ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Inverted Search Engine

Τρίγκας Ευστάθιος 1115201700167
Τσιαμούρας Δημήτριος 1115201700168

Εισαγωγή

Στα πλαίσια του μαθήματος “Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα” μας ζητήθηκε να δημιουργήσουμε μια εφαρμογή Ανεστραμμένης μηχανής αναζήτησης σε C/C++.

Ανεστραμμένη Μηχανή Αναζήτησης

Ανεστραμμένη Μηχανή Αναζήτησης (inverted search engine) Μια συνηθισμένη μηχανή αναζήτησης έχει ένα μεγάλο και ως επί το πλείστον στατικό πλήθος από κείμενα στη διάθεσή της. Ανά τακτά χρονικά διαστήματα γίνονται στη μηχανή διαφορετικά ερωτήματα με στόχο την εύρεση και ανάκτηση των σχετικών κειμένων με αυτά.

Κλασικό παράδειγμα μιας τέτοιας εφαρμογής είναι η μηχανή αναζήτησης της Google, στην οποία οι χρήστες γράφουν ερωτήματα σχετικά με την πληροφορία που αναζητούν, και η μηχανή επιστρέφει τα κείμενα που θεωρεί ότι σχετίζονται. Μια ανεστραμμένη μηχανή αναζήτησης λειτουργεί με τον ίδιο τρόπο, αλλά προϋποθέτει ότι τα ερωτήματα που της δίνονται είναι ως επί το πλείστον στατικά, και τα διαθέσιμα κείμενα είναι αυτά που έρχονται δυναμικά.

Σχετικό παράδειγμα μιας τέτοιας εφαρμογής είναι το Twitter, όπου τα κείμενα εδώ αντιστοιχούν σε tweets. Νέα tweets, διαφορετικού περιεχομένου έρχονται συνεχώς στην εφαρμογή και πρέπει να ανακτηθούν όταν γίνει σχετική αναζήτηση.

Στα πλαίσια αυτής της εργασίας έχουμε αναπτύξει μια εφαρμογή η οποία δέχεται ένα πλήθος από κείμενα που φτάνουν με συνεχή ροή (stream of documents) και ερωτήματα (queries) που πρέπει να απαντηθούν, εφόσον σχετίζονται με κάποιο κείμενο. Ένα κείμενο αναπαριστάται ως μια ακολουθία από λέξεις που χωρίζονται μεταξύ τους με κενά, ενώ ένα ερώτημα ως ένα σύνολο από λέξεις. Τόσο τα ερωτήματα όσο και τα κείμενα αναπαριστώνται από ένα σύνολο από λέξεις (keywords). Όποτε φτάνει ένα νέο κείμενο, η εφαρμογή θα πρέπει γρήγορα να βρει όλα τα ερωτήματα που σχετίζονται με αυτό. Για να θεωρήσουμε ότι ένα κείμενο απαντά σε ένα ερώτημα πρέπει το κείμενο να περιέχει λέξεις που ταιριάζουν με όλες τις λέξεις του ερωτήματος. Η διαδικασία με την οποία εξετάζουμε αν μια λέξη ταιριάζει με μια άλλη λέγεται keyword matching και μπορεί να έχει διάφορες παραλλαγές:

- να είναι οι δύο λέξεις απολύτως ίδιες μεταξύ τους (exact matching),
- να υπάρχει ανοχή να διαφέρουν σε κάποια σημεία.(approximate matching)
    - απόσταση απόκλισης επεξεργασίας (edit distance)
    - απόσταση απόκλισης Hamming (Hamming distance)

Στόχος είναι να μειωθεί όσο το δυνατό περισσότερο ο χρόνος απόκρισης του συστήματος στις απαντήσεις των ερωτημάτων.

Δομές δεδομένων

Το πρόγραμμα διαβάζει μια συνεχή ροή queries από ένα αρχείο. Υπάρχουν 3 διαφορετικοί τύποι από queries.

