Τελική Αναφορά

Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

2021-2022

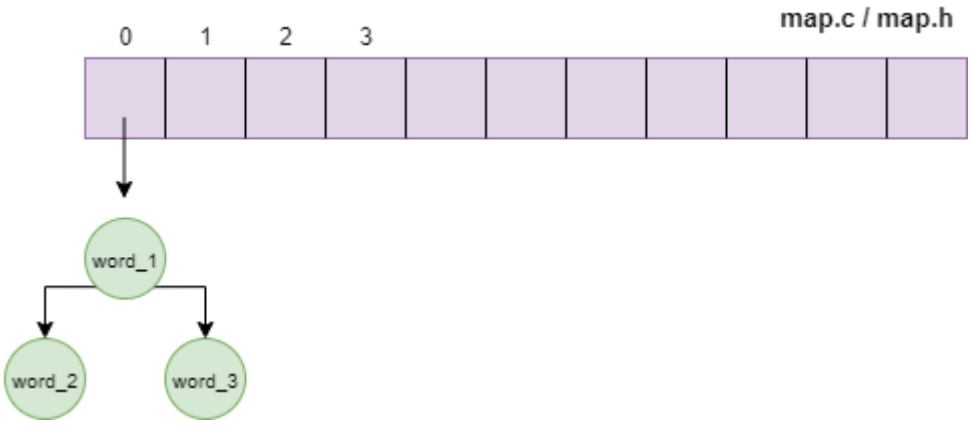
Κωστόπουλος Σταύρος - 1115201700068

Κοκκίνης Γεώργιος -1115201700050

Μπέρος Δημήτριος -1115201600269

Deduplication

Για την απαλοιφή διπλότυπων λέξεων από το εκάστοτε κείμενο, επιλέξαμε μία δομή ενός πίνακα κατακερματισμού (/modules/hash/map.c) , όπου κάθε κελί του αναπαρίσταται από ένα Binary Search Tree (/modules/trees/bst.c). Κάθε φορά που θέλουμε να διαπιστώσουμε αν υπάρχει ήδη μία λέξη, μας καθοδηγεί η hash function στο κατάλληλο κελί του hashtable (O(1)) και ψάχνουμε στο BST του για να δούμε αν υπάρχει αυτή η λέξη (O(logn)). Το BST επιλέχθηκε γιατί είναι πιο αποδοτικό από μια απλή υλοποίηση hashtable bucket, όπου το complexity αναζήτησης θα ήταν O(n), παρόλο που το insertion complexity του θα ήταν O(1). Παρατηρήθηκε ότι τα κείμενα αποτελούνται από πολλές διπλότυπες λέξεις, για αυτό επιλέξαμε να δώσουμε μεγαλύτερη βαρύτητα στο Search και όχι στο Insert operation.



