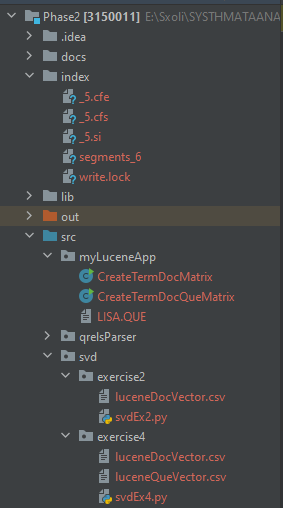
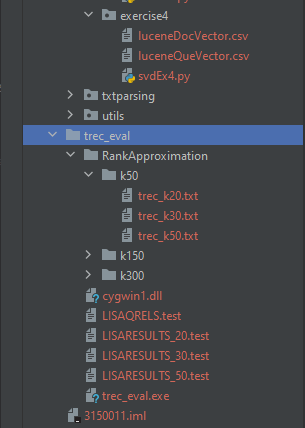
**Συστήματα ανάκτησης πληροφοριών**

**2η Φάση Προγραμματιστικής Εργασίας**

**Βιβλιοθήκη LISA**

**Ον/μο:** ΒΙΤΑΛΗΣ ΙΩΑΝΝΗΣ

ΑΜ: 3150011

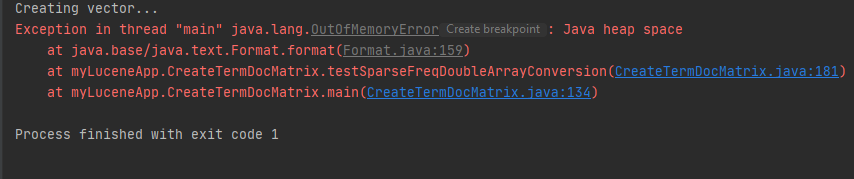
**Τελική Δομή:**

**Σημαντικο!**

Ξεκίνησα ακολουθώντας την ροή της εκφώνησης κατασκευάζοντας τον termXdocument πίνακα για τον πρώτο ερώτημα με **13579** terms, χρησιμοποιώντας την TXTparsing όπως ακριβώς την είχα κατασκευάσει για την 1η Φάση της προγραμματιστικής εργασίας. Ο τελικός termXdocument πίνακας είχε διαστάσεις **13579x6004** με τον οποίο έκανα επιτυχώς SVD ανάλυση για το 2ο βήμα χρησιμοποιώντας python.

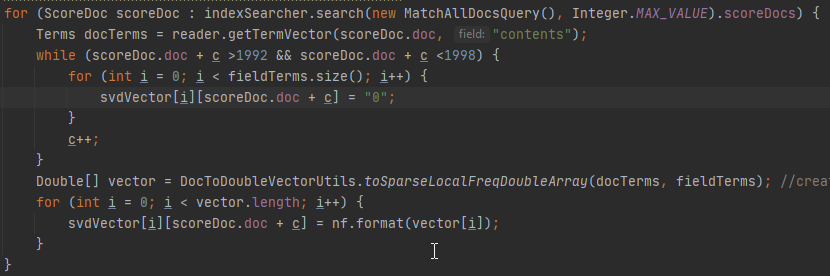
Προχωρώντας στο βήμα 3, οπου απαιτούνταν η ενσώματωση των terms των queries στα terms των documents δημιουργώντας ένα ενιαίο index κατέληξα με **13774** terms και συνολικά **6039** documents(docs+queries) από τον οποίο θα δημιουργούσα τα δύο αρχεία που απαιτούνται για τον υπολογισμό της συνημιτονοειδής ομοιότητας των διανυσμάτων των ερωτημάτων με τα διανύσματα των κειμένων.

Παρόλα αυτά έτρεχε η “testSparseFreqDoubleArrayConversion(IndexReader reader)” για περίπου 2 λεπτά και η εφαρμογή μου εμφάνιζε το εξής error:

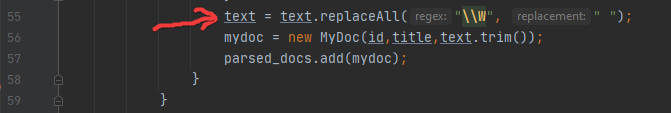


Αρχικά έψαξα για λάθη στον κωδικά μου χωρίς να παρατηρήσω κάτι, δεδομένου και οτι το πρόγραμμα λειτουργούσε σωστά για το 1ο ερώτημα. Επιχείρησα να αυξήσω το max heap size αλλάζοντας τα memory settings και αυξάνοντας την μέγιστη διαθέσιμη μνήμη μέχρι και 4GB με τον εξής τρόπο στα settings του IntelliJ -Xmx4096m, αύξησα την ελάχιστη μνήμη που γίνεται allocate με το -Xms2024m, αλλά και πάλι συνέχιζε να εμφανίζεται το ίδιο error.