0: Exact Match
1: Edit Distance
2: Hamming Distance

Οι τύποι αυτοί αναφέρονται στις παραλλαγές των keyword matching που αναφέραμε στην εισαγωγή. Για κάθε τύπο ερωτήματος υπάρχει και μία διαφορετική δομή που αποθηκεύονται οι λέξεις και θα αναλυθούν παρακάτω. Όλες οι λέξεις των queries πρέπει να αποθηκευτούν στις δομές.
Όταν υπάρχουν διπλότυπες λέξεις δεν αποθηκεύεται η λέξη πάνω από μία φορά, αλλά στο payload της λέξεις το query id. Πχ μπορεί να έχω την λέξη “fly” που υπάρχει στο q3,q5,q6 και στην δομή θα υπάρχει μία φορά η λέξη με μια λίστα για όλα τα queries που υπάρχει.

*μπορείτε να δείτε μια γραφική αναπαράσταση των δομών στο αρχείο indexes_visualization.pdf

Δομή για exact match

Στην δομή αυτή έχουμε έναν πίνακα με 32 θέσεις (32 είναι το max lenght των λέξεων). Κάθε στοιχείο του πίνακα δείχνει σε ενά άλλο hash table με Ν θέσεις που κάθε στοιχείο του δείχνει σε μία λίστα απο λέξεις. Χρησιμοποιείτε επίσης και ένα bloom filter που χρησιμεύει μετέπειτα, στην αναζήτηση λέξεων.

Παράδειγμα εισαγωγής λέξης:
Διαβάζω μια λέξη την κάνω 2 φορές hash και ανανεώνω το bloom filter. Βρίσκω το hash table που αναλογεί στο μέγεθος αυτής της λέξης και χρησιμοποιώντας το hash value βρίσκω την λίστα που πρέπει να μπει αυτή η λέξη. Αν στη λίστα υπάρχει είδη αυτή η λέξη τότε βάζω στο payload της αποθηκευμένης λέξης το query id της λέξης που διαβάστηκε. Αν δεν υπάρχει αποθηκεύω την λέξη στην λίστα.

Δομή για edit distance

Στην δομή αυτή έχουμε ένα hash table με N θέσεις. Κάθε θέση δείχνει σε μια λίστα από δείκτες. Οι δείκτες αυτοί δείχνουν σε κόμβους ενός bk_tree και εκεί αποθηκεύονται οι λέξεις. Το bk tree αναλύεται στην συνέχεια.

Π*αράδειγμα εισαγωγής λέξης:*
Διαβάζω μια λέξη και παίρνω το hash value της. Ανάλογα το hash value πάω στην αντίστοιχη λίστα με τους πίνακες. Στη λίστα για κάθε λέξη τσεκάρω αν υπάρχει. Αν δεν υπάρχει την προσθέτω αλλιώς ανανεώνω το payload.

Δομή για hamming distance

Στην δομή αυτή έχουμε έναν πίνακα με 32 θέσεις . Κάθε στοιχείο του πίνακα δείχνει σε ενά άλλο hash table με N θέσεις που κάθε στοιχείο του δείχνει σε μία λίστα από δείκτες σε bk tree και εκεί αποθηκεύονται οι λέξεις. Έχουμε ένα bk tree για κάθε διαφορετικό length. Κρατάμε επίσης ένα πίνακα root table για να αποθηκεύουμε τις ρίζες τον bk tree που θα χρειαστεί στην αναζήτηση.

Παράδειγμα εισαγωγής λέξης:
Διαβάζω μια λέξη και ανάλογα το length πάω στο αντίστοιχο hash table. Παίρνω το hash value της λέξης. Ανάλογα το hash value πάω στην αντίστοιχη λίστα με τους πίνακες. Στη λίστα για κάθε λέξη τσεκάρω αν υπάρχει. Αν δεν υπάρχει την προσθέτω, αλλιώς ανανεώνω το payload.

Bk Tree

Οι δομές για hamming distance και edit distance χρησιμοποιούν έχουν στις δομές του bk trees όπως αναφέρεται παραπάνω.

Ένα BK-tree είναι μία δενδρική δομή δεδομένων, που ειδικεύεται ως ευρετήριο δεδομένων σε ένα μετρικό χώρο. Ένας μετρικός χώρος είναι ουσιαστικά ένα σύνολο αντικειμένων, μαζί με μία μετρική συνάρτηση

απόστασης $d(x, y)$ που ορίζεται για κάθε ζεύγος αντικειμένων στο σύνολο. Η μετρική συνάρτηση πρέπει να ικανοποιεί ένα σύνολο αξιωμάτων προκειμένου να είναι καλώς ορισμένη, όπως θα αναλυθεί παρακάτω.

