένη βιβλιοθήκη χρησιμοποιείται σε πολλές εφαρμογές κάποιες από τις οποίες είναι το Netflix και το LinkedIn.

Η Lucene είναι μια βιβλιοθήκη η οποία έχει σκοπό την ευρετηρίαση κειμένου και την αναζήτηση σε αυτό. Στην ουσία δίνει δυνατότητες αναζήτησης στην εφαρμογή, ευρετηριάζει και μετατρέπει τα δεδομένα σε αναζητήσιμα έτσι ώστε να μπορεί να γίνει εξαγωγή κειμένου. Αγνοεί την προέλευση των δεδομένων, τον τύπο τους αλλά και τη γλώσσα που έχουν γραφτεί καθώς το μόνο που την απασχολεί είναι η παραγωγή κειμένου από αυτά. Έτσι, μπορούμε να ευρετηριάσουμε και να αναζητήσουμε πλήθος δεδομένων από οποιοδήποτε μορφής αρχείου που μπορεί να γίνει εξαγωγή πληροφορίας κειμένου όπως για παράδειγμα τα PDF αρχεία ή ακόμη και σε κάποια βάση δεδομένων.

**Ερώτηση 2:** Αναφέρατε ποια είναι τα στοιχεία-συνιστώσες που απαρτίζουν μία εφαρμογή αναζήτησης και ποια από αυτά μπορεί να υλοποιήσει η Lucene. Περιγράψτε συνοπτικά καθένα από τα στοιχεία-συνιστώσες που υλοποιεί η Lucene.

Σελίδα 48

Οι εφαρμογές αναζήτησης αποτελούνται από τα εξής συστατικά:

1. Στοιχεία για ευρετηρίαση (components for indexing): Σκοπός του συγκεκριμένου στοιχείου είναι λήψη και η μετατροπή μεγάλου πλήθους δεδομένων σε μορφή επιτρεπτή για αναζήτηση με αποτέλεσμα την δημιουργία ενός ευρετηρίου. Ωστόσο, η ευρετηρίαση ακολουθεί κάποια βήματα τα οποία θα αναφερθούν ονομαστικά παρακάτω. Αρχικά γίνεται απόκτηση του περιεχομένου (acquire content) έπειτα η κατασκευή του εγγράφου (build document) ακολουθεί η ανάλυση του εγγράφου (analyze document) και τελευταίο βήμα είναι η ευρετηρίαση του εγγράφου (index document).
2. Στοιχεία για αναζήτηση (components for searching): Η λειτουργία του στοιχείου της αναζήτησης δίνει την δυνατότητα σε ένα χρήστη να υποβάλει κάποιο ερώτημα αναζήτησης σε έναν ευρετήριο ώστε να επιστραφεί κάποιο σχετικό έγγραφο/-α ή δεδομένο/-α. Τα συστατικά τα οποία απαρτίζουν μία μηχανή αναζήτησης είναι η διεπαφή χρήστη αναζήτησης (search user interface), έπειτα είναι η κατασκευή ερωτήματος (build query), ακολουθεί η ερώτηση αναζήτησης (search query) και τελευταία τα αποτελέσματα απόδοσης (render results).

Στην υπόλοιπη εφαρμογή αναζήτησης παρατηρούμε σε πρώτη φάση την διεπαφή της διαχείρισης (administration interface), την διεπαφή για τα αναλυτικά στοιχεία (analytics interface) και την κλιμάκωση (scaling).

Όσον αφορά τα παραπάνω στοιχεία-συνιστώσες που αναφέρθηκαν η Lucene μπορεί να υλοποιήσει κάποια αυτά τα οποία θα περιγράφουν στην συνέχεια. Για την Lucene το ευρετήριο είναι μία δομή δεδομένων που βρίσκεται στο σύστημα των αρχείων σαν ένα σύνολο αρχείων ευρετηρίου.

