Εισαγωγή

Στην εργασία αυτή θα υλοποιήσετε μία μίνι μηχανή αναζήτησης (search engine). Οι δομές δεδομένων και η προσέγγιση που θα εξετάσουμε είναι μια πάρα πολύ απλουστευμένη μορφή της υποδομής που χρησιμοποιούν μηχανές αναζήτησης όπως το Google ή το Bing. Πιο συγκεκριμένα, καλείστε να υλοποιήσετε μία εφαρμογή που θα χρησιμοποιεί ένα ανεστραμμένο ευρετήριο (inverted index) για να αποθηκεύσει με κατάλληλο τρόπο ένα σύνολο από αρχεία κειμένου (documents) και θα είναι σε θέση να απαντάει ερωτήματα με λέξεις-κλειδιά (keyword queries) επιστρέφοντας στο χρήστη τα top-k κείμενα που είναι πιο σχετικά με το query.

Interface της εφαρμογής

Η εφαρμογή θα ονομάζεται `minisearch` και θα χρησιμοποιείται ως εξής:

```
./minisearch -i docfile -k K
```

- Η παράμετρος `K` λέει στην εφαρμογή πόσα το πολύ αποτελέσματα να γυρίσει για κάθε query. Π.χ. Για `K 5` η εφαρμογή θα επιστρέφει 5 αποτελέσματα (το πολύ) για κάθε query. Αν ένα query έχει λιγότερα από `K` θα επιστρέφει μόνο αυτά.
- Το `docfile` (ή κάποιο άλλο όνομα αρχείου) είναι ένα αρχείο που περιλαμβάνει τα κείμενα που θα αποθηκεύσει και θα δεικτοδοτήσει η εφαρμογή. Κάθε γραμμή του αρχείου αυτού είναι ένα document με λέξεις. Για παράδειγμα αν τα περιεχόμενα του αρχείου είναι:

  ```
  0 The quick brown fox leaped over the lazy lazy dog
  1 Quick brown foxes leaped over lazy dogs for fun
  ```

  Σημαίνει πως έχουμε δύο κείμενα, ένα για κάθε γραμμή. Η πρώτη λέξη σε κάθε γραμμή είναι το id του συγκεκριμένου κειμένου (document). Μπορείτε να υποθέσετε πως τα ids δίνονται σε σειρά αλλά θα πρέπει να ελέγχετε πως αυτό ισχύει και αν όχι να χειριστείτε το λάθος παρουσιάζοντας το κατάλληλο μήνυμα και βγαίνοντας από την εφαρμογή.
- Ξεκινώντας, η εφαρμογή σας θα πρέπει να ανοίξει το αρχείο και να διαβάσει ένα-ένα τα κείμενα (γραμμή γραμμή δηλαδή) και να κάνει δύο πράγματα: α) να τα αποθηκεύσει στη μνήμη γιατί θα χρειαστούν κατά την εκτέλεση των ερωτημάτων και β) να δημιουργήσει το inverted index επίσης στη μνήμη. Λεπτομέρειες για το index και τις βοηθητικές δομές δεδομένων στην επόμενη ενότητα. Αυτό που θα σημειώσουμε εδώ είναι πως όταν, κατά την επεξεργασία κάθε κειμένου, χρειαστεί να αναγνωρίσετε τις λέξεις, θεωρείστε πως αυτές διαχωρίζονται μεταξύ τους με έναν ή περισσότερους κενούς χαρακτήρες (είτε space είτε tab). Για παράδειγμα, το κείμενο: `state-of-the-art index` θεωρούμε πως αποτελείται από δύο λέξεις: `state-of-the-art` και `index`.
- Όταν η εφαρμογή τελειώσει τη δημιουργία του index, θα περιμένει είσοδο από το χρήστη από το πληκτρολόγιο. Ο χρήστης θα μπορεί να δίνει τις ακόλουθες εντολές:
  - `/search q1 q2 q3 q4 … q10`

    Ο χρήστης αναζητά κείμενα που περιέχουν μία ή περισσότερες από τις λέξεις `qi` μετά την εντολή. Οι λέξεις αυτές αποτελούν το query. Για παράδειγμα για την εντολή `/search brown lazy` θα επιστραφούν και τα δύο παραπάνω κείμενα. Η εντολή αναμένει μία τουλάχιστον λέξη και το πολύ 10. Αν δοθούν πάνω από 10 λέξεις, χρησιμοποιούνται μόνο οι 10 πρώτες. Μόλις δοθεί η εντολή `/search` η εφαρμογή βρίσκει τα κείμενα που περιέχουν τις λέξεις, υπολογίζει ένα σκορ βάσει του πόσο σχετικό είναι είναι το κάθε κείμενο με το query, ταξινομεί τα κείμενα βάσει αυτού του σκορ και τυπώνει τα top-K. Λεπτομέρειες για τον υπολογισμό του σκορ στην επόμενη ενότητα.

