# Τίτλος Εργασίας:
# Μηχανή Αναζήτησης Επιστημονικών Άρθρων


| **Ονοματεπώνυμο:** | **ΑΜ:** |
|---|---|
| Σπήλιος Σπηλιόπουλος | 4495 |
| Κωνσταντίνος Χατζόπουλος | 1796 |


## Κλάσεις του Project:

**Main** -> Υπεύθυνη για την διεπαφή του χρήστη με το σύστημα και τις σχετικές λειτουργίες αναζήτησης.

**CSVReader** -> Υπεύθυνη για την ανάγνωση και την φόρτωση του αρχείου συνόλου δεδομένων στη μνήμη.

**SearchHistory** -> Υπεύθυνη για την φόρτωση του ιστορικού αναζήτησης και την δημιουργία εγγραφών αναζήτησης.

**SearchResultsWriter** -> Υπεύθυνη για την καταγραφή των αποτελεσμάτων σε αρχείο .txt.

**SearchResultsWriterHTML** -> Υπεύθυνη για την καταγραφή των αποτελεσμάτων σε αρχείο .html.

**LuceneSearch** -> Υπεύθυνη για την δημιουργία του ευρετηρίου αναζήτησης, την αναζήτηση των σχετικών αποτελεσμάτων, την ταξινόμηση τους και την δημιουργία ευρετηρίου για την παρουσίαση των αποτελεσμάτων.

## Εισαγωγή / Γενική περιγραφή του έργου:

Ο στόχος του έργου είναι η δημιουργία μιας μηχανής αναζήτησης που θα μπορεί να βρίσκει τα καλύτερα αποτελέσματα με βάση τις επιλογές που εισάγει ο χρήστης.

Τα δεδομένα τα οποία θα χρησιμοποιηθούν για αναζήτηση αποτελούν ερευνητικές και επιστημονικές εργασίες με διάφορες θεματολογίες. Τα δεδομένα αυτά έχουν πεδία τα οποία θα φανούν πολύ χρήσιμα κατά την αναζήτηση.

Ο χρήστης σε κάθε βήμα της αναζήτησης θα επιλέγει την δομή που θέλει να έχει το ερώτημά του (query). Η δομή του ερωτήματος μπορεί να διαφέρει. Η μηχανή αναζήτησης θα προσπαθεί να επιστρέψει τα καλύτερα δυνατά αποτελέσματα σχετικά με το ερώτημα του χρήστη.

Παρέχονται πολλές επιλογές στον χρήστη για την δόμηση του query και η κάθε μία από αυτές μπορεί να έχει ως συνέπεια διαφορετικά αποτελέσματα.

Μερικές από τις λειτουργίες που υποστηρίζει το σύστημα μας είναι:

1) Η επιλογή πεδίου αναζήτησης

2) Η χρήση συνώνυμων όρων και η σύγκριση τους για τον υπολογισμό της ερώτησης με τα περισσότερα σχετικά αποτελέσματα

3) Η επιλογή για Wildcard Search κ.α

Πριν την παρουσίαση των αποτελεσμάτων, παρέχεται η επιλογή στον χρήστη για ταξινόμηση των αποτελεσμάτων, καθώς και για το είδος της ταξινόμησης (Αύξουσα ή Φθίνουσα σειρά).
Τέλος, παρουσιάζονται τα αποτελέσματα στην Κονσόλα (Console) σε παρτίδες των 10.

Ο χρήστης επιλέγει αν θέλει να δει την επόμενη σελίδα του Ευρετηρίου, εφόσον αυτή είναι διαθέσιμη.

Μετά το πέρας μιας αναζήτησης, τα αποτελέσματα αποθηκεύονται σε 2 αρχεία, με όνομα:

a) το πεδίο αναζήτησης,

b) το query και

c) τον αριθμό των αποτελεσμάτων γι αυτήν την αναζήτηση.

Τα 2 αρχεία είναι της μορφής .txt και .html αντίστοιχα.

Επίσης, τα ερωτήματα όλων των χρηστών κρατούνται σε ένα αρχείο, έτσι ώστε να μπορούν να υλοποιηθούν διάφορες λειτουργίες όπως προτάσεις για σχετικές αναζήτησεις.