Από τα στοιχεία της ευρετηρίασης (components for indexing) η Lucene μπορεί να υλοποιήσει την ανάλυση του εγγράφου (analyze document) καθώς πριν γίνει ευρετηρίαση του κειμένου απαιτείται η διάσπαση των δεδομένων σε ξεχωριστά στοιχεία τα λεγόμενα ως tokens κάτι το οποίο γίνεται στην διαδικασία της ανάλυσης του εγγράφου παρέχοντας ενσωματωμένους αναλυτές. Στην συνέχεια υλοποιείται η ευρετηρίαση του εγγράφου (index document) και το έγγραφο εισάγεται στο ευρετήριο όπου η Lucene παρέχει μέσα τα οποία βοηθούν στην συγκεκριμένη διαδικασία χάρες το απλό API της.

Από τα στοιχεία της αναζήτησης (components for searching) η Lucene μπορεί να υλοποιήσει την κατασκευή ερωτήματος (build query) χρησιμοποιώντας έναν QueryParser καθώς όταν ο χρήστης πραγματοποιεί μία αναζήτηση σε οποιαδήποτε πλατφόρμα τότε πρέπει να γίνει μετάφραση του συγκεκριμένου αιτήματος προκειμένου να μπορεί να αναγνωριστεί από την συγκεκριμένη μηχανή αναζήτησης. Για την ερώτηση αναζήτησης (search query) η Lucene δίνει την δυνατότητα πλήθους επεκτάσεων συνδυάζοντας τον διανυσματικό χώρο με τα μοντέλα Boolean δίνοντας τον απόλυτο έλεγχο στον χρήστη κατά την αναζήτησή του.

Τέλος, η Lucene για την επαφή της διαχείρισης (administration interface) δίνει την δυνατότητα πολλών επιλογών ρύθμισης των παραμέτρων.

**Ερώτηση 3:** Τρέξτε την Indexer και περιγράψτε πως γίνεται το indexing παρουσιάζοντας εικόνα με τα indexed αρχεία.

Αρχικά έγινε εισαγωγή του έτοιμου Ant project στο java IDE Eclipse. Στην συνέχεια έγινε εκτέλεση της Indexer και τα αποτελέσματά της φαίνονται στο παρακάτω screenshot.

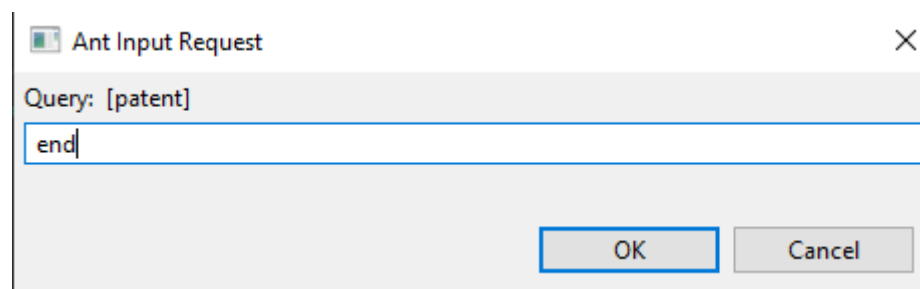
```
[echo] Running lia.meetlucene.indexer...
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\apache1.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\apache1.1.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\apache2.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\cpl1.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\cpl1.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\Freeds.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\gpl1.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\gpl2.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\gpl2.1.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\gpl2.1.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\lgpl3.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\lgpl3.0.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\mit.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\mozilla1.1.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\mozilla_eula_firefox3.txt
[java] Indexing E:\Files\0000045\000076\00000000\00000000\00000000\00000000\00000000\00000000\lia2\src\lia\meetlucene\data\mozilla_eula_thunderbird2.txt
[java] Indexing 16 files took 757 milliseconds
[echo] SUCCESSFUL
BUILD SUCCESSFUL
Total time: 26 seconds
```