Η δομή δεδομένων BK-tree προτάθηκε από τους Burkhard και Keller [1] για το πρόβλημα της αναζήτησης σε ένα σύνολο από κλειδιά, προκειμένου να βρεθεί το κλειδί που είναι εγγύτερα σε ένα συγκεκριμένο κλειδί ερώτησης. Ο απλοϊκός τρόπος επίλυσης του προβλήματος αυτού είναι απλά η σύγκριση του κλειδιού της ερώτησης με κάθε στοιχείο του συνόλου. Ένα BK-tree επιτρέπει να εξαιρεθεί ένα μεγάλο τμήμα του συνόλου των κλειδιών από τις συγκρίσεις.
Πιο συγκεκριμένα μπορείτε να διαβάσε πληροφορίες στον παρακάτω σύνδεσμο : https://signal-to-noise.xyz/post/bk-tree/

Match Document

Όταν διαβάζεται από το αρχείο ένα document, το πρόγραμμα πρέπει για κάθε λέξη του να ψάξει στις δομές ποιες λέξεις ταιριάζουν με αυτή την λέξη. Μόλις βρει τις λέξεις που ταιριάζουν πρέπει να βρει τα queries που όλες του οι λέξεις βρέθηκαν στην αναζήτηση. Προκειμένου να ξέρουμε για κάθε queries ποιές λέξεις έχει, χρησιμοποιούμε μία βοηθητική δομή που αποθηκεύει όλα τα queries. Για τις λέξεις των Documents γίνεται επίσης deduplication.

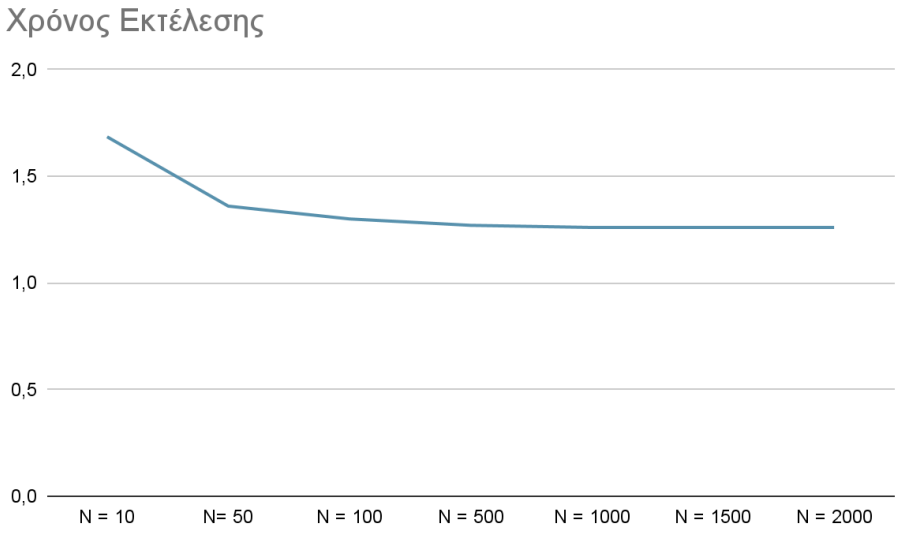
Βοηθητική δομή q_hash table

Ένα hash table που δείχνει σε μία λίστα από queries. Κάθε στοιχείο της λίστας δείχνει σε μια λίστα που αποθηκεύονται οι λέξεις.

Παράδειγμα αναζήτησης για ένα Document:
Για κάθε λέξη ελέγχουμε πρώτα αν την έχουμε ελέγξει είδη. Αν την έχουμε ελέγξει κανουμε skip αλλιώς ελέγχουμε στις 3 δομές με ποιες λέξεις “ταιριάζει”. Οι λέξεις που ταιριάζουν για κάθε δομή μπαίνουν σε μία λίστα και στο τέλος χρησιμοποιώντας τις λίστες αυτές και την βοηθητική δομή, βρίσκουμε ποια queries ταιριάζουν στο document.

Για να κάνουμε την αναζήτηση στις δομές hamming και edit, στα bk tree γίνεται μία επαναληπτική αναζήτηση απο 1 εως 3. Ο αριθμός αυτός αναφέρεται στο threshold που δίνουμε σαν ανοχή διαφοράς της λέξης που αναζητάμε με τις λέξεις στις δομές.