Το ιστορικό αναζήτησης αποθηκεύεται σε ένα αρχείο.

Το αρχείο του ιστορικού αναζήτησης “φορτώνεται” στο σύστημα κατά την εκκίνηση του.
Μετά το πέρας μιας ολοκληρωμένης αναζήτησης, ο χρήστης ερωτάται αν θέλει να συνεχίσει με μια επόμενη αναζήτηση.

Η διαδικασία επαναλαμβάνεται εως ότου ο χρήστης αποφασίσει να μην πραγματοποιήσει άλλη αναζήτηση.

Σε αυτό το σημείο το πρόγραμμα τερματίζει.

## Συλλογή (Corpus):

Το σύνολο δεδομένων αποτελεί ένα subset από το αρχικό dataset, το οποίο έχουμε αντλήσει από το Kaggle:
@https://www.kaggle.com/datasets/rowhitswami/nips-papers-1987-2019-updated/data?select=papers.csv

Πιο συγκεκριμένα, έχουν επιλεχθεί 500 τυχαίες εγγραφές από το σύνολο papers.csv.
Το αρχικό σύνολο δεδομένων περιέχει τις στήλες:

i) 'source_id'

ii) 'year'

iii) 'title'

iv) 'abstract'

v) 'full_context'

Για να καταφέρουμε να διαβάσουμε και να αναλύσουμε τα δεδομένα, δημιουργήσαμε το αρχείο clean_data_script.py.

Το συγκεκριμένο αρχείο έχει ως σκοπό να "καθαρίσει" τα δεδομένα και να επιλέξει ένα μέρος του αρχικού dataset, στην συγκεκριμένη περίπτωση 500 εγγραφές. Αφαιρεί την στήλη “abstract”, επειδή στις περισσότερες εγγραφές (αν όχι σε όλες) ήταν κενή. *[1]Έπειτα, αφαιρεί τα κενά για αλλαγές γραμμών στην στήλη "full_context". Μετά από την επεξεργασία του αρχείου, το νέο σύνολο δεδομένων αποθηκεύεται στο αρχείο papers_cleaned.csv.

## Ανάλυση κειμένου και κατασκευή Ευρετηρίου:

Πρώτα διαβάζουμε τα αρχεία από το αρχείο papers_cleaned.csv με τις μεθόδους readCSV και parseCSVLine της κλάσης CSVReader. Τα δεδομένα αποθηκεύονται ως ένα List<String[]>.

Κάθε γραμμή αντιπροσωπεύει μία εγγραφή και κάθε θέση της λίστας αντιπροσωπεύει το αντίστοιχο πεδίο.

Επόμενο βήμα είναι να δημιουργήσουμε ένα ευρετήριο για τα δεδομένα του αρχείου papers_cleaned.csv το οποίο θα χρησιμοποιήσουμε αργότερα για την αναζήτηση.

Για την ανάλυση κειμένου χρησιμοποιούμε τις έτοιμες συναρτήσεις και μεθόδους της Lucene. Με την μέθοδο indexCSV της κλάσης LuceneSearch, δημιουργούμε ένα ευρετήριο το οποίο βρίσκεται αποθηκευμένο στο Path:

"user.home/Documents/MetaData"

Έπειτα δημιουργούμε έναν αναλυτή StandardAnalyzer. Ο αναλυτής αυτός χρησιμοποιείται για την επεξεργασία και την κανονικοποίηση των δεδομένων που θα εισαχθούν στο ευρετήριο (π.χ., αφαίρεση σημείων στίξης, μετατροπή σε πεζά γράμματα κ.λπ.).

Μετά ρυθμίζουμε το configuration του IndexWriter. Η ρύθμιση αυτή καθορίζει πώς θα λειτουργήσει ο IndexWriter. Η επιλογή OpenMode.CREATE_OR_APPEND επιτρέπει την προσθήκη νέων εγγράφων σε ένα υπάρχον ευρετήριο ή τη δημιουργία νέου αν δεν υπάρχει
Ανοίγεται ένας IndexWriter με τις προκαθορισμένες ρυθμίσεις και τον κατάλογο ευρετηρίου indexDirectory.