### Screenshot αποτελέσματος εκτέλεσης Indexer

Παρατηρώντας τα παραπάνω αποτελέσματα καταλαβαίνουμε ότι η διαδικασία της ευρετηρίασης (indexing) προσπαθεί να κάνει προσθήκη των εγγράφων στο ευρετήριο προκειμένου να γίνεται μία πιο γρήγορη αναζήτηση με σχετικά αποτελέσματα υψηλής ποιότητας. Η Lucene δημιουργεί ένα ανεστραμμένο δηλαδή αντίστροφο ευρετήριο. Έτσι κατά την εκτέλεση ενός ερωτήματος αναζήτησης επιστρέφονται τα πιο σχετικά έγγραφα με βάση το ερώτημα δίνοντας την δυνατότητα ευρετηρίασης ακόμα και εντός των πεδίων του εγγράφου αλλά και πραγματοποιείται βελτιστοποίησης της όλης της διαδικασίας.

**Ερώτηση 4:** Τρέξτε την Searcher και περιγράψτε πως γίνεται το searching παρουσιάζοντας αποτελέσματα από 4 χαρακτηριστικά ερωτήματα (queries) που κάνατε.

Για την πρώτη εκτέλεση της Searcher χρησιμοποιήθηκε το ερώτημα (query) 'end' όπως φαίνεται στο παρακάτω screenshot.



#### **Screenshot εισαγωγής ερωτήματος (query) 'end'**

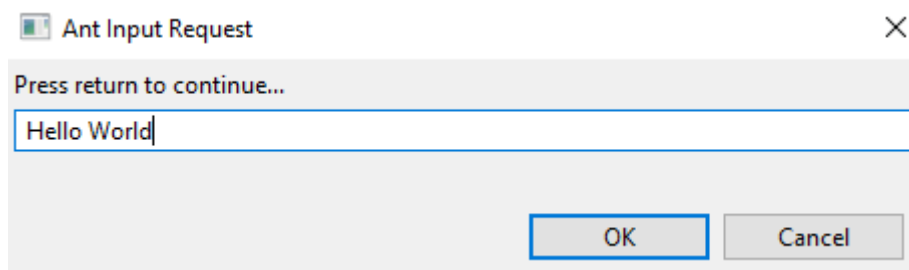
Παρακάτω φαίνεται στο screenshot που ακολουθεί το αποτέλεσμα της εκτέλεσης της Searcher για το ερώτημα (query) 'end'.

```
[echo] Running lia.meetlucene.Searcher...
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\apache1.1.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\mozilla_eula_firefox3.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\mozilla_eula_thunderbird2.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\mozilla1.1.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\apache2.0.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\cp11.0.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\ep11.0.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\cp11.0.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\cp12.0.txt
[java] E:\Files\0000\40 0000\70 000000 00000000 00000000\000000 000000 lia2e\src\lia\meetlucene\data\lp12.0.txt
[java] Found 12 document(s) (in 5 milliseconds) that matched query 'end':
BUILD SUCCESSFUL
Total time: 8 seconds
```

#### **Screenshot εκτέλεσης της Searcher για το ερώτημα (query) 'end'**

Αυτό που βλέπουμε αξιοποιώντας το παραπάνω screenshot μετά την εκτέλεση του εκτέλεσης της Searcher για το ερώτημα (query) 'end' είναι ότι εντοπίστηκαν 12 έγγραφα που αντιστοιχούσαν στο συγκεκριμένο ερώτημα που δόθηκε προς εύρεση.

Για την δεύτερη εκτέλεση της Searcher χρησιμοποιήθηκε το ερώτημα (query) 'Hello World' όπως φαίνεται στο παρακάτω screenshot.