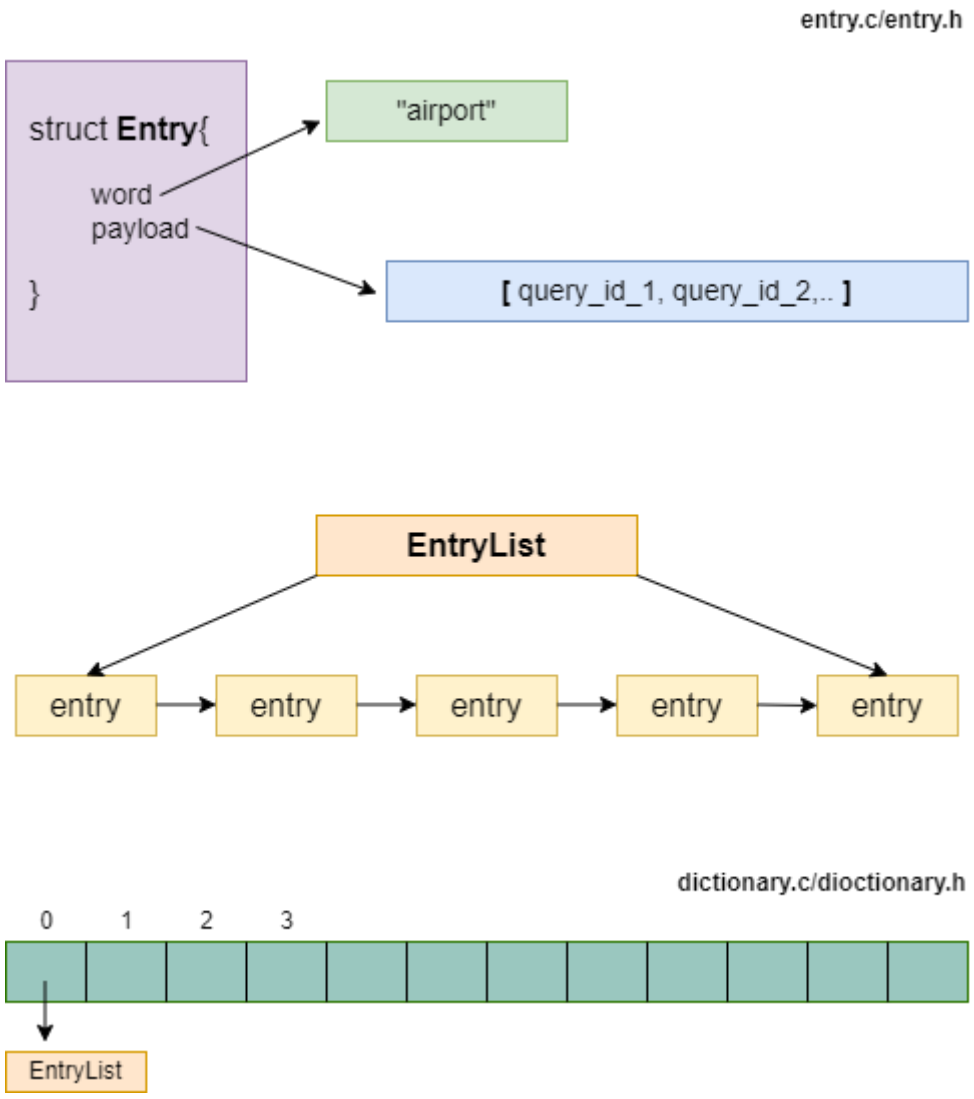
Queries

Για την αποθήκευση των ενεργών queries επιλέξαμε την δομή Dictionary(/modules/hash/dictionary.c), που πρόκειται για μία δομή hashtable, όπου κάθε κελί του αναπαρίσταται από μία λίστα από entries (EntryList - /modules/entries.entry.c). Η δομή ενός Entry είναι υλοποιημένη ακριβώς όπως

έχει ζητηθεί στην εκφώνηση της εργασίας. Αποτελείται από το πεδίο **word**, που είναι μία λέξη/μία σειρά από χαρακτήρες, και το πεδίο **payload,** που είναι μία λίστα με τα id των queries στα οποία εμφανίζεται η εκάστοτε λέξη.

Αν μία λέξη έχει εμφανιστεί ήδη ως λέξη κάποιου άλλου query, τότε βρίσκουμε το Entry της λέξης αυτής, από το EntryList που μας οδήγησε η hash function και απλώς προσθέτουμε το νέο query_id στο payload της λέξης αυτής. (O(n))

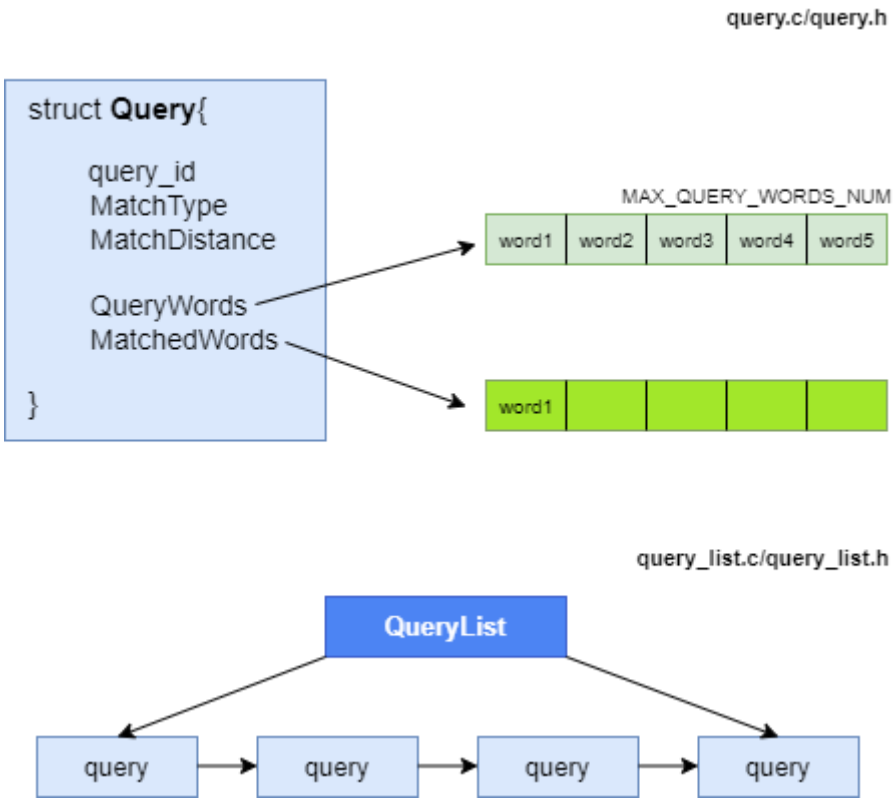
Υπάρχει 1 dictionary για κάθε MatchType (Edit,Hamming,Exact).