    Ο χρήστης βλέπει στην οθόνη του τον αύξοντα αριθμό του αποτελέσματος, το id του κειμένου, το σκορ, το κείμενο, καθώς επίσης και πού βρίσκονται οι λέξεις του query μέσα στο κείμενο. Πιο συγκεκριμένα για το query `/search brown lazy` για τα δύο κείμενα πιο πάνω, χρήστης θα πρέπει να βλέπει το εξής:

```
/search brown lazy
1.( 1)[0.0341] Quick brown foxes leaped over lazy dogs for fun
                     ^^^^^                   ^^^^
2.( 0)[0.0320] The quick brown fox leaped over the lazy lazy dog
                          ^^^^^                     ^^^^ ^^^^
```

(Σημείωση: τα νούμερα 0.0341 και 0.0320 είναι τυχαία).

Για την εκτύπωση των αποτελεσμάτων θα πρέπει να κρατήσετε συγκεκριμένο εύρος (το οποίο αποφασίζετε εσείς) για κάθε πεδίο που εμφανίζεται πριν το κείμενο, ώστε να φαίνονται ευθυγραμμισμένα τα αποτελέσματα όπως παραπάνω. Τα πεδία θα πρέπει να εμφανίζονται με τη συγκεκριμένη σειρά του παραδείγματος ενώ το φορμάτ τους είναι αυστηρό (αύξων αριθμός: αριθμός τελεία, id κειμένου μέσα σε παρενθέσεις, σκορ μέσα σε αγκύλες).

Στην περίπτωση που το κείμενο δε χωράει σε μια γραμμή, θα πρέπει να τυπώνετε την ένδειξη για τη θέση των λέξεων εναλλάξ με το κείμενο. Για παράδειγμα στην παραπάνω εκτύπωση αν το πλάτος της σελίδας ήταν το μισό:

```
/search brown lazy
1.( 1)[0.0341] Quick brown foxes leaped
                     ^^^^^
               over lazy dogs for fun
                    ^^^^
2.( 0)[0.0320] The quick brown fox leaped
                          ^^^^^
               over the lazy lazy dog
                        ^^^^ ^^^^
```

Για να βρείτε το τρέχον μήκος της γραμμής του τερματικού, μπορείτε να χρησιμοποιείστε την `ioctl()` στο `STDOUT`.

- `/df`

  Εκτυπώνεται το document frequency vector, δηλαδή κάθε λέξη από το αρχείο εισόδου μαζί με τον αριθμό των κειμένων στα οποία εμφανίζεται. Στο παράδειγμα των δύο κειμένων πιο πάνω θα εμφανίζεται το εξής:

  ```
  /df
  the 1
  quick 1
  over 2
  leaped 2
  lazy 2
  fun 1
  foxes 1
  fox 1
  for 1
  dogs 1
  dog 1
  brown 2
  The 1
  Quick 1
  ```

  (Σημείωση, είναι καλό να είναι ταξινομημένες οι λέξεις, αλλά όχι απαραίτητο).

  Αν υπάρχει λέξη μετά την εντολή τότε τυπώνεται το document frequency μόνο για τη λέξη. Για παράδειγμα:

  ```
  /df leaped
  leaped 2
  ```

- `/tf id word`
  Τυπώνει το term frequency για τη συγκεκριμένη λέξη στο συγκεκριμένο κείμενο, δηλαδή το πόσες φορές εμφανίζεται η λέξη `word` στο document με το δεδομένο `id`. Για παράδειγμα:
  `/tf 0 lazy`
  `0 lazy 2`
- `/exit`
  Έξοδος από την εφαρμογή. Βεβαιωθείτε πως ελευθερώνετε σωστά όλη τη δεσμευμένη μνήμη.

Δομές δεδομένων

Για την υλοποίηση της εφαρμογής μπορείτε να χρησιμοποιήσετε C ή C++. Δεν μπορείτε να χρησιμοποιήσετε όμως την Standard Template Library (STL). Όλες οι δομές δεδομένων θα πρέπει να υλοποιηθούν από εσάς. Βεβαιωθείτε πως δεσμεύετε μόνο όση μνήμη χρειάζεστε, π.χ. η ακόλουθη τακτική δε συνιστάται:

```
int doc_ids[512];     // store up to 512 docs, but really we don't know how many
```

Επίσης βεβαιωθείτε πως απελευθερώνετε τη μνήμη σωστά και κατά την εκτέλεση του προγράμματός σας αλλά και κατά την έξοδο.

Για να ολοκληρώσετε την άσκηση θα χρειαστεί, μεταξύ άλλων, να υλοποιήσετε τις εξής δομές δεδομένων.

