ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Κ23α: Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα
Χειμερινό Εξάμηνο 2021 – 2022

Εργαστηριακή αναφορά Sigmod 2013

Βασιλική Ευσταθίου
Νίκος Ευτυχίου

Περιεχόμενα

1 Εισαγωγή **3**

2 Δομές Δεδομένων και Μέθοδοι Υλοποιήσης **4**

2.1 Δομή Αποθήκευσης Ερωτημάτων 4

2.2 Ευρετήρια Αναζήτησης 4

2.3 Διπλότυπες Λέξεις 6

2.4 Εύρεση ερωτημάτων για ένα κείμενο 6

2.5 Πολυνηματισμός 7

3 Πειραματική Αξιολόγηση **7**

3.1 Εκτέλεση Πειράματος 1 με αρχείο smalltest.txt 7

3.2 Εκτέλεση Πειράματος 2 με αρχειο input30m.txt 11

4 Αναφορές **15**

Περίληψη

Σε αυτή την αναφορά, στόχος μας είναι η ανάλυση και κατανόηση της εφαρμογής που υλοποιήσαμε για τον διαγωνισμό Sigmod 2013. Το ζητούμενο του διαγωνισμού ήταν η υλοποίηση μιας αντεστραμμένης μηχανής αναζήτησης, η οποία έχει ως επί το πλείστον στατικά ερωτήματα με τα οποία θα μπορεί να ταιριάζει τα διαθέσιμα κείμενα που έρχονται δυναμικά.

1 Εισαγωγή

Μια ανεστραμμένη μηχανή αναζήτησης λειτουργεί με τον ίδιο τρόπο που λειτουργεί και μια κανονική, αλλά προϋποθέτει ότι τα ερωτήματα που της δίνονται είναι ως επί το πλείστον στατικά, και τα διαθέσιμα κείμενα είναι αυτά που έρχονται δυναμικά. Για κάθε νέο κείμενο που φτάνει, η μηχανή αναζήτησης θα πρέπει να βρει όλα τα ερωτήματα που σχετίζονται με αυτό, όσο το δυνατόν γρηγορότερα. Για να θεωρήσουμε ότι ένα κείμενο απαντά σε ένα ερώτημα πρέπει το κείμενο να περιέχει λέξεις που ταιριάζουν με όλες τις λέξεις του ερωτήματος.

Τα ερωτήματα είναι χωρισμένα σε τρεις κατηγορίες:

1. Exact matching όπου όλες οι λέξεις του ερωτήματος θα πρέπει να είναι απολύτως ίδιες με λέξεις του κειμένου.
2. Edit distance όπου όλες οι λέξεις του ερωτήματος θα μπορούν να έχουν μια μέγιστη διαφορά με λέξεις του κειμένου (Η διαφορά θα ορίζεται με την edit διαφορα τους)
3. Hamming distance όπου όλες οι λέξεις του ερωτήματος θα μπορούν να έχουν μια μέγιστη διαφορά με λέξεις του κειμένου (Η διαφορά θα ορίζεται με την hamming διαφορα τους)

Σκοπός μας είναι να υλοποιήσουμε την εν λόγω μηχανή και να μειώσουμε όσο το δυνατόν περισσότερο τον χρόνο απόκρισης της στην εύρεση ερωτημάτων.

2 Δομές Δεδομένων και Μέθοδοι Υλοποιήσης

2.1 Δομή Αποθήκευσης Ερωτημάτων

Όλα τα ερωτήματα αποθηκεύονται σε ένα HashTable (Map of Queries) πριν εισαχθούν στο ευρετήριο αναζήτησης για την καλύτερη διαχείρηση τους. ;;;;

2.2 Ευρετήρια Αναζήτησης

Θα χρησιμοποιήσουμε τρία ευρετήρια αναζήτησης για τη μεθοδολογική αποθήκευση των δεδομένων, με στόχο την ταχεία ανάκτηση πληροφοριών. Τα τρία αυτά ευρετήρια είναι:

- Index-Exact
- Index-Edit
- Index-Hamming

Χωρίς τη χρήση των πιο πάνω ευρετηρίων, η μηχανή αναζήτησης θα έπρεπε να ελέγξει κάθε λέξη από το σύνολο των ερωτημάτων του συστήματος, έτσι ώστε να εντοπίσει τα ερωτήματα τα οποία απαντούν ένα κείμενο.