### **Screenshot εισαγωγής ερωτήματος (query) 'Hello World'**

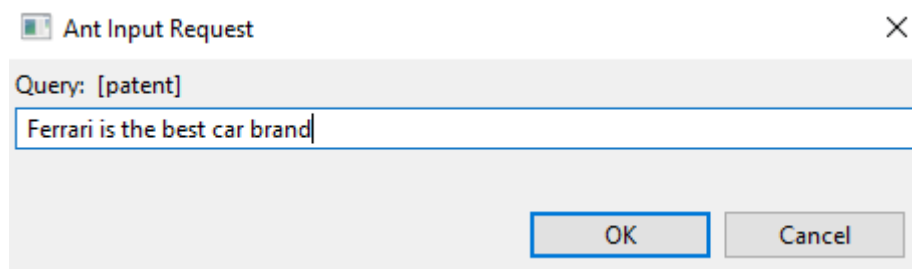
Παρακάτω φαίνεται στο screenshot που ακολουθεί το αποτέλεσμα της εκτέλεσης της Searcher για το ερώτημα (query) 'Hello World' .

```
[echo] Running lia.meetlucene.searcher...  
[java] Found 1 document(s) (in 8 milliseconds) that matched query 'Hello World':  
[java] E:\Files\00000\40 000\70 000000\00000000 0000000000\00000000\000000 000000\lia2e\src\lia\meetlucene\data\mozilla1.1.txt  
BUILD SUCCESSFUL  
Total time: 11 seconds
```

### **Screenshot εκτέλεσης της Searcher για το ερώτημα (query) 'Hello World'**

Αυτό που βλέπουμε αξιοποιώντας το παραπάνω screenshot μετά την εκτέλεση του εκτέλεσης της Searcher για το ερώτημα (query) 'Hello World' είναι ότι βρέθηκε μόνο 1 έγγραφο που αντιστοιχεί στο συγκεκριμένο ερώτημα που δόθηκε προς εύρεση.

Για την τρίτη εκτέλεση της Searcher χρησιμοποιήθηκε το ερώτημα (query) 'Ferrari is the best car brand' όπως φαίνεται στο παρακάτω screenshot.



### **Screenshot εισαγωγής ερωτήματος (query) 'Ferrari is the best car brand'**

Παρακάτω φαίνεται στο screenshot που ακολουθεί το αποτέλεσμα της εκτέλεσης της Searcher για το ερώτημα (query) 'Ferrari is the best car brand'.

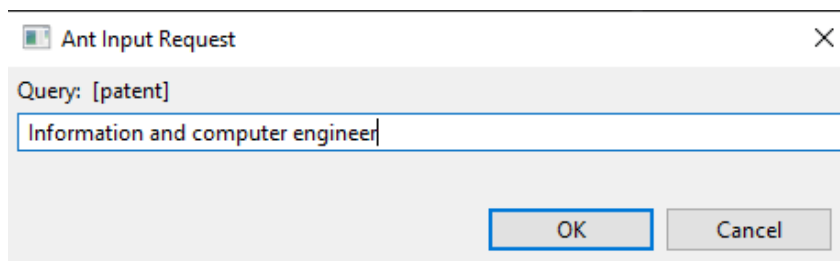
```
[echo] Running lia.meetlucene.Searcher...
[java] Found 3 document(s) (in 8 milliseconds) that matched query 'Ferrari is the best car brand':
[java] E:\Files\@0000\40 0000\70 0000000\00000000 000000000\00000000\0000000\0000000\lia2e\src\lia\meetlucene\data\gp11.0.txt
[java] E:\Files\@0000\40 0000\70 0000000\00000000 000000000\00000000\0000000\0000000\lia2e\src\lia\meetlucene\data\gp12.0.txt
[java] E:\Files\@0000\40 0000\70 0000000\00000000 000000000\00000000\0000000\0000000\lia2e\src\lia\meetlucene\data\gp13.0.txt
BUILD SUCCESSFUL
Total time: 5 seconds
```

### **Screenshot εκτέλεσης της Searcher για το ερώτημα (query) 'Ferrari is the best car brand'**

Αυτό που βλέπουμε αξιοποιώντας το παραπάνω screenshot μετά την εκτέλεση του εκτέλεσης της Searcher για το ερώτημα (query) 'Ferrari is the best car brand' είναι ότι βρέθηκαν 3 έγγραφα που αντιστοιχούν στο συγκεκριμένο ερώτημα που δόθηκε προς εύρεση.