1. Ένα map που να αποθηκεύει το id και το περιεχόμενο κάθε document. Αυτό μπορεί να είναι ένα απλό array αλλά θα πρέπει να το δεσμεύσετε δυναμικά. Αυτό το map θα χρειαστεί για την παρουσίαση των αποτελεσμάτων στο χρήστη της εντολής `/search`.

0 →	The quick brown fox leaped over the lazy lazy dog
1 →	Quick brown foxes leaped over lazy dogs for fun

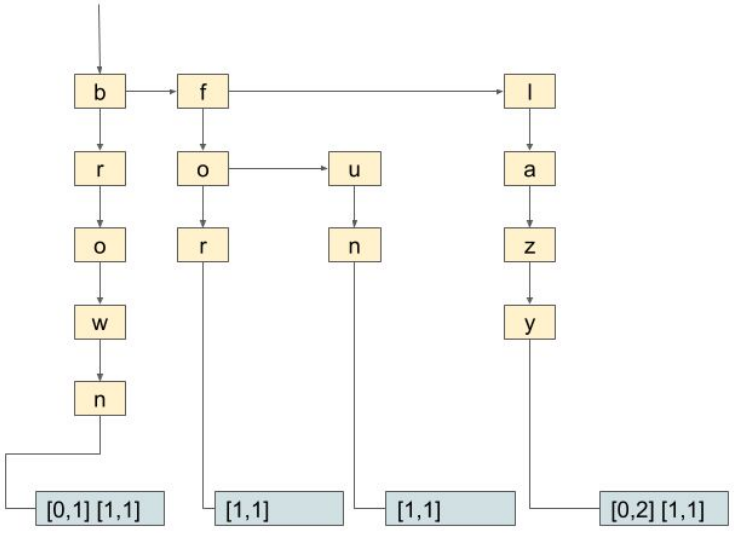
2. Το inverted index είναι ουσιαστικά μια αντιστοίχηση από κάθε λέξη που υπάρχει στη συλλογή από documents προς τα documents αυτά. Για παράδειγμα, για τα δύο κείμενα που έχουμε πιο πάνω, ένα μέρος του inverted index αναπαρίσταται ως εξής:

brown →	[0, 1] [1, 1]
lazy →	[0, 2] [1, 1]
for →	[1, 1]
fun →	[1, 1]
...	...

   Στο αριστερό μέρος είναι το κλειδί, κάθε λέξη δηλαδή, ενώ στο δεξί υπάρχει μία λίστα από τα ids των κειμένων καθώς επίσης και το πόσες φορές εμφανίζεται η κάθε λέξη στο κάθε κείμενο. Η λίστα αυτή ονομάζεται postings list. Για παράδειγμα η λέξη `lazy` εμφανίζεται δύο φορές στο document 0 (βλ. `[0,2]`) και μία φορά στο document 1.

   Για να μπορούμε να κάνουμε γρήγορες αναζητήσεις στο index, χρειαζόμαστε μια δομή που να μπορούμε πολύ γρήγορα να βρούμε τα postings lists.

Για αυτό το σκοπό θα υλοποιήσετε ένα Trie στο οποίο θα αποθηκεύονται οι λέξεις των κειμένων. Στα φύλλα του Trie θα περιλαμβάνεται η εκάστοτε posting list. Για παράδειγμα στο ακόλουθο σχήμα φαίνεται μέρος του Trie και των postings lists. Το Trie που θα δημιουργήσετε στη μνήμη θα πρέπει να κρατάει όλη τη συλλογή των documents.



Συμβουλή: επειδή θα χρειαστείτε το document frequency για κάθε λέξη, δηλαδή το πόσες εγγραφές έχει η κάθε posting list, μπορείτε να αποθηκεύετε τον αριθμό αυτό στην αρχή του posting list. Εναλλακτικές υλοποιήσεις είναι εξίσου σωστές προφανώς.

Δεδομένου ενός query, για παράδειγμα `brown fun`, ο υπολογισμός της απάντησης γίνεται ως εξής:
Για κάθε λέξη, βρίσκουμε το posting list από το Trie. Στο παράδειγμά μας `[0,1] [1,1]` και `[1,1]` αντίστοιχα. Για κάθε document id D που εμφανίζεται είτε στη μία είτε στην άλλη λίστα υπολογίζετε το πόσο σχετικό είναι αυτό το document με το query (δηλαδή το relevance score του) χρησιμοποιώντας τον τύπο BM25:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Όπου:
- qi: είναι λέξη του query
- $f(qi, D)$: είναι το συχνότητα της λέξης (term frequency) qi, στο κείμενο
- $|D|$: είναι το πλήθος των λέξεων του κειμένου
- $avgdl$: είναι το μέσο πλήθος λέξεων από όλα τα κείμενα που έχουμε στο index
- $IDF(qi)$: είναι το inverse document frequency (αναπαριστά το σε πόσα κείμενα υπάρχει η λέξη qi) και υπολογίζεται ως εξής:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