2.2.1 Index Exact

Το συγκεκριμένο ευρετήριο επιλέξαμε να το υλοποιήσουμε με τη δομή hash-table. Με αυτή τη δομή μπορούμε να έλέγχουμε αν μια λέξη ταιριάζει ακριβώς με μια άλλη σε χρόνο O(1).

2.2.2 Index Edit

Το συγκεκριμένο ευρετήριο επιλέξαμε να το υλοποιήσουμε με τη δομή BK-Tree. Με αυτή την εξειδικεύμενη δομή δέντρων μπορούμε εύκολα να αποκλείσουμε πολλές λέξεις που δεν έχουν την επιθυμητή edit απόσταση μεταξύ τους. Επιπρόσθετα, επιλέξαμε όπως και με το ευρετήριο Index Hamming να υλοποιήσουμε το εν λόγω ευρετήριο με την χρήση ενός πίνακα μεγέθους 28, όπου σε κάθε του θέση θα αποθηκεύεται ένα BK-Tree για κάθε διαφορετικό μήκος λέξεως (4-31 γράμματα το ελάχιστο και το μέγιστο μήκος μιας λέξης, αντίστοιχα). Έτσι μπορούμε να αποφύγουμε αρκετούς επιπρόσθετους ελέγχους. Επίσης με αυτή τη δομή μπορούμε να βρούμε όλες τις λέξεις με την επιθυμητη απόσταση σε χρόνο O(log(n)) όπου n ο αριθμός των κόμβων στο δέντρο.

2.2.3 Index Hamming

Λόγω της υλοποιήσης της hamming distance (δηλαδή ότι μπορεί να υπολογίσει απόσταση μόνο για λέξεις ιδίου μήκους) επιλέξαμε να υλοποιήσουμε το εν λόγω ευρετήριο με την χρήση ενός πίνακα μεγέθους 28, όπου σε κάθε του θέση θα αποθηκεύεται ένα BK-Tree για κάθε διαφορετικό μήκος λέξεως (4-31 γράμματα το ελάχιστο και το μέγιστο μήκος μιας λέξης, αντίστοιχα).

Δομική Οντότητα Ευρετηρίων

Η δομική οντότητα των ευρετηρίων είναι το entry και περιλαμβάνει δύο πεδία, το word και το payload. Το word αναπαριστά την λέξη, πάνω στην οποία χτίστηκε το entry και το payload είναι μια λίστα, η οποία κρατάει τα ερωτήματα που περιέχουν την λέξη word.

Ένα entry δημιουργείται σε κάθε ευρετήριο όταν έρχεται μια καινούργια λέξη σε αυτό. Έτσι τα entries είναι μοναδικά σε κάθε ευρετήριο, αφού για κάθε λέξη αντιστοιχίζεται μόνο ένα entry.

2.3 Διπλότυπες Λέξεις

Τα κείμενα περιέχουν μεγάλο πλήθος λέξεων, έτσι είναι εξαιρετικά απίθανο να μην υπάρχουν πολλές ίδιες λέξεις σε ένα κείμενο. Για να αποφύγουμε τον έλεγχο αυτών τον διπλότυπων λέξεων δημιουργήσαμε την συνάρτηση Deduplicate Words η οποία επιστρέφει το κείμενο χωρίς διπλότυπες λέξεις.
Η υλοποίηση της Deduplicate Words έγινε με δύο διαφορετικούς τρόπους

2.3.1 Deduplicate Words Map

Η συνάρτηση υλοποιήθηκε με τη χρήση HashTable και συνδεδεμένης λίστας. Συγκεκριμένα στο HashTable και στη λίστα κρατάμε όλες τις μοναδικές λέξεις του κειμένου. Χρησιμοποιούμε το HashTable γιατί μπορούμε να βρούμε αν μια λέξη είναι διπλότυπη σε χρόνο O(1). Το μειονέκτημα σε αυτή την Deduplcate Words είναι περισσότερη δέσμευση μνήμης που χρειάζεται σε σχέση με την Deduplicate Words List.

2.3.2 Deduplicate Words list

Η συνάρτηση υλοποιήθηκε με τη χρήση συνδεδεμένης λίστας. Συγκεκριμένα στη λίστα κρατάμε όλες τις μοναδικές λέξεις του κειμένου. Χρησιμοποιούμε τη λίστα γιατί χρειάζεται λιγότερη μνήμη σε σχέση με την Deduplicate Words Map με το μειονέκτημα να είναι χρόνος εκτέκεσης της (O(n)).

