Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

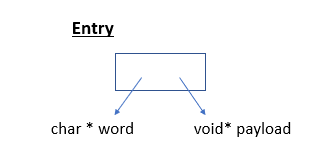
|  |  |
| --- | --- |
| Ονοματεπώνυμο | Αριθμός Μητρώου |
| Ευάγγελος Φιαμπόλης | 1115201800205 |
| Γεώργιος Γαλάνης | 1115201800024 |
| Γρηγόριος Γαλανάκης | 1115201600026 |

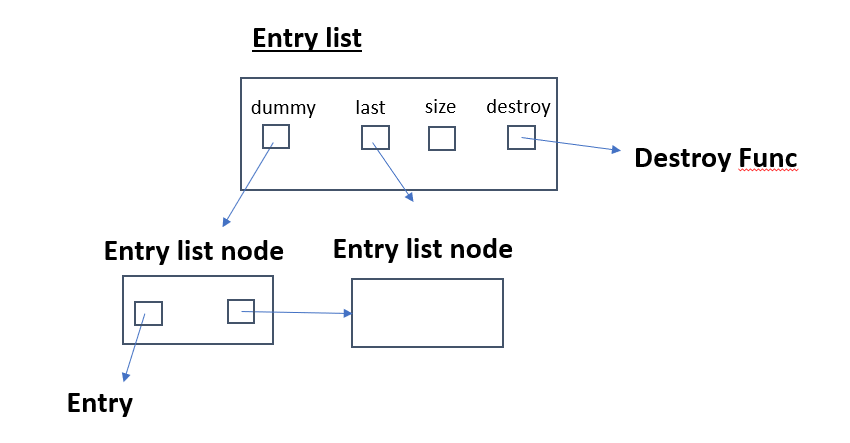
*Θέμα*: Ανεστραμμένη Μηχανή Αναζήτησης (inverted search engine)

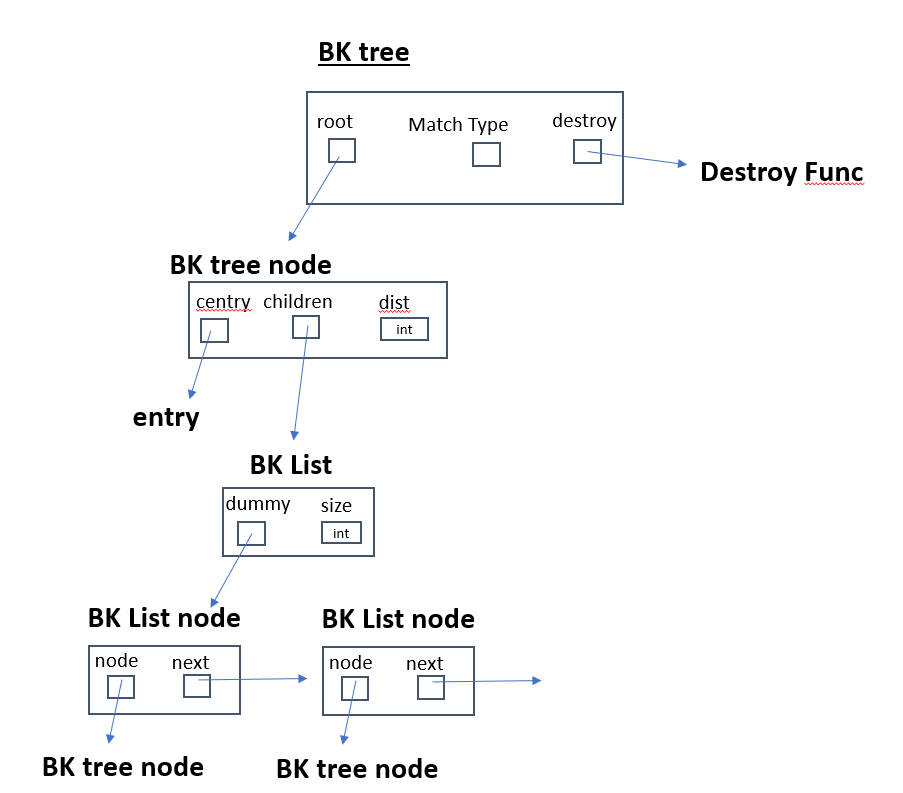
Τελική Αναφορά:

# 1ο Μέρος:

Δομές Δεδομένων:





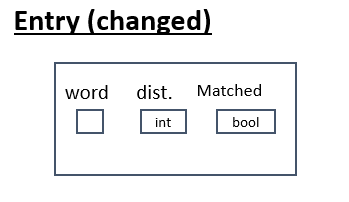


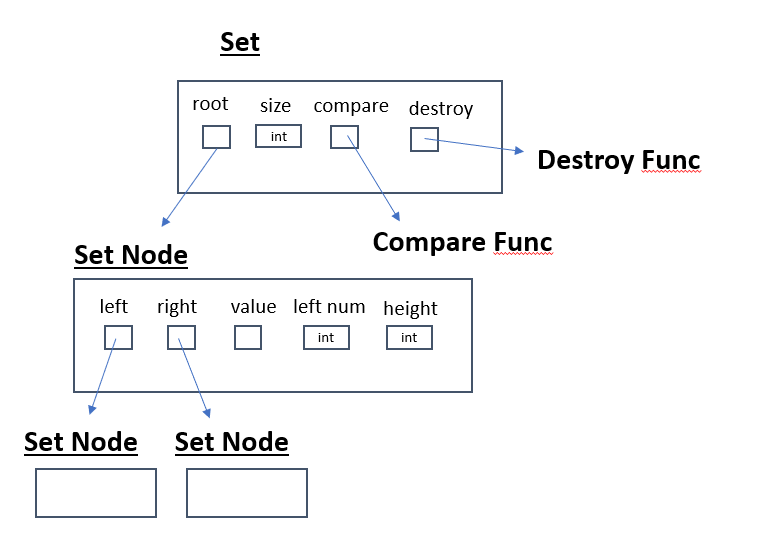
Περιγραφή:

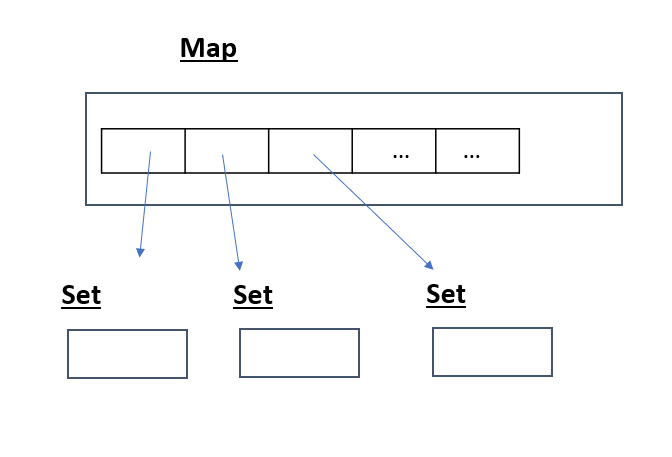
Το πρόγραμμα διαβάζει από το ένα txt αρχείο ώστε να παρθούν κάποια words, να εισαχθούν στις δομές(Entry List, BK tree) και να γίνουν τα απαραίτητα test για το 1ο μέρος της εργασίας.

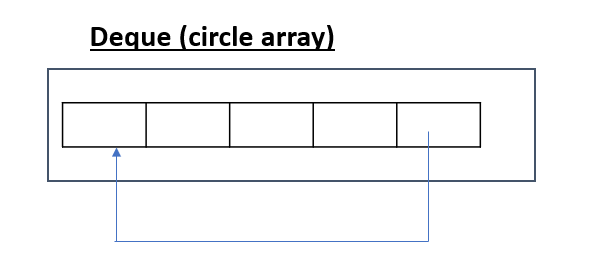
# 2ο Μέρος:

Δομές Δεδομένων:









Περιγραφή:

Έχουν χρησιμοποιηθεί 3 δομές, μία για κάθε Match Type (exact, hamming, edit)

1. Exact:

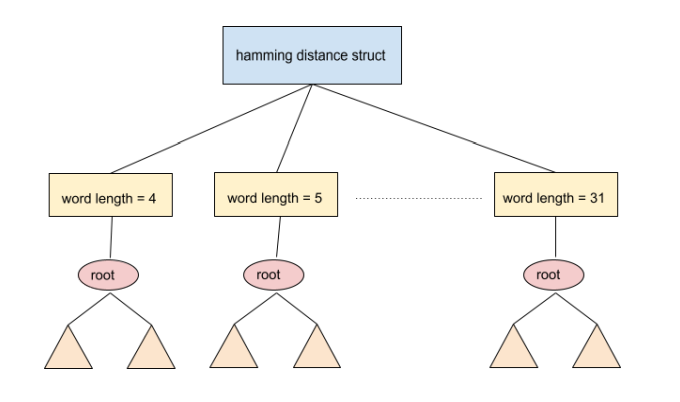
[HashMap](#Map) με [Set](#Set) (hash function [djb2](http://www.cse.yorku.ca/~oz/hash.html))

1. Edit:

[BK tree](#BKtree) struct

1. Hamming:

[BK tree](#BKtree)[28]. Array of BK trees of size 28, 1 struct for each word length (from 4 to 31 = 28).



Επίσης έχουν υλοποιηθεί:

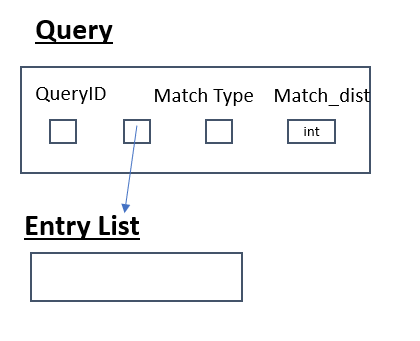
* Δομή [Set](#Set) για την αποθήκευση των Queries
* Δομή [Deque](#Deque) για την αποθήκευση των Documents
* Ένα [Entry List](#EntryList) για να κρατάει το result list των αποτελεσμάτων των query id’s που έκαναν match με το document.

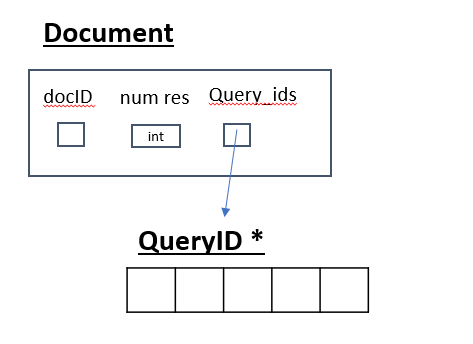
**Deduplication στα entries των δομών**

* Για τα διπλότυπα, στα δέντρα όταν γίνεται insert ένα καινούργιο entry του οποίου το word ήδη υπάρχει μέσα στο εκάστοτε δέντρο, τότε το δέντρο απλώς διατηρεί το παλιό entry και αγνοεί το καινούργιο. Για την exact\_match, όμως, το map και το set σε περίπτωση duplications συγκρατούν το καινούργιο στοιχείο και διαγράφουν το παλιό, επομένως πρώτα θα εφαρμόζουμε map\_find(O(1)) για να δούμε αν υπάρχει ήδη ίδιο word μέσα στη δομή, και ύστερα αν δεν υπάρχει εισάγουμε το καινούργιο entry στη δομή.

**Deduplication στις λέξεις των Documents**

* Δομή [Map](#Map) για τις λέξεις του κάθε Document ώστε να μην ελέγχεται μια λέξη που εμφανίζεται πολλές φορές μες το Document πολλές φορές.
* Αναζήτηση: O(1) για την εύρεση του κελιού και O(logn) για την εύρεση της λέξης στο Set, απλώς αν έχουν χωριστεί καλά οι λέξεις μέσα στον πίνακα τότε το set του κάθε κελιού θα είναι μικρό άρα θα "αγγίζει" χρόνους Ο(1)
* Για το Query έχει προστεθεί μια λίστα [Entry List](#EntryList) που κρατάει τα entries στα οποία αναφέρεται





* Δομή Deque για να κρατάει προσωρινά τα ids των queries που έκαναν match μέσα στο Document (μεταβλητή matched = true μέσα στο entry) και ύστερα να τα αποθηκεύει στη λίστα query\_ids του struct Document.