- N: είναι το πλήθος των κειμένων στο index
- $n(qi)$: είναι το πλήθος των κειμένων που περιέχουν τη λέξη qi

- $k1$ και b: είναι παράμετροι που μας επιτρέπουν να βελτιστοποιήσουμε το search engine. Για την άσκηση αυτή θεωρήστε πως $k1=1.2$ και $b=0.75$.
Αφού υπολογίσουμε το score για κάθε κείμενο βρίσκουμε τα `K` πιο υψηλά scores και παρουσιάζουμε τα αντίστοιχα κείμενα στο χρήστη με τη μορφή που συζητήσαμε στην προηγούμενη ενότητα.

Βεβαιωθείτε πως για κάθε υπο-πρόβλημα που χρειάζεται να επιλύσετε κατά την υλοποίηση της άσκησης, χρησιμοποιείτε τον πιο αποτελεσματικό αλγόριθμο ή δομή δεδομένων. Π.χ, για την αποτελεσματική εύρεση των `top-K` κειμένων μπορείτε να υλοποιήσετε ένα heap ή μια άλλη κατάλληλη δομή δεδομένων/αλγόριθμο. Όποιες σχεδιαστικές αποφάσεις και επιλογές κάνετε κατά την υλοποίηση, θα πρέπει να τις περιγράψετε στο README, στα παραδοτέα.

Παραδοτέα

- Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματός σας. 1-2 σελίδες ASCII κειμένου είναι αρκετές. Συμπεριλάβετε την εξήγηση και τις οδηγίες για το compilation και την εκτέλεση του προγράμματός σας σε ένα αρχείο README μαζί με τον κώδικα που θα υποβάλετε.
- Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο ή αλλού θα πρέπει να αναφερθεί στον πηγαίο κώδικά σας αλλά και στο παραπάνω README.
- Όλη η δουλειά σας (πηγαίος κώδικας, Makefile και README) σε ένα tar.gz file με ονομασία OnomaEponymoProject1.tar.gz. Προσοχή να υποβάλετε μόνο κώδικα, Makefile, README και όχι τα binaries.

Διαδικαστικά

- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com. Η πλήρης διεύθυνση είναι https://piazza.com/uoa.gr/spring2018/k24/home. Η παρακολούθηση του φόρουμ στο Piazza είναι υποχρεωτική.
- Το πρόγραμμά σας θα πρέπει να γραφεί σε C (ή C++). Στην περίπτωσή που χρησιμοποιήσετε C++ δεν μπορείτε να χρησιμοποιήσετε τις έτοιμες δομές της Standard Template Library (STL). Σε κάθε περίπτωση το πρόγραμμά σας θα πρέπει να τρέχει στα Linux workstations του Τμήματος.
- Ο κώδικάς σας θα πρέπει να αποτελείται από τουλάχιστον δύο (και κατά προτίμηση περισσότερα) διαφορετικά αρχεία. Η χρήση του separate compilation είναι επιτακτική και ο κώδικάς σας θα πρέπει να έχει ένα Makefile.
- Βεβαιωθείτε πως ακολουθείτε καλές πρακτικές software engineering κατά την υλοποίηση της άσκησης. Η οργάνωση, η αναγνωσιμότητα και η ύπαρξη σχολίων στον κώδικα αποτελούν κομμάτι της βαθμολογίας σας.
- Στα πρώτα δύο μαθήματα κυκλοφορεί hard-copy λίστα στην τάξη στην οποία θα πρέπει οπωσδήποτε να δώσετε το όνομά σας και το Unix user-id σας. Με αυτό τον τρόπο μπορούμε να γνωρίζουμε ότι προτίθεστε να υποβάλετε την παρούσα άσκηση και να προβούμε στις κατάλληλες ενέργειες για την τελική υποβολή της άσκησης.

Άλλες σημαντικές παρατηρήσεις

- Οι εργασίες είναι ατομικές.
- Όποιος υποβάλει / παρουσιάσει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο μηδενίζεται στο μάθημα.
- Αν και αναμένετε να συζητήσετε με φίλους και συνεργάτες το πώς θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που δεν επιτρέπεται. Οποιοσδήποτε βρεθεί αναμεμιγμένος σε αντιγραφή κώδικα απλά παίρνει μηδέν στο μάθημα. Αυτό ισχύει για όσους εμπλέκονται ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
- Οι ασκήσεις προγραμματισμού μπορούν να δοθούν με καθυστέρηση το πολύ 3 ημερών και με ποινή 5% για κάθε μέρα αργοπορίας. Πέραν των 3 αυτών ημερών, δεν μπορούν να κατατεθούν ασκήσεις.