2.4 Εύρεση ερωτημάτων για ένα κείμενο

Για να βρούμε τα ερωτήματα τα οποία απαντάνε σε ένα κείμενο εκτός από τα ευρετήρια αναζήτησης χρειαζόμαστε μια επιπλέον δομή, τη δομή Special. Η συγκεκριμενη δομή αποθηκευεται σε ένα HashTable ώστε να μπορούμε να έχουμε πρόσβαση σε κάθε κόμβο της άμεσα. Κάθε κόμβος αποθηκεύει ένα μοναδικό ερώτημα και μια λίστα με τις λέξεις τις οποίες έχει κοινές αυτό το

ερώτημα με το κείμενο που ψάχνουμε. Άν ένα ερώτημα έχει όλες τις λέξειες του στη λίστα τότε ικανοποιεί το κείμενο.

2.5 Πολυνηματισμός

Με τη χρήση πολυνηματισμού καταφέραμε να παραλληλοποιήσουμε δύο πολύ βασικές διεργασίες. Την εισαγωγή ερωτημάτων στα ευρετήρια (StartQuery) και την εύρεση ερωτημάτων για τα κείμενα (MatchDocument).

3 Πειραματική Αξιολόγηση

3.1 Εκτέλεση Πειράματος 1 με αρχείο smalltest.txt

Το πείραμα εκτελέστηκε με τις 4 παρακάτω διαφορετικές υλοποιήσεις.

α΄) Χρήση πολυνηματισμού στην StartQuery και στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα (Not optimal) .

β΄) Χρήση πολυνηματισμού στην StartQuery και στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα και HashTable (Optimal) .

γ΄) Χρήση πολυνηματισμού μόνο στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα (Not optimal) .

δ΄) Χρήση πολυνηματισμού μόνο στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα και HashTable (Optimal) .

Για κάθε μια από τις παραπάνω εφαρμογές του πειράματος μετρήσαμε τον χρόνο λειτουργίας τους για 1 εως 7 threads, καθώς επίσης και τη μνήμη που χρησιμοποιήσαν. Στη συνέχεια συγκρίναμε αυτές τις τιμές μεταξύ τους για να βρούμε ποια υλοποίηση είναι η αποδοτικότερη.

Για τα πειράματα είχαμε στη διάθεση μας τον ακόλουθο υπολογιστή:

- OS: Ubuntu 18.04.6 LTS GNU/Linux
- Cores: 4
- Threads: 8
- CPU: Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz
- RAM: 16 GB

3.1.1 Επεξήγηση Γραφικών Παραστάσεων

Στο σχήμα 1 στον κάθετο άξονα βλέπουμε τον συνολικό χρόνο εκτέλεσης του προγράμματος και στον οριζόντιο άξονα τον αριθμό των threads που χρησιμοποιήθηκαν κάθε φορά που εκτελείτουν το πρόγραμμα. Σε κάθε γραφική παράσταση χρησιμοποιήσαμε τις εξείς υλοποιήσεις:

α′) Χρήση πολυνηματισμού στην StartQuery και στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα (Not optimal) .

β′) Χρήση πολυνηματισμού στην StartQuery και στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα και HashTable (Optimal) .

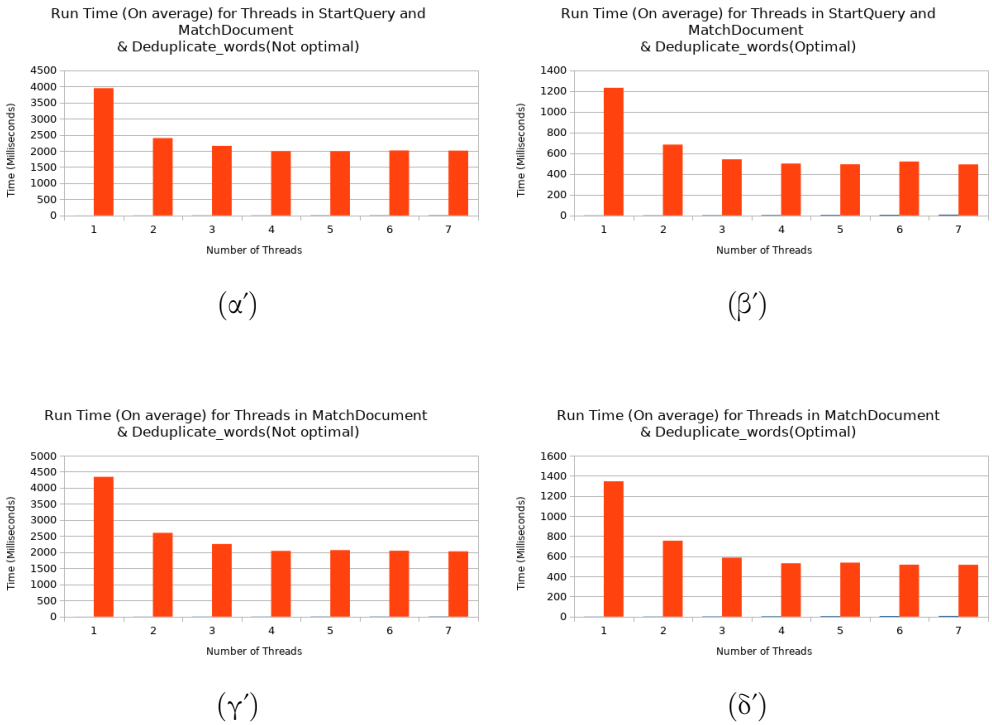
γ′) Χρήση πολυνηματισμού μόνο στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα (Not optimal) .

