Πανεπηστίμιο Κρήτης

Τμήμα Επιστήμης Υπολογιστών

HY463 Συστήματα Ανάκτησης Πληροφοριών

Εξάμηνο: Άνοιξη 2021

Στοιχεία:

Μέλος	1ο
Ονοματεπώνυμο	Νικόλαος Γουνάκης
AM	3932
Email	csd3932@csd.uoc.gr

Πίνακας Περιεχομένων

Εισαγωγή

Το project υλοποιήθηκε σε python και για να τρέξει θα χρειαστεί να εγκαταστήσετε ότι βιβλιοθήκη περιέχει το αρχείο requirements.txt με την εντολή:

```
$ pip install -r requirements.txt .
```

Γενικά υλοποιήθηκε και το απλό indexing το οποίο είναι εξερεπικά γρήγορο αλλά καταναλώνει πολύ μνήμη και το partial indexing το οποίο είναι αργό χρησιμοποιέι λιγότερη μνήμη αλλα μπορεί να ευρετηριάσει μεγαλύτερες συλλογές εγγράφων.

Εφόσον το project έγινε σε διαφορεπική γλώσσα απο την προτεινόμενη (java) χρειάστηκε να υλοποιήσω κάποια απο τα δοθέντα κομμάπια κώδικα απο την αρχή ή να χρησιμοποιήσω βιβλιοθήκες που κάνουν παρόμοια δουλειά.

Διαδικασία Ευρετηρίασης

Για απλό indexing:

```
$ python app.py -index
```

Για partial indexing:

```
$ python app.py -pindex
```

Στην συνέχεια ανοίγει γραφική διεπαφή για την επιλογή φακέλου και είναι η μόνη γραφική διδεπαφη που υπάρχει προς το παρόν. Τα υπόλοιπα γίνονται μέσο του τερματικού.

Διάβασμα Αρχείων

Εφόσον επιλέξουμε φάκελο το πρόγραμμα αρχίζει να διαβάζει αναδρομικά όλους του υπο-φακέλους και διαβάζει μόνο τα αρχεία με κατάληξη `.nxml` . Τα υπόλοιπα τα αγνοεί.

Στο αρχείο `readxml.py` φαίνεται η υλοποίηση , όπου το πρόγραμμα διαβάζει τα απαραίπητα tags. Στην συνέχεια περνάνε απο tokenizer και χρησιμοποιούνται για να φιαχτεί ένα document object (`Document.py`) το οποίο αναπαρηστά ένα έγγραφο.

Αξίζει να αναφερθεί οπι κρατιέται πληροφορία σε ποία tags εμφανίζεται κάθε λέξη και πόσες φορές μέσα σε κάθε document object.

Tokenizing

Η υλοποίηση βρίσκεται στο `tokenizer.py` . Η συνάρτηση που κάνει tokenize παίρνει ως παράμετρο ένα string η έναν πίνακα απο strings και επιστρέφει έναν πίνακα με tokens όπου κάθε token αποτελείται μόνο απο αλφαριθμητικούς χαρακτήρες.

Αφαίρεση Stopwords

Η υλοποίηση βρίσκεται πάλι στο `tokenizer.py` . Διαβάζει τα δοθέντα αρχεία `stopwordsEn.txt` και `stopwordsGr.txt` χρησιμοποιείται απο την συνάρτηση `tokenize(s)` για να τα αφαιρεί.

Stemming

Το stemming γίνεται μέσα στον constructor του document object (`Document.py`) κατα τον υπολογισμό συχνοτήτων των λέξεων.

Για το stemming χρησιμοποιήθηκε η βιβλιοθήκη `nltk` όπου υποστιρίζει αγγλικό stemming

Ανεστραμμένο Ευρετήριο

Και με τις δύο τεχνικές indexing παράγεται το ίδιο ανεστραμμένο ευρετήριο.

DocumentsFile

Έχει τη μορφή:

```
doc_id  path  norm
```

PostingFile

Έχει τη μορφή:

```
doc_id  tf  appearances  pointer to DocumentsFile
```

όπου `appearances` : `{'abstract': [89, 94, 96, 98], 'body': [624, 722]}` ένα dictionary που γράφει σε ποιό σημείο μέσα tag εμφανίζεται ο όρος και απο το length του array μπορούμε να μάθουμε και πόσες φορές.