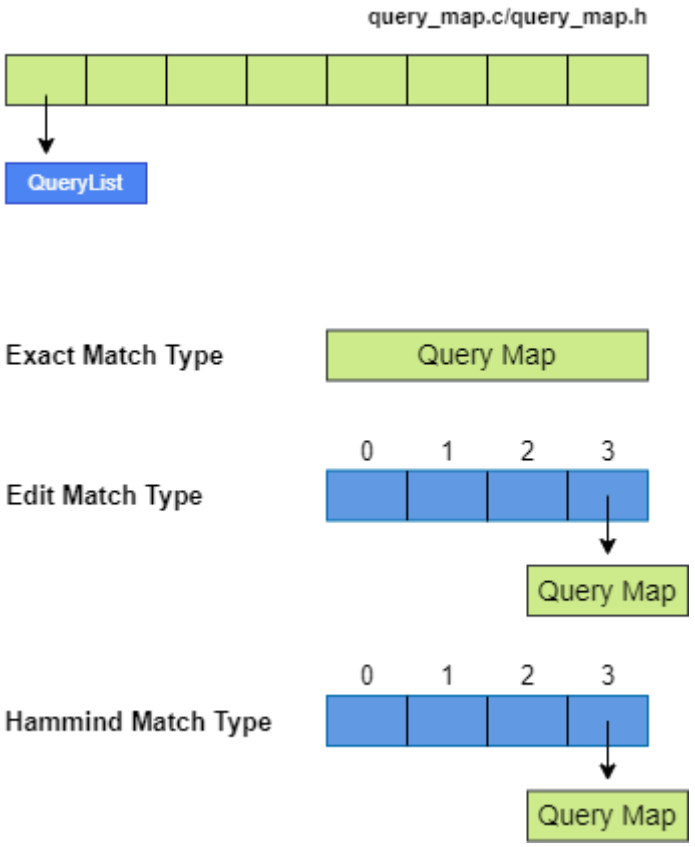


Οι επόμενες σχεδιαστικές επιλογές που έπρεπε να πάρουμε ήταν για την υλοποίηση της MatchDocument() διαδικασίας.Για το σκοπό αυτό, έπρεπε αρχικά να δημιουργήσουμε ακόμα μία δομή, την χρησιμότητα της οποίας, θα αναλύσουμε αργότερα όταν περιγράψουμε την διαδικασία.

Η δομή αυτή πρόκειται για ένα hashtable όπου χρησιμοποιώντας το query_id κατανέμει queries στα κελιά του προσθέτοντάς τα σε μία query list.(/modules/hash/query_map.c)

Χρησιμοποιήσαμε αυτή την δομή για να οργανώσουμε τα queries σύμφωνα με το MatchType (Exact/Hamming/Edit) και το MatchDistance τους (0/1/2/3). Επομένως, καταλήξαμε σε έναν πίνακα με 4 query maps (1 ανά threshold) για τα Edit Distance queries, άλλον έναν ίδιο πίνακα για τα Hamming Distance queries, και ένα query map για τα Exact Match Type queries(καθώς έχουν μονάχα ένα πιθανό threshold, το 0).