δ′) Χρήση πολυνηματισμού μόνο στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα και HashTable (Optimal) .

3.1.2 Παρατηρήσεις

Παρατηρούμε ότι ο ταχύτερος χρόνος γίνεται όταν υπάρχει πολυνηματισμός, τόσο στην StartQuery όσο και στην MatchDocument, καθώς και όταν η Dedupli-

cate Words είναι optimal (γραφική β΄). Η αμέσως επόμενη ταχύτερη υλοποίηση είναι αυτή με πολυνηματισμό μόνο στην MatchDocument και με optimal Deduplicate Words (γραφική δ΄) Τρίτη ταχύτερη με αρκετά μεγάλη διαφορά σε σχέση με τις δύο πρώτες είναι η υλοποίηση με τη not optimal Deduplicate Words και με πολυνηματισμό στην StartQuery και στην MatchDocument (γραφική α΄). Τέλος, χείριστη είναι η υλοποίηση με πολυνηματισμό μόνο στην MatchDocument και με not optimal Deduplicate Words (γραφική γ΄).
Συγκρίνοντας την γραφική α΄ με την γραφική γ΄ και την γραφική β΄ με την γραφική δ΄ παρατηρούμε πως οι διαφορές τους στον χρόνο εκτέλεσης είναι ελάχιστες, ενώ αν συγκρίνουμε την γραφική α΄ με την β΄ και την γραφική γ΄ με την δ΄ παρατηρούμε πως οι διαφορές τους στον χρόνο εκτέλεσης είναι τεράστιες.
Επίσης οι υλοποιήσεις με ίδια Deduplicate Words δεν έχουν μεγάλη απόκλειση στο χρόνο εκτέλεσης τους για 1 thread. Τέλος, σε κάθε γραφική παράσταση παρατηρούμε πως η διαφορά χρόνου όταν χρησιμοποιούμε 4 threads και άνω είναι ελάχιστη.



(α΄) (β΄)

(γ΄) (δ΄)

Σχήμα 1: Οι μετρήσεις έγιναν με το αρχείο small-test.txt

Από το σχήμα 2 παρατηρούμε ότι η Deduplicate Words Map χρειάζεται περισσότερη μνήμη, όπως αναμέναμε. Ωστόσο η διαφορά της Deduplicate Words Map με της Deduplicate Words List δεν είναι τόσο μεγάλη, έτσι ώστε να επιβαρύνει το πρόγραμμα. Επίσης, παρατηρούμε ότι η χρήση πολυνηματισμού και στην StartQuery και στην MatchDocument απαιτεί ελαχιστα περισσοτερη μνήμη σε σχέση με την χρήση πολυνηματισμού μόνο στην MatchDocument.

(α΄)

Σχήμα 2: Οι μετρήσεις έγιναν με το αρχείο small-test.txt

3.1.3 Συμπεράσματα

Με τις παραπάνω παρατηρήσεις καταλήγουμε στο εξής συμπέρασμα ότι η χρήση του πολυνηματισμού στην StartQuery βελτιστοποιεί ελάχιστα το πρόγραμμα μας σε αντίθεση με την χρήση της optimal Deduplicate Words, η οποία το βελτιώνει αισθητά. Επίσης, συμπεράναμε πως ο ιδανικός αριθμός threads που μπορεί να χρησιμοποιηθεί στο πρόγραμμα είναι τουλάχιστον όσος και ο αριθμός των threads που διαθέτει ο υπολογιστής.

3.2 Εκτέλεση Πειράματος 2 με αρχειο input30m.txt

Το πείραμα εκτελέστηκε με τις 2 παρακάτω διαφορετικές υλοποιήσεις.