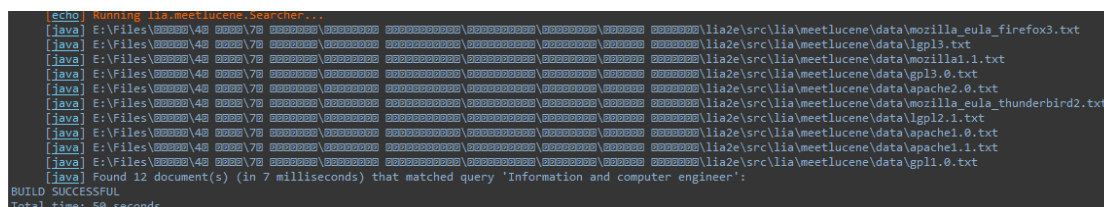


Για την τέταρτη εκτέλεση της Searcher χρησιμοποιήθηκε το ερώτημα (query) 'Information and computer engineer' όπως φαίνεται στο παρακάτω screenshot.



### **Screenshot εισαγωγής ερωτήματος (query) 'Information and computer engineer'**

Παρακάτω φαίνεται στο screenshot που ακολουθεί το αποτέλεσμα της εκτέλεσης της Searcher για το ερώτημα (query) 'Information and computer engineer'.



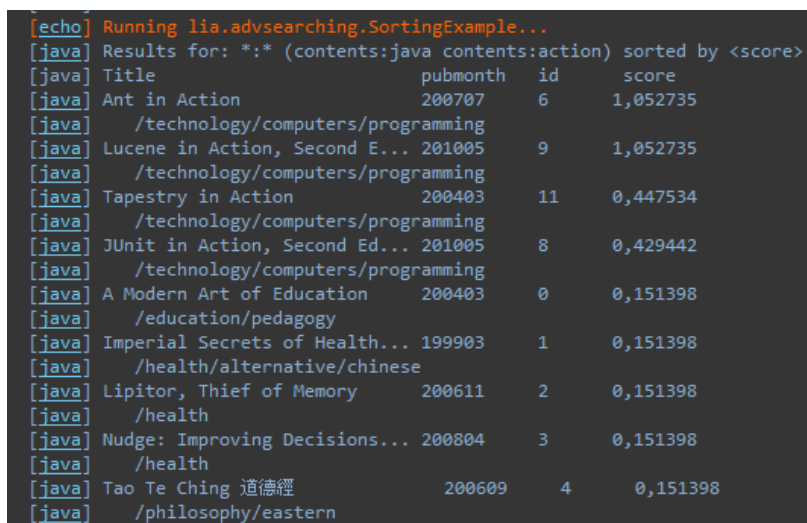
### **Screenshot εκτέλεσης της Searcher για το ερώτημα (query) 'Ferrari is the best car brand'**

Αυτό που βλέπουμε αξιοποιώντας το παραπάνω screenshot μετά την εκτέλεση του εκτέλεσης της Searcher για το ερώτημα (query) 'Information and computer engineer' είναι ότι βρέθηκαν 12 έγγραφα που αντιστοιχούν στο συγκεκριμένο ερώτημα που δόθηκε προς εύρεση.

Με βάση τα παραπάνω αποτελέσματα βλέπουμε ότι η λειτουργία της Searcher χρησιμοποιεί αλγορίθμους για να αντιστοιχίσει τις λέξεις ενός ερωτήματος ώστε να επιστρέψει τα σχετικά αποτελέσματα με βάση το ερώτημα που δόθηκε. Επιπλέον, ψάχνει το ευρετήριο το οποίο δημιουργήθηκε από το προηγούμενο ερώτημα της διαδικασίας του Indexer.