Για κάθε γραμμή των δεδομένων CSV (που είναι πίνακες String), δημιουργείται ένα νέο έγγραφο Lucene (Document).

Σε κάθε έγγραφο προστίθενται τέσσερα πεδία: source_id, year, title, και full_text, με την τιμή που αντιστοιχεί στο αντίστοιχο στοιχείο της γραμμής CSV.

Τα πεδία προστίθενται ως TextField και αποθηκεύονται (Field.Store.YES), που σημαίνει ότι τα δεδομένα αυτά θα είναι διαθέσιμα για ανάκτηση μετά την εισαγωγή.

Τελικά, καταφέραμε να δημιουργήσουμε ένα ευρετήριο στον φάκελο MetaData, το οποίο θα χρησιμοποιήσουμε για την αναζήτηση.

## Προετοιμασία Ερωτήματος ( query ):

Προτού πραγματοποιήσουμε την αναζήτηση χρειάζεται να έχουμε έτοιμο το ερώτημα από τον χρήστη, μαζί με όλες τις επιλογές που θέλει να έχει.

Μια ακόμα λειτουργία που υλοποιεί το πρόγραμμα είναι η αποθήκευση και η φόρτωση του ιστορικού αναζήτησης από προηγούμενες αναζητήσεις σε προηγούμενα Sessions.

Ως Session θεωρούμε κάθε φορά που κάποιος χρήστης τρέχει το πρόγραμμα.

Κατά την εκκίνηση του προγράμματος φορτώνεται το ιστορικό αναζήτησης εφόσον αυτό δεν είναι κενό.

Στην κλάση Main ο χρήστης ερωτάται αρχικά σε ποιό πεδίο θα ήθελε να πραγματοποιήσει την αναζήτηση. Αφού επιλέξει ένα έγκυρο πεδίο αναζήτησης, ζητείται από τον χρήστη να εισάγει την ερώτηση του (query).

Αμέσως μετά, ανιχνεύουμε αν έχουν γίνει σχετικές αναζητήσεις στο παρελθόν (βάση του ιστορικού αναζήτησης). Αν υπάρχει κάποια σχετική αναζήτηση δίνεται η επιλογή στον χρήστη να την χρησιμοποιήσει. Η τεχνική αυτή υλοποιείται στην κλάση SearchHistory με την μέθοδο suggestQueries.

Για παράδειγμα :

Enter the field you want to search (e.g., source_id, year, title, full_text):
title
Enter the query:
nlp
Related queries:
1. neural nlp
2. NLP machine
3. neural nlp*

Συνεχίζοντας, το σύστημα παρέχει στον χρήστη την Αναζήτηση με wildcard search, την οποία υποστηρίζει η Lucene. Με τον όρο wildcard search εννοούμε:

Μια τεχνική αναζήτησης που επιτρέπει στους χρήστες να βρουν λέξεις ή φράσεις που ταιριάζουν με ένα πρότυπο (pattern) που περιέχει χαρακτήρες μπαλαντέρ (wildcards). Οι χαρακτήρες μπαλαντέρ χρησιμοποιούνται για να αντιπροσωπεύουν ένα ή περισσότερους άλλους χαρακτήρες σε μια λέξη ή φράση. Αυτή η τεχνική είναι ιδιαίτερα χρήσιμη όταν δεν είμαστε σίγουροι για την ακριβή μορφή της λέξης που ψάχνουμε ή θέλουμε να συμπεριλάβουμε παραλλαγές της λέξης στην αναζήτησή μας.

Οι χαρακτήρες μπαλαντέρ είναι οι εξής:

Για την επιλογή "multi" :

Αστερίσκος (*): Αντιπροσωπεύει οποιοδήποτε αριθμό χαρακτήρων (συμπεριλαμβανομένου του μηδενός χαρακτήρων).

Για παράδειγμα:

Η αναζήτηση για comp* θα βρει τις λέξεις "computer", "computation", "company", κ.λπ.

Για την επιλογή "single" :

Ερωτηματικό (?): Αντιπροσωπεύει ακριβώς έναν χαρακτήρα.

Παράδειγμα: Η αναζήτηση για te?t θα βρει τις λέξεις "test", "text", κ.λπ.