Με περαιτέρω debugging είδα οτι το πρόβλημα εμφανιζόταν μετα από περίπου 5900 loops του παρακάτω, στην μέθοδο “testSparseFreqDoubleArrayConversion(IndexReader reader)”



Τελικά κατέληξα στο να γυρίσω στην κλάση TXTParsing και να αφαιρέσω από όλα τα texts όλους τους ειδικούς χαρακτήρες επιχειρώντας να μειώσω το μέγεθος των terms και άρα το μέγεθος του πίνακα που δημιουργούσα με την παρακάτω εντολή στο TXTParsing.java:



Με αυτήν την προσθήκη κατάφερα να ολοκληρώσω την Φαση 3 αλλά με πλεον **13033 terms,** μπορεί να γίνει αναπαραγωγή του error αν αφαιρεθεί η εντολή στην σειρά 55 του αρχείου TXTParsing.java και γίνει run του “CreateTermDocQueMatrix.java”.

PC Specs: CPU-Ryzen 5 3600, RAM-16GB@3200MHz, Storage-500GB NVME SSD

**Αναφορά**

**1.)Αναπαράσταση συλλογή κειμένων σαν πίνακα διαστάσεων termsXdocuments.**

Για την αναπαράσταση του πίνακα επεξεργάστηκα τον κώδικα που μας δώθηκε στο εργαστήριο προκειμένου να λειτουργεί με την συλλογή κειμένων LISA.

Από την 1η προγραμματιστική εργασία χρησιμοποιήσα τις κλάσεις “MyDoc” και “TXTParsing” που χρησιμοποιούνται από τον “CreateTermDocMatrix” αρχικά για την παραγωγή του Index και έπειτα για την παραγωγή του πίνακα και την εγγραφή του σε ένα αρχείο “luceneDocVector.csv” με σκοπό να χρησιμοποιηθεί για SVD ανάλυση.

Κατα την δημιουργία του “IndexWriter” πρόσθεσα το εξής configuration έτσι ώστε να κάνει overwrite ήδη υπάρχοντα ευρετήρια εάν αυτά υπάρχουν στον φάκελο index.



Για την δημιουργία του Index η διαδικασία είναι παρόμοια με την πρώτη φάση καλώντας αυτή την φορά την “addDocWithTermVector();”με μοναδική αλλαγή το εξής πεδίο:

Πλέον δημιουργούμε στο ευρετήριο προσθέτουμε τύπους “Field” αντί για “TextField” επειδή χρειάζεται η παραμετροποίηση που προσφέρεται από το πεδίο “type”.

Αφού δημιουργηθεί το ευρετήριο, δημιουργούμε έναν “IndexReader” για να το διαβάσει και τον χρησιμοποιούμε σαν όρισμα στην μέθοδο “testSparseFreqDoubleArrayConversion(reader)” που θα δημιουργήσει τον πίνακα termsXdocuments.

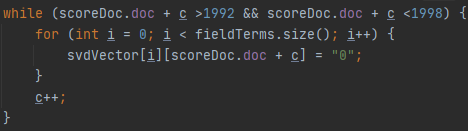
Η συλλογή μου σε αυτό το σημείο αποτελούνταν από:

**Terms:** 13161

**Documents:** 5999

**IDs:** 6004

Επειδή η αξιολόγηση γίνεται με βάση τα IDs δημιουργώ έναν πίνακα διαστάσεων 13161Χ6004 και παρατηρώ οτι στο αρχείο “LISA1.501” τα λείπουν τα κείμενα με ID 1993 έως 1997, προσθέτω μερικα «κενά» κείμενα στον πίνακα σε αυτές τις θέσεις με τον εξής τρόπο:



Με χρήση “DocToDoubleVectorUtils.toSparseLocalFreqDoubleArray(docTerms, fieldTerms);“ δημιουργούμε ένα vector[] που αποθηκεύει πόσες φορές εμφανίζεται το κάθε term μέσα στο document που εξετάζουμε.

Διατρέχουμε τον vector[] και αποθηκεύουμε τα αποτελέσματα του στα κατάλληλα rows του svdVector[][] κρατώντας σταθερή την στήλη, η στήλη αυξάνεται όταν εξετάσουμε το επόμενο Document.

Με χρήση της “writeVector(svdVector)”, αν υπάρχει ήδη αρχείο με το όνομα “luceneDocVector.csv” το κάνουμε overwrite, αν δεν υπάρχει τότε το δημιουργούμε και γράφουμε σε αυτό τον πίνακα termsXdocuments

**2.)Παραγοντοποίηση του αραιού πίνακα εφαρμόζωντας SVD ανάλυση**

Αναζήτησα βιβλιοθήκες που να μπορούν να εφαρμόσουν SVD ανάλυση στον πίνακα, μια εξ’αυτών ήταν η βιβλιοθήκη “JAMA” χωρίς όμως να παράγει κάποιο αποτέλεσμα ακόμα και αν το «άφησα να τρέχει» για περίπου δύο ώρες. Συνάδελφοι ανέφεραν ότι κατάφεραν SVD ανάλυση με “JAMA” αλλά χρειάστηκε να περιμένουν 7+ ώρες.