**Ερώτηση 5:** Τρέξτε το SortingExample και περιγράψτε τη λειτουργία της.

Αφού εκτελέστηκε η SortingExample εμφανίστηκαν τα αποτελέσματά της όπως φαίνεται στο παρακάτω screenshot.



```
[echo] Running lia.advsearching.SortingExample...
[java] Results for: *.* (contents:java contents:action) sorted by <score>
[java] Title                pubmonth  id      score
[java] Ant in Action          200707   6       1,052735
[java] /technology/computers/programming
[java] Lucene in Action, Second E... 201005   9       1,052735
[java] /technology/computers/programming
[java] Tapestry in Action        200403  11      0,447534
[java] /technology/computers/programming
[java] JUnit in Action, Second Ed... 201005   8       0,429442
[java] /technology/computers/programming
[java] A Modern Art of Education   200403   0       0,151398
[java] /education/pedagogy
[java] Imperial Secrets of Health... 199903   1       0,151398
[java] /health/alternative/chinese
[java] Lipitor, Thief of Memory    200611   2       0,151398
[java] /health
[java] Nudge: Improving Decisions... 200804   3       0,151398
[java] /health
[java] Tao Te Ching 道德經        200609   4       0,151398
[java] /philosophy/eastern
```

### Screenshot εκτέλεσης SortingExample και εμφάνιση μερικών αποτελεσμάτων

Αξιοποιώντας το παραπάνω screenshot βλέπουμε ότι γίνεται ταξινόμηση των δεδομένων με βάση το score που έχουν συγκεντρώσει.

## Ερώτηση 6: Παρουσιάστε τον κώδικα και τυχόν αλλαγές που κάνατε.

Ο κώδικας που υλοποιήθηκε προκειμένου να δημιουργηθεί μία μηχανή αναζήτησης συνδυάζοντας τους αλγόριθμους BM25 και TF-IDF είναι ο παρακάτω.

```
import pandas as pd
import spacy
from rank_bm25 import BM25Okapi
from tqdm import tqdm
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

df = pd.read_csv("_data/text_data.csv") # read csv file with project
Guttenberg documents
nlp = spacy.load("en_core_web_sm") # load spacy and change max
document length
nlp.max_length = np.inf

text_list = df.text.str.lower().values
tok_text=[] # for our tokenised corpus
#Tokenising using SpaCy:
for doc in tqdm(nlp.pipe(text_list, disable=["tagger",
"parser", "ner"])):
    tok = [t.text for t in doc if t.is_alpha]
    tok_text.append(tok)

bm25 = BM25Okapi(tok_text) # create a MB25 object
query = "rose" # query to use
tokenized_query = query.lower().split(" ") # preprocess the query
import time
results = bm25.get_top_n(tokenized_query, df.index, n=3) # get index
of top n documents
print(results)

query_df = pd.DataFrame() # create a new dataframe based on query
query_df["doc"] = ["query"]
query_df["text"] = [query]
query_df

vec = TfidfVectorizer(stop_words="english", lowercase=True) # compute
the tf idf dataframe of both corpus and query
data = vec.fit_transform(np.concatenate([df.text.values,
query_df.text.values]))
features = vec.get_feature_names_out()
tf_idf_df = pd.DataFrame(data=data.toarray(), columns=features)
tf_idf_df
# calculate the cosine similarity and get top 3 documents (we take 4
and remove the first one)
pd.Series(cosine_similarity(tf_idf_df.values, tf_idf_df.values)[-
1]).nlargest(n=4).index[1:].to_list()
```

Οι επιπρόσθετες πηγές που χρησιμοποιήθηκαν για την υλοποίηση του παραπάνω κώδικα είναι οι σύνδεσμοι των ιστοτόπων: [https://scikitlearn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html), <https://www.w3resource.com/pandas/series/series-nlargest.php>, <https://medium.com/analytics-vidhya/build-your-semantic-document-search-engine-with-tf-idf-and-google-use-c836bf5f27fb>

**Ερώτηση 7:** Φορτώστε κάποια documents π.χ. από το Wikipedia abstracts, project gutenberg (για τα οποία πρέπει να περιγράψετε τι περιέχουν) και τρέξτε τουλάχιστον 5 ερωτήματα (queries). Παρουσιάστε και σχολιάστε τα αποτελέσματα για τον κάθε αλγόριθμο.