Σε αυτή τη φάση οι ειδικοί χαρακτήρες προστίθενται μόνο στο τέλος της λέξης αλλά με κατάλληλη παραμετροποίηση του κώδικα θα μπορούσαμε να τους τοποθετούμε σε οποιοδήποτε σημείο.
Σε επόμενο βήμα έχουμε την λειτουργία για πρόταση του καλύτερου δυνατού ερωτήματος. Η τεχνική αυτή πραγματοποιείται με την χρήση συνώνυμων όρων.

Το σύστημα ψάχνει για συνώνυμες λέξεις με αυτές που υπάρχουν στο query. Για κάθε πιθανή συνώνυμη λέξη εξετάζει τον αριθμό των αποτελεσμάτων που θα επιστραφούν αν ο χρήστης πραγματοποιήσει την αναζήτηση με το συγκεκριμένο query. Έτσι τελικά επιστρέφει στον χρήστη το προτεινόμενο και "καλύτερο" query και τον ρωτάει αν θέλει να το χρησιμοποιήσει.

Για παράδειγμα:

Αν το αρχικό query είναι: "software research"

Και ο πίνακας συνωνύμων είναι ο εξής :

```
synonymsMap.put("software", new String[]{"application", "program", "tool"});
synonymsMap.put("research", new String[]{"study", "investigation", "inquiry", "exploration"});
```

Τότε έχουμε στην κονσόλα:

Do you want to see an improved query suggestion? (yes/no):
yes
Total results for initial query **'software research': 0**
Total results for query **'application research': 4**
Total results for query **'program research': 1**
Total results for query **'tool research': 0**
Total results for query **'software study': 1**
Total results for query **'software investigation': 1**
Total results for query **'software inquiry': 0**
Total results for query **'software exploration': 3**
Improved query suggestion: **application research**
Do you want to use the improved query? (yes/no):

Σε αυτό το σημείο έχει νόημα το Flag για την ταξινόμηση που θα δούμε παρακάτω στο επίπεδο της αναζήτησης *[2] .

## Αναζήτηση:

Για αυτήν την λειτουργία χρησιμοποιούμε την μέθοδο search της κλάσης LuceneSearch.

Τα αντικείμενα της κλάσης LuceneSearch δέχονται ως όρισμα ένα Path το οποίο αντιπροσωπεύει το Directory όπου τα ευρετήρια της μεθόδου indexCSV αποθηκεύονται.

Η μέθοδος search δέχεται ως ορίσματα την ερώτηση (query) καθώς και το πεδίο αναζήτησης (search field).

Στην μέθοδο δημιουργείται ένας αναγνώστης (reader) και ένας ερευνητής (searcher). Ο searcher, περιέχει την βασική μέθοδο search της Lucene, όπου επιστρέφει αποτελέσματα βάση ενός Score. Μεγαλύτερο Score σημαίνει περισσότερη σχετικότητα με το ερώτημα του χρήστη. Πρωτού αποθηκεύσουμε τα αποτελέσματα, χρησιμοποιούμε άλλη μια λειτουργία της Lucene για το Highlighting τον λέξεων οι οποίες είναι σχετικές με την αναζήτηση του χρήστη. Το Highlighting γίνεται στοχευμένα μόνο στο πεδίο αναζήτησης. Επίσης, επιβεβαιώνουμε ότι κάθε αποτέλεσμα είναι ξεχωριστό .

Αφού ολοκληρωθεί και αυτή η διαδικασία, αποθηκεύουμε τα αποτελέσματα.

Σε αυτό το σημείο έχουμε ένα List<Document> το οποίο περιέχει όλα τα σχετικά αποτελέσματα για μία αναζήτηση. Έτσι δημιουργούμε ένα ευρετήριο το οποίο μοιράζει τα αποτελέσματα σε σελίδες, με κάθε σελίδα να περιέχει 10 αποτελέσματα.