MatchDocument()

Για την διαδικασία MatchDocument() καταλήξαμε ύστερα από διαφορετικές υλοποιήσεις και δοκιμές, στον παρακάτω αλγόριθμο:

Για Edit MatchType:

1. Για κάθε πιθανό threshold (0/1/2/3)

2. Για κάθε Entry στο Edit MatchType dictionary

3. lookup( ) για την λέξη του Entry

4. Αν υπάρχουν results, τότε υπάρχουν λέξεις του κειμένου που matchάρουν για αυτό το MatchType και αυτό το threshold, και αυτή τη λέξη
5. Διανύοντας το κατάλληλο query map(για αυτό το MatchType και αυτό το threshold) πάρε τα query_id και δες ποια υπάρχουν στο payload του Entry αυτού
6. Αν υπάρχει κάποιο query id στο payload του Entry, συμπλήρωσε την λέξη αυτή στο MatchedWords array
7. Αν έγιναν match όλες οι λέξεις αυτού του query (matched_words_num==query_words_num), τότε πρόσθεση το query στα results του Document

Για Hamming MatchType:

Η ίδια νοοτροπία με την παραπάνω, μόνο που η lookup() θα εκτελεστεί στο κατάλληλο word length BK-tree. Όπως ζητήθηκε στην εκφώνηση, έχει υλοποιηθεί ένα Hamming Distance BK-tree για κάθε πιθανό word length.

Για Exact MatchType:

Η ίδια νοοτροπία, μόνο που εκτελείται μόνο για threshold==0 η διαδικασία, χρησιμοποιώντας ως ευρετήριο το map των deduplicated document words, και αντί για lookup, χρησιμοποιούμε την map_find() στο hashtable, για κάθε λέξη του Exact MatchType dictionary.

Παραλληλοποίηση/Multi-threading

Για τη διεκπεραίωση της 3ης εργασίας, έπρεπε να παραλληλοποιήσουμε κάποια/ες λειτουργίες του προγράμματός μας, χρησιμοποιώντας νήματα ώστε αυτό να γίνει ταχύτερο/αποδοτικότερο.

Έπειτα από δοκιμές, οι οποίες θα συζητηθούν αργότερα, καταλήξαμε στην παραλληλοποίηση ολόκληρης της διαδικασίας MatchDocument(). Με άλλα λόγια, σκεφτήκαμε η διαδικασία να εκτελείται από πολλά νήματα, για πολλά documents, την ίδια στιγμή.

Το κύριο νήμα, συντηρεί και προστατεύει απαραίτητες πληροφορίες(π.χ. active queries πριν εμφανιστεί ένα document), και αντιγράφοντάς-μετατρέποντάς τες σε εργασίες/jobs τις μεταφέρει στα νήματα, όπου αυτά με τη σειρά τους θα εκτελέσουν την εργασία.