Ο παρακάτω κώδικας διαβάζει τα αρχεία από το project Gutenberg και τα αποθηκεύει σε ένα csv αρχείο.

```
import pandas as pd
import os

docData=[]
docName=[]
# read directory of txt data
for file in os.listdir("_data/"):
    if not file.endswith("txt"):
        continue
    with open("_data/"+file,encoding="UTF-8") as infile:
        text = " ".join(infile.readlines()[:1000])
        docData.append(text)
        docName.append(file)
# store the text data and docId in a dataframe
df = pd.DataFrame()
df["doc"] = docName
df["text"] = docData

df.to_csv("_data/text_data.csv", encoding="utf-8", index=False) #
store the dataframe in a csv for future use
```

Στην συνέχεια ακολουθούν τα αποτελέσματα της παραπάνω διαδικασίας.

Query	BM-25 docId	TF_IDF docId
"whale boat"	[7, 2, 9]	[7, 2, 0]
"dagger heart"	[9, 4, 8]	[8, 9, 6]
"boat aboard"	[7, 1, 9]	[7, 1, 0]
"steel armour"	[0, 7, 6]	[0, 7, 6]
"rose"	[2, 4, 7]	[2, 4, 7]

Παρατηρούμε ότι τα αποτελέσματα των δύο αλγορίθμων είναι περίπου τα ίδια αφού ο BM-25 είναι μία βελτίωση του αλγορίθμου TF\_IDF.

**Ερώτηση 8:** Περιγράψτε τις διαφορές στη λειτουργία των δύο διαφορετικών προσεγγίσεων (του προγράμματος σε rython και του αντίστοιχου στη Lucene).

Η Lucene είναι μία υψηλής απόδοσης βιβλιοθήκη Java που χρησιμοποιείται για την ευρετηρίαση αλλά και την αναζήτηση κειμένου. Η Lucene παρέχεται σε μορφή ανοικτού κώδικα κάτι που σημαίνει ότι μπορεί να αναπτύσσεται και να βελτιώνεται με το πέρασμα του χρόνου από τους προγραμματιστές. Μπορεί να ευρετηριάσει και να εντοπίσει μεγάλο εύρος κειμένου καθώς μπορεί να δώσει πολλές επιλογές προκειμένου να προσαρμοστεί για την αναζήτηση κάποιου συγκεκριμένου ερωτήματος.

Όσον αφορά τους αλγορίθμους BM-25 και TF\_IDF παρατηρούμε ότι ο BM-25 αποτελεί μία βελτιστοποίηση του αλγορίθμου TF\_IDF. Ο Αλγόριθμος BM25 είναι στατικό μοντέλο καθώς υπολογίζει τη σχετικότητα του εγγράφου με βάση τη συχνότητα όρων του ερωτήματος εντός του εγγράφου αλλά και το μήκος του ενώ ο αλγόριθμος TF\_IDF εντοπίζει τα σχετικά έγγραφα

με βάση το πλήθος εμφάνισης του όρου καθώς και την σπανιότητά σε όλα τα έγγραφα. Αν ένας όρος δεν υπάρχει εντός του λεξικού τότε σε αυτήν την περίπτωση τον προσθέτουμε και μετά ξεκινάμε την διαδικασία της αναζήτησης.

**Ερώτηση 9:** Πως πιστεύετε ότι μπορεί να βελτιωθεί το καθένα από τα δύο προγράμματα (υλοποιημένα σε Python και σε Lucene). Ποιες δυνατότητες ή/και αλγόριθμοι θα βελτίωναν το κάθε project.