Παρακάτω παρουσιάζεται ο χρόνος εκτέλεσης του προγράμματος για διαφορετικά N. (το N αναφέρεται στο μέγεθος των hash tables των δομών)



Βλέπουμε ότι για N>100 έχουμε σταθερό χρόνο.

Multithreading

Με στόχο τη μείωση του συνολικού χρόνου εκτέλεσης έχει εισαχθεί παραλληλία. Η παραλληλία έχει εφαρμοστεί μέσω πολυνηματισμού.

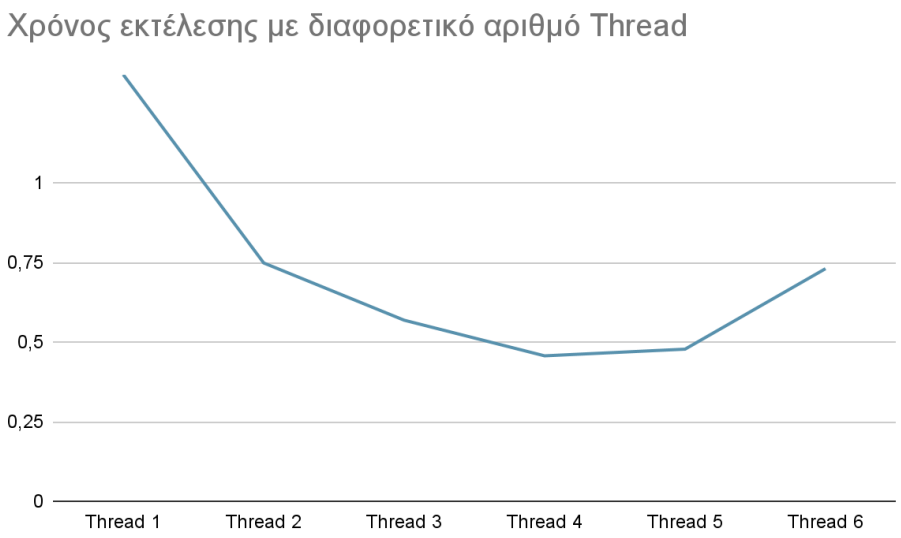
Πιο συγκεκριμένα έχουμε δημιουργήσει έναν Job Scheduler. Ο Job Scheduler αποτελείται από μία ουρά εργασιών, και ένα σύνολο από threads που ονομάζονται workers. Οι worker περιμένουν να εισαχθεί ένα job στην ουρά, και ένας ένας αναλαμβάνουν να εκτελέσουν τις εργασίες.

Στο πρόγραμμα μας χρησιμοποιούμε τον job scheduler για να κάνουμε παράλληλη την Match document. Κάθε φορά που διαβάζεται ένα Documet από το πρόγραμμα αντί να γίνεται η αναζήτηση τών λέξεων του στις δομές απο το main thread, δημιουργείται ένα job που μπαίνει στην ουρά, και αναλαμβάνουν οι thread worker να κάνουν την αναζήτηση.

Έτσι έχουμε σαν αποτέλεσμα να γίνονται οι αναζητήσεις πάνω στις δομές παράλληλα για πολλά documents και να μειώνεται ο χρόνος σημαντικά.

Τα αρχεία που τεστάραμε το πρόγραμμα μας έχουν εναλλάξ εισαγωγές query, διαβασμα document και διαγραφές query με διάφορους συνδυασμούς.
Ένα πρόβλημα που αντιμετωπίσαμε σε αυτό το σημείο, ήταν πως μετά από μια σειρά εισαγωγής και διαγραφής query που διαβάζαμε μια παρτίδα από document πρέπει το main thread να σταματήσει την εισαγωγή και διαγραφή από queries μεχρι τα documents να ολοκληρώσουν τις αναζητήσεις τους στις δομές.

Στο παρακάτω γράφημα φαίνεται ο χρόνος εκτέλεσης για διαφορετικό αριθμό thread που χρησιμοποιεί ο job scheduler



Παρατηρούμε οτι για 4 thread έχουμε τον βέλτιστο χρόνο που είναι 0,457 millisec.