VocabularyFile

Έχει τη μορφή:

```
term_id  df  pointer to PostingFile
```

όπου `term_id` : το string που αναπαρηστά έναν όρο

Απλό Indexing

1. Το πρόγραμμα ξεκινά να διαβάζει όλα τα documents απο έναν φάκελο.
2. Στην συνέχεια απο όλα τα documents κάνει extract τα terms και δημιουργεί το vocabulary
3. Έπειτα εφόσον έχει όλα τα documents και το vocabulary στην μνήμη ξεκινά να παράγει το inverted file

Partial Indexing

1. Το πρόγραμμα ξεκινά να διαβάζει αρχεία απο έναν φάκελο
2. Όταν η μνήμη φτάσει στο 80% τότε ξεκινά να παράγει ένα partial inverted file
3. Επαναλαμβάνονται τα βήματα 1 και 2 μέχρις ότου να έχουν διαβαστεί όλα τα αρχεία του φακέλου
4. Εφόσον έχουν διαβαστεί όλα τα αρχεία του φακέλου και έχου παραχθεί όλα τα partial inverted files , τότε ξεκινάει η διαδικασία του merging όπως περιγραφεται στο δοσμένο pdf
   `Partial Indexing and Merging`

Αποτιμητής Ερωτήσεων

Για να τρέξει το πρόγραμμα σε query evaluation mode το τρέχουμε χωρίς κανένα argument:

```
$ python app.py
```

Στην συνέχεια φορτώνεται το Vocabulary στην μνήμη και το σύστημα ρωτάει τον χρήστη να επιλέξει ανάμεσα σε:

1. diagnosis
2. test
3. treatment

Εφόσον επιλέξει τότε του δίνεται η ευκαιρία να εισάγει summary ή description.

Έπειτα εφόσον πατήσει `enter` ο χρήστης τότε γίνεται evaluation του query χρησιμοποιόντας το `Διανυσματικό Μοντέλο` και επιστρέφονται όλα τα έγγραφα τα οποία περιέχουν όρους απο το query σε αύξουσα σειρα με βάση το score.

Γενικά , δεν έχει υλοποιηθεί ακόμα κάποιος μηχανισμός για τον διαχωρισμό των εγγράφων στις 3 παραπάνω κατηγορίες διότι παραπάνω έμφαση έδωσα στο να υλοποιήσω και τις 2 τεχνικές indexing. Θα υλοποιηθεί όμως στην επόμενη φάση που είναι πιο χρήσιμο.

Μετρήσεις

Ευρετηρίαση

Method	MiniCollection (4.89 MB)	MedicalCollection/00 (309 MB)	Medical Collection (4.56 GB)
Simple Indexing	6.148648262023926 s	480.54263734817505 s	-
Partial Indexing	6.488290309906006 s (no need for merging)	796.763519525528 s	doc analysis: 7852.338171958923

Αποτίμηση Ερωτήσεων

Query	CollectionIndex size	Time
64-year-old woman with uncontrolled diabetes, now with an oozing, painful skin lesion on her left lower leg.	3.06 MB	0.03461456298828125 s
-	220 MB	1.91103196144104 s
-	-	-

Επίλογος

Ένα πρόβλημα που άργησα να παρατηρήσω και αποτρέπει το partial indexing να δουλέψει κανονικά είναι οτι η βιβλιοθήκη που χρησιμοποιώ για random access file (linecache) φορτώνει το αρχειό στην μνήμη και χρησιμοποιεί cache για να κάνει fetch γρήγορα lines απο το αρχείο , με αποτέλεσμα να γεμίζει η μνήμη κατα το merging της μεγάλης συλλογής. Οπότε στην επόμενη φάση θα πρέπει να κατασκευάσω άλλη μέδοθο κρατόντας το offset κάθε entry στο inverted file.

Αναφορές

1. pip
2. nltk
3. linecache