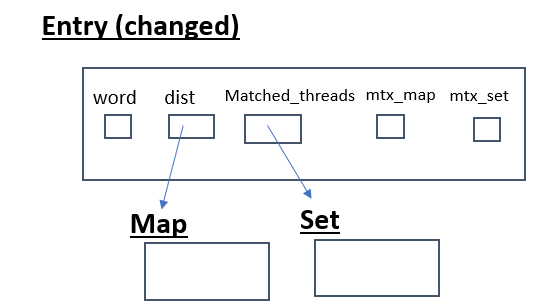
# 3ο Μέρος: Multithreading

Τρόπος μεταγλώττισης και εκτέλεσης (εντός του directory final-testdriver):

1. ***make run***
2. ***make valgrind (έλεγχος για memory leaks)***

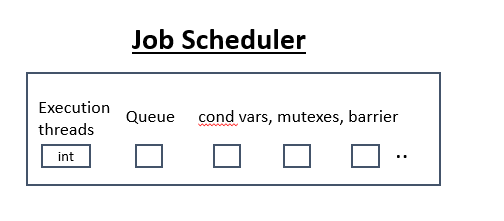
*Σημείωση*: Τα tests(συγκεκριμένα το entry\_test.c) που είχαν δημιουργηθεί για τις δομές όπως είχαν υλοποιηθεί στο 2ο μέρος της εργασίας δεν θα τρέξουν διότι η υλοποίηση του 3ου μέρους είναι πολυνηματική και λειτουργούν για ένα μόνο νήμα.

**Νέα σχεδίαση δομής entry:**



* Όταν το entry κάνει match με το document ενός thread, τότε το thread\_id θα υπάρχει μέσα στη δομή matched\_threads(set).
* Αντίστοιχα όταν θα έχει γίνει matched, αποθηκεύεται και όποτε χρειάζεται κάνει update στο ζευγάρι key-value -> thread\_id, dist στη δομή dist(map).

Με αυτό τον τρόπο συγκρατούνται οι απαραίτητες πληροφορίες για το συγκεκριμένο entry για κάθε thread.



Χρησιμοποιείται ένα Queue για να για να κρατάει τα jobs που εκτελούνται εκείνη τη στιγμή στον Job Scheduler.

* **Initialize scheduler**:

Αρχικοποιεί τις μεταβλητές, τα mutexes, condition variables.

* **Submit Job**:

Κάνει push στην ουρά ένα job.

* **Execute all jobs:**

Δίνει εντολή στα threads να ξεκινήσουν την δουλειά τους.

* **Wait all tasks finish**

Περιμένει να ολοκληρωθούν όλα τα jobs από τα threads και λαμβάνει σήμα όταν σταματήσουν.

* **Destroy Scheduler**

Καταστρέφει τις μεταβλητές που είχαν αρχικοποιηθεί στην initialize.

**Threads:**

Τα threads δημιουργούνται στην Initialize Index και περιμένουν signal σε ένα condition variable από την execute all jobs για να ξεκινήσουν να παίρνουν jobs από την ουρά. Όταν δεν υπάρχει άλλο job να πάρουν, τελειώνουν και δίνουν σήμα ότι τελείωσαν στην wait all tasks finish. Στην συνέχεια περιμένουν ξανά στο αρχικό condition variable και συνεχίζεται η διαδικασία συνέχεια μέχρις ότου να έρθει σήμα από την destroy scheduler και να σταματήσουν.