Σε αυτό το σημείο αποφάσισα οτι το debugging θα ήταν αδύνατο αν χρειαζόταν να περιμένω κάθε φορά 7 ώρες για την SVD ανάλυση οπότε χρησιμοποίησα Python.

Στον φάκελο “src\\svd\\exercise2” βρίσκεται το άρχειο “luceneDocVector.csv” που δημιουργήθηκε από την “CreateTermDocMatrix” και το python script “svdEx2.py” με το οποίο έκανα SVD ανάλυση.

**3.) Αναπαράσταση ερωτημάτων ως διανύσματα**

Για τον σκοπό της άσκησης και για να είναι εμφανής η διαφορά μεταξύ του 1ου και του 3ου ερωτήματος δημιούργησα μια καινούρια main με όνομα “CreateTermDocQueMatrix”. Τροποποίησα κατάλληλα την κλάση TXTParsing επειδή τα Queries που μου δώθηκαν είχαν διαφορετική δομή.

*Δομή Document*

ID: Document 3532

Title: Summary of main text

Mesh: Main text of the document

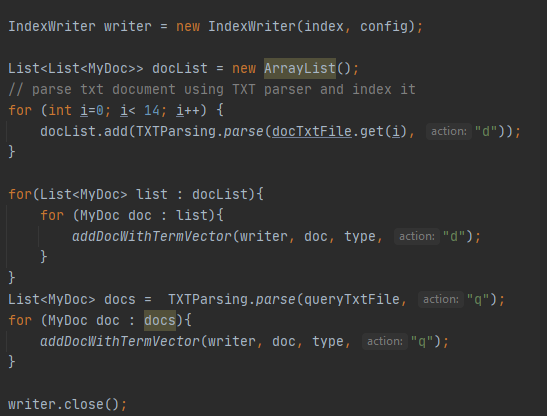
*Δομή Queries*

ID: 3532

Title:

Mesh: Main text of the document

Επομένως με την προσθήκη των κατάλληλων if statement στο TXTParsing καταφέρνω μία ομοιόμορφη δομή μεταξύ των δύο για να ενταχθούν οι όροι στο ευρετήριο.



Με χρήση της “addDocWithTermVector()” δημιουργούμε το ευρετήριο και κλείνουμε τον writer.

Ανοίγουμε έναν “IndexReader” και διαβάζουμε το ευρετήριο που δημιουργήθηκε, και καλούμε την “testSparseFreqDoubleArrayConversion(reader)” για να αναπαραστήσω την συλλογή κειμένων και τα queries σε έναν ενιαίο πίνακα με ένα ενιαίο ευρετήριο. Το ευρετήριο πλέον έχει μέγεθος 13303 > 13161 που είχαμε στο πρώτο ερώτημα, λογικό καθώς προσθέσαμε και τα terms που έχουν τα queries και δεν εμφανίζονται στα Docs.

Ο αριθμός των συνολικών Documents είναι πλέον 6039, αφού πήραμε τα 6004 Documents της συλλογής και προσθέσαμε τα 35 κείμενα από τα queries

Δημιουργούμε το 2D Array svdVector με μέγεθος terms(13303)XDocumentsQueries(6039), οπου κάθε στήλη αναπιριστά ένα Document ή ένα query και κάθε row αναπαριστά πόσες φορές εμφανίζεται το term στο Document/Query.

Καλούμε την writeVector(svdVector) προκειμένου να δημιουργήσουμε 2 .csv αρχεία

Το ένα ονομάζεται “luceneDocVector.csv” και έχει μέγεθος termsXdocuments δηλαδή 13303Χ6004.

Το δεύτερο αρχείο ονομάζεται “luceneQueVector.csv” και έχει μέγεθος termsXqueries δηλαδή 13303Χ35 και τα αποθηκεύουμε στον φάκελο “src\\svd\\exercise4” προκειμένου να χρησιμοποιηθούν με χρήση της python για τον υπολογισμό της συνημιτονοειδής ομοιότητας των διανυσμάτων.

**4.)**Για τον υπολογισμό της συνημιτονοειδής ομοιότητας των διανυσμάτων των ερωτημάτων με τα διανύσματα των κειμένων χρησιμοποίησα Python όπως και στο ερώτημα 2, στον πίνακα termsXdocuments (διαστάσεων: 13303x6004).