α΄) Χρήση πολυνηματισμού στην StartQuery και στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα και HashTable

(Optimal) .

β′) Χρήση πολυνηματισμού μόνο στην MatchDocument και υλοποιήση της Deduplicate Words με συνδεδεμένη λίστα και HashTable (Optimal) .

Για κάθε μια από τις παραπάνω εφαρμογές του πειράματος μετρήσαμε τον χρόνο λειτουργίας τους για 1 εως 7 threads, καθώς επίσης και τη μνήμη που χρησιμοποιήσαν. Στη συνέχεια συγκρίναμε αυτές τις τιμές μεταξύ τους για να βρούμε ποια υλοποίηση είναι η αποδοτικότερη.

Για τα πειράματα είχαμε στη διάθεση μας τον ακόλουθο υπολογιστή:

- OS: Ubuntu 18.04.6 LTS GNU/Linux
- Cores: 4
- Threads: 8
- CPU: Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz
- RAM: 16 GB

3.2.1 Επεξήγηση Γραφικών Παραστάσεων

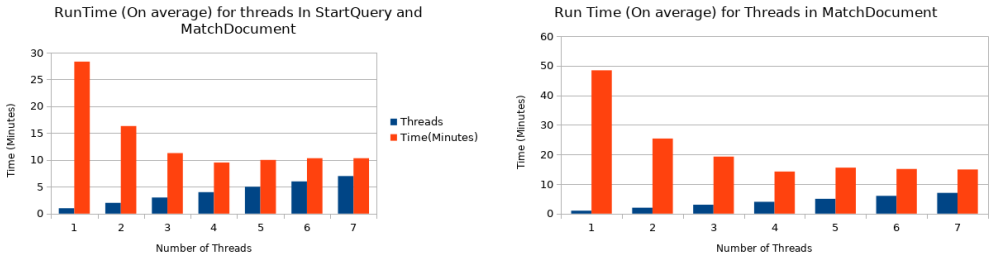
Στο σχήμα 3 στον κάθετο άξονα βλέπουμε τον συνολικό χρόνο εκτέλεσης του προγράμματος και στον οριζόντιο άξονα τον αριθμό των threads που χρησιμοποιήθηκαν κάθε φορά που εκτελείτουν το πρόγραμμα. Σε κάθε γραφική παράσταση χρησιμοποιήσαμε τις εξείς υλοποιήσεις:

α′) Χρήση πολυνηματισμού στην StartQuery και στην MatchDocument.

β′) Χρήση πολυνηματισμού μόνο στην MatchDocument.

3.2.2 Παρατηρήσεις

Παρατηρούμε ότι ο ταχύτερος χρόνος γίνεται όταν υπάρχει πολυνηματισμός, τόσο στην StartQuery όσο και στην MatchDocument(γραφική β΄). Όπως αναμέναμε η διαφορά στον χρόνο εκτέλεσης των δύο υλοποιήσεων είναι αισθητή. Αυτό οφείλεται στον μεγάλο όγκο ερωτημάτων που υπάρχουν στο αρχείο.
Τέλος, σε κάθε γραφική παράσταση παρατηρούμε πως η διαφορά χρόνου όταν χρησιμοποιούμε 4 threads και άνω είναι ελάχιστη.



(α΄) (β΄)

Σχήμα 3: Οι μετρήσεις έγιναν με το αρχείο input300m.txt

(α′)

Σχήμα 4: Οι μετρήσεις έγιναν με το αρχείο input300m.txt

3.2.3 Συμπεράσματα

Με τις παραπάνω παρατηρήσεις καταλήγουμε στο εξής συμπέρασμα.Η χρήση του πολυνηματισμού στην StartQuery μειώνει αισθητά το χρόνο εκτέκεσης σε αντίθεση με το πρώτο πείραματα το οποίο βελτιστοποιούσε ελάχιστα. Επισης η χρηση πολυνηματισμου μονο για την MatchDocument χρειαζεται ελαχιστα λιγοτερη μη σε σχεση με την υλοποιηση πολυνηματισμου και στην StartQuery και στην MatchDocument

4 Αναφορές

https://sigmod.kaust.edu.sa/task-details.html

https://signal-to-noise.xyz/post/bk-tree/

W.BurkhardandR.Keller.Someapproachestobest-matchfilesearching,CACM, 1973

https://en.wikipedia.org/wiki/Hamming_distance

https://www.lemoda.net/c/levenshtein/