* Match\_Document:

Η συνάρτηση Match Document έχει τροποποιηθεί ώστε να κάνουμε submit την MatchDocument\_mt, στην οποία έχει μεταφερθεί η κύρια λειτουργικότητα της Match Document, μαζί με τα ορίσματά της τα οποία συμπεριλαμβάνονται σε ένα struct.

* Struct Job:

Το struct αυτό κρατάει τις πληροφορίες για ένα Job, πιο συγκεκριμένα τη συνάρτηση διεκπεραίωσης (MatchDocument\_mt) καθώς και τα arguments που αναφέρονται σε ένα συγκεκριμένο κείμενο (δομή arguments μόνο για MatchDocument περιπτώσεις ενώ η αρχική ιδέα ήταν το job να εκτελεί και Start/End Queries. Γι’ αυτό τα ορίσματα είναι generic).

**Σχεδιαστικές επιλογές τοποθέτησης mutexes-condition variables**

* MatchDocument:

Τα mutexes που χρησιμοποιήθηκαν για την επίτευξη του παραλληλισμού στην εκτέλεση των Jobs τοποθετήθηκαν εντός των δομών entry. Πιο συγκεκριμένα η ανάγκη κλειδώματος μεταβλητών κοινής μνήμης για αποφυγή συγκρούσεων βρίσκεται μόνο στις περιπτώσεις ενημέρωσης των δομών ενός entry. Πράγμα που σημαίνει ότι μπορεί να γίνεται διάσχιση των δομών δεδομένων ταυτόχρονα από πολλά threads και το μόνο ίσως σημείο καθυστέρησης βρίσκεται σε συγκεκριμένες οντότητες.

* JobScheduler:

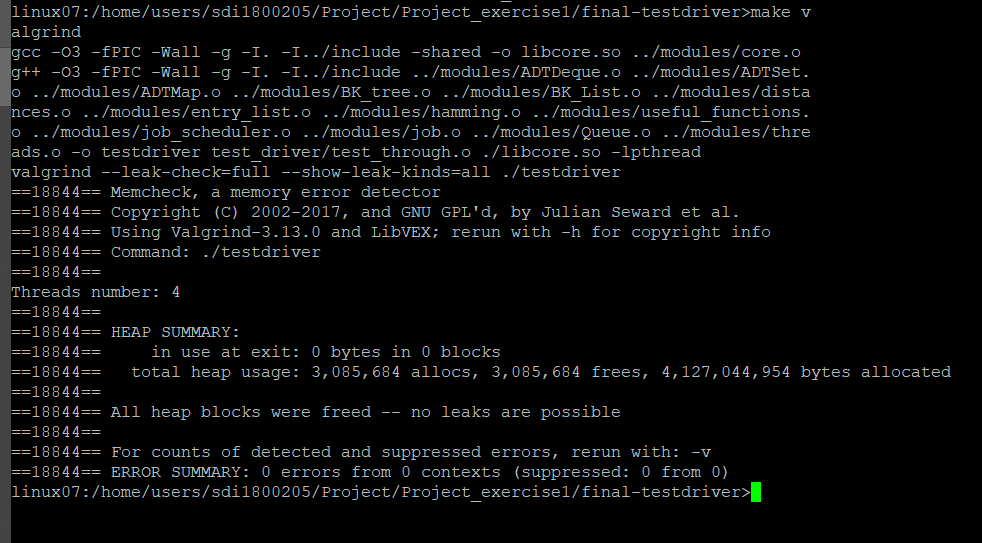
Χρησιμοποιήθηκαν mutexes για την απόκτηση ενός Job από την ουρά και condition variables για τον έλεγχο της εκκίνησης και λήξης των threads.

**Πειράματα:**

|  |  |
| --- | --- |
| **Αριθμός Threads** | **Χρόνος (ms)** |
| 1 | 714 |
| 2 | 410 |
| 3 | 318 |
| 4 | 280 |
| 5 | 281 |
| 6 | 281 |
| 8 | 293 |
| 10 | 293 |
| 50 | 318 |
| 100 | 320 |
| 1000 | 380 |
| 10000 | 1082 |

**Παρατηρήσεις:**