Επίσης σε αυτή τη μέθοδο πραγματοποιείται και η λειτουργία για την ταξινόμηση των αποτελεσμάτων βασισμένη στην χρονολογία δημοσίευσης της εργασίας. Έχει τεθεί ένα global flag (εκ των υστέρων χρειάστηκε). Αυτό γιατί κατά την σύγκριση των αποτελεσμάτων στην λειτουργία για πρόταση του καλύτερου query (βλέπε *[2])καλούμε πάλι την μέθοδο search αλλά δεν θέλουμε να ρωτήσουμε τον χρήστη για ταξινόμηση. Θέλουμε να συγκρίνουμε απλώς πιο ερώτημα έχει τα περισσότερα αποτελέσματα.

Τέλος, επιστρέφεται το ευρετήριο το οποίο έχει την δομή :
List<List<Document>> όπου :

i) Document = Εγγραφή (αποτέλεσμα αναζήτησης)

ii) List<Document> = Λίστα από εγγραφές σε δέσμες των 10 (αντιπροσωπεύει την σελίδα)

iii) List<List<Document>> = Λίστα από τις λίστες (αντιπροσωπεύει τις συνολικές σελίδες του ευρετηρίου που κατασκευάσαμε).

## Παρουσίαση Αποτελεσμάτων:

Είναι σημαντικό να αναφέρουμε πως πρωτού την παρουσίαση των αποτελεσμάτων, μία καινούργια εγγραφή αποθηκεύεται στο αρχείο του ιστορικού αναζήτησης. Η δομή της εγγραφής είναι :

“ (Index). Field: (search_field), Query: (query), NumResults: (numResults) “

και υλοποιείται με την μέθοδο addSearchEntry

Τα SearchEntries είναι οι εγγραφές που περιέχονται στο SearchHistory και είναι αντικείμενα της εσωτερικής κλάσσης SearchEntry.

Η παρουσίαση των αποτελεσμάτων γίνεται με 3 τρόπους.

**Ο πρώτος τρόπος:**

Αφορά την εκτύπωση των αποτελεσμάτων σε δέσμες των 10 ανά σελίδα. Πρώτα εκτυπώνεται ο συνολικός αριθμός των αποτελεσμάτων για μια αναζήτηση καθώς και ο αριθμός των σελίδων του ευρετηρίου. Ξεκινά η εκτύπωση της πρώτης σελίδας. Εφόσον υπάρχει επόμενη σελίδα ο χρήστης ερωτάται αν θέλει να δει περισσότερα αποτελέσματα. Στην περίπτωση που θελήσει εκτυπώνεται η επόμενη σελίδα. Σε περίπτωση που ο χρήστης θέλει να δει περισσότερα αποτελέσματα αλλά δεν υπάρχει επόμενη διαθέσιμη σελίδα, επιστρέφεται μήνυμα πως αυτή ήταν η τελευταία σελίδα και η αναζήτηση τερματίζει. Σε περίπτωση που ο χρήστης δεν επιθυμεί να δει περισσότερα αποτελέσματα, η αναζήτηση τερματίζει.

Με την κατάλληλη παραμετροποίηση ο χρήστης θα μπορούσε να πηγαίνει και πίσω στις σελίδες (όχι μόνο μπροστά).

Μόλις τερματίζει μια αναζήτηση, ο χρήστης έχει την επιλογή να πραγματοποιήσει μια καινούργια αναζήτηση αλλιώς το πρόγραμμα τερματίζει.

**Ο δεύτερος/τρίτος τρόπος:**

Αφορά την αποθήκευση των αποτελεσμάτων σε δέσμες των 10 ανά σελίδα. Σε αυτήν την περίπτωση αποθηκεύονται όλα τα σχετικά αποτελέσματα και όχι μόνο τα αποτελέσματα της σελίδας που παρουσιάζονται στον χρήστη από την κονσόλα.

Τα αποτελέσματα αποθηκεύονται στην μορφή .txt και .html με τις κλάσεις SearchResultsWriter και SearchResultsWriterHTML αντίστοιχα. Σκεφτήκαμε πως το Highlighting θα μπορούσε να φανεί καλύτερα όταν το αρχείο βρισκόταν σε html αρχείο.

*[1]: Αυτό το κάναμε επειδή είχαμε πρόβλημα με το parsing του αρχείου, καθώς δεν χρησιμοποιήσαμε κάποια εξωτερική βιβλιοθήκη, όπως για παράδειγμα την OpenCSV.

1,952 words 13,625 characters