Στον φάκελο “src\\svd\\exercise4” βρίσκεται το άρχειο “luceneDocVector.csv” και “luceveQueVector” που δημιουργήθηκε από την “CreateTermDocQueMatrix”, και το python script “svdEx4.py” με το οποίο έκανα SVD ανάλυση και τον υπολογισμό της συνημιτονοειδής ομοιότητας.

Έχω 35 queries και για κάθε query τρέχω την συνάρτηση cosine\_similarity, η οποία με την “txq.iloc[:, q] παίρνει όλα τα terms για το query που δίνεται σαν όρισμα, και υπολογίζω την συνημιτονοειδή ομοιότητα με τον τύπο της.

Κανω sort με το -cosineSimilarity για να ταξινομηθούν τα φθίνουσα σειρά και τα γράφω στο αντίστοιχο αρχείο, στο .py αρχείο που παρέδωσα έχω για RankApproximation k=50 και γράφω αρχεία LISARESULTS για 20, 30 και 50 κείμενα.

**5.)Αποτελέσματα στον φάκελο trec\_eval\\RankApproximation\\k50**

**Rank Approximation = 50**

|  |  |  |
| --- | --- | --- |
| **Μέτρο**  **Αξιολόγησης**  **K πρώτα**  **Ανακτηθέντα κείμενα** | MAP  (Mean Average Precision) | avgPre@k  (Average Precision @k) |
| Κ = 20 | 0.0303 | P\_05 = 0.0471  P\_10 = 0.0441  P\_15 = 0.0451  P\_20 = 0.0485 |
| Κ = 30 | 0.0316 | P\_05 = 0.0471  P\_10 = 0.0324  P\_15 = 0.0353  P\_20 = 0.0368 |
| Κ = 50 | 0.0582 | P\_05 = 0.0412  P\_10 = 0.0382  P\_15 = 0.0314  P\_20 = 0.0279 |

Συγκριτικά με την πρώτη φάση, παρατηρούμε κατά πολύ μικρότερο map και Average Precision @k που υποδεικνύει οτι για την συγκεκριμένη συλλογή αυτή η μέθοδος ανάκτησης κειμένων δεν είναι τόσο καλή και ίσως να ταίριαζε καλύτερα το μοντέλο ανάκτησης διανυσματικού χώρου.

Παρόλα αυτά παρατηρούμε «λογικούς» αριθμούς καθώς το map αυξάνεται όσο αυξάνονται τα K πρώτα ανακτηθέντα κείμενα.

**6)**

**Αποτελέσματα στον φάκελο trec\_eval\\RankApproximation\\k150**

**Rank Approximation = 150**

|  |  |  |
| --- | --- | --- |
| **Μέτρο**  **Αξιολόγησης**  **K πρώτα**  **Ανακτηθέντα κείμενα** | MAP  (Mean Average Precision) | avgPre@k  (Average Precision @k) |
| Κ = 20 | 0.0371 | P\_05 = 0.0647  P\_10 = 0.0676  P\_15 = 0.0686  P\_20 = 0.0676 |
| Κ = 30 | 0.0319 | P\_05 = 0.0588  P\_10 = 0.0647  P\_15 = 0.0608  P\_20 = 0.0647 |
| Κ = 50 | 0.0264 | P\_05 = 0.0294  P\_10 = 0.0353  P\_15 = 0.0373  P\_20 = 0.0397 |

**Αποτελέσματα στον φάκελο trec\_eval\\RankApproximation\\k300**

**Rank Approximation = 300**

|  |  |  |
| --- | --- | --- |
| **Μέτρο**  **Αξιολόγησης**  **K πρώτα**  **Ανακτηθέντα κείμενα** | MAP  (Mean Average Precision) | avgPre@k  (Average Precision @k) |
| Κ = 20 | 0.0340 | P\_05 = 0.0706  P\_10 = 0.0765  P\_15 = 0.0824  P\_20 = 0.0691 |
| Κ = 30 | 0.0303 | P\_05 = 0.0529  P\_10 = 0.0588  P\_15 = 0.0686  P\_20 = 0.0721 |
| Κ = 50 | 0.0239 | P\_05 = 0.0294  P\_10 = 0.0353  P\_15 = 0.0353  P\_20 = 0.0338 |

Για Rank Approximation 150 και 300 παρατηρούμε οτι όσο το K αυξάνεται, το MAP μειώνεται σε αντίθεση με τα αποτελέσματα για Rank Approximation = 50 και σε αντίθεση και μετα αποτελέσματα της 1ης φάσης της εργασίας.

Αυτό μπορεί να είναι κάποιο πρόβλημα στον κώδικα κατα τον υπολογισμό της συνημιτονοειδούς ομοιότητας, κάποια ιδιομορφία της συλλογής ή και να φταίει που χρειάστηκε να αφαιρέσω τους ειδικούς χαρακτήρες από τα κείμενα πριν την δημιουργία του Index.