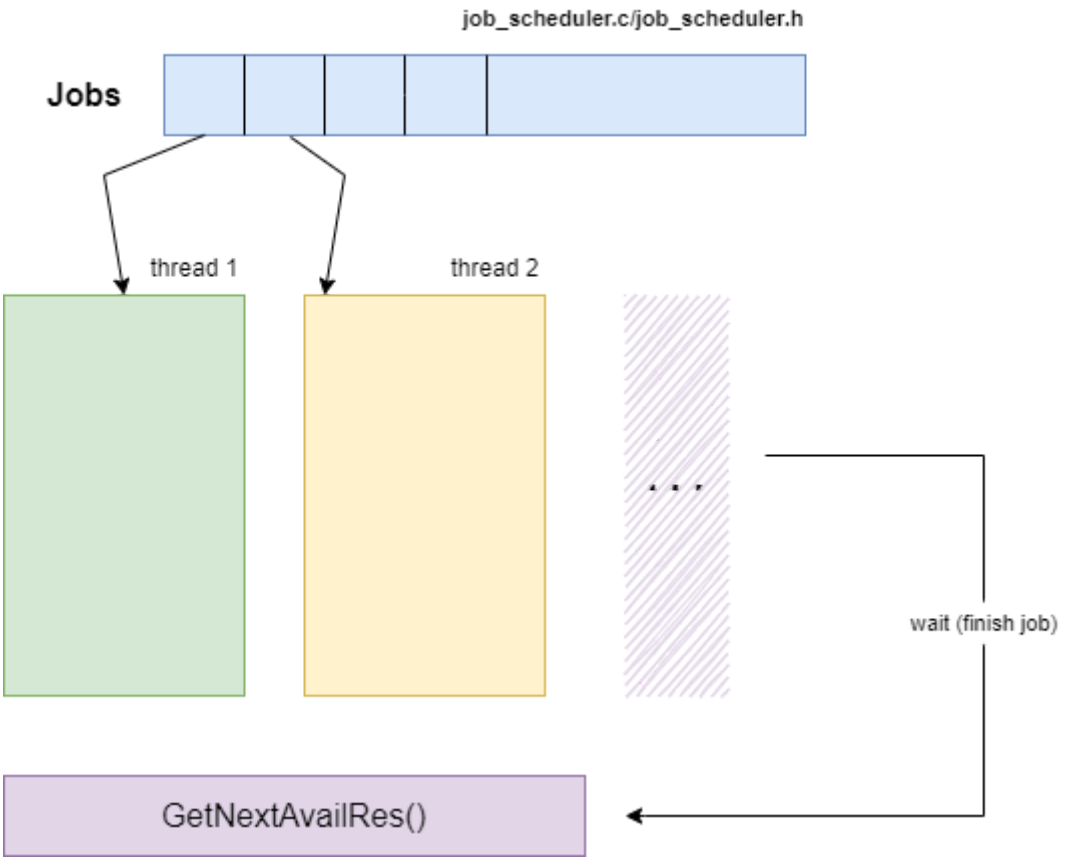
Όπως ζητήθηκε στην εκφώνηση, υλοποιήθηκε ένας Job Scheduler ο οποίος εκτός από άλλες απαραίτητες πληροφορίες(π.χ. mutexes, barriers), συγκρατεί και το First-In-First-Out (FIFO) jobs queue. Η παραγωγή και η άντληση εργασιών από την ουρά αυτή, αντιμετωπίστηκε ως ένα producer/consumer problem, και έχει συντονιστεί επιτυχώς, χωρίς busy-waiting με την χρήση mutex και semaphore.

Υπάρχουν 3 είδη εργασιών/jobs στην υλοποίησή μας, τα οποία μπορούν αν προστεθούν στην ουρά:

-**Document job:** Ένα document job πρόκειται για ένα πακέτο με όλες τις απαραίτητες πληροφορίες που χρειάζεται ένα νήμα για να εκτελέσει την MatchDocument() και να παράγει αποτελέσματα για ένα document.

-**Finish job:** Τα finish jobs εξυπηρετούν τον συντονισμό των threads για την ορθή εκτέλεση της GetNextAvailRes() διαδικασίας.Με τον τρόπο που χρησιμοποιείται η διαδικασία αυτή από την δοσμένη main function, απαιτείται όλα τα νήματα να έχουν τελειώσει την παραγωγή αποτελεσμάτων για όλα τα documents που τους έχουν ανατεθεί πριν από την κλήση της GetNextAvailRes(). Τα finish jobs αποτελούν έναυσμα για τον συντονισμό αυτών καθώς αφού προστεθούν στην ουρά, το κάθε νήμα και η GetNextAvailRes() διαδικασία, θα τεθούν σε αναμονή(thread barrier), μέχρι να τελειώσουν όλες οι MatchDocument().

-**End job:** Τα end jobs ανατίθενται στα threads όταν επιθυμούμε να τερματίσουν. Όταν ένα thread αναλάβει ένα end job, σταματάει να ακούει από την ουρά εργασιών και τερματίζει.



Μία άλλη υλοποίηση που δοκιμάστηκε και απορρίφθηκε, ήταν αυτή της παραλληλοποίησης των διαδικασιών αναζήτησης, για κάθε MatchType, που εκτελούνται εντός της MatchDocument. Η υλοποίηση αυτή δεν προσέφερε καλύτερους χρόνους σε σχέση με αυτούς της 2ης εργασίας, άρα δεν είχε νόημα ο παραλληλισμός. Για τον λόγο αυτό απορρίφθηκε.Στα GitHub pull merges της 3ης εργασίας, μπορούν να φανούν οι διαδικασίες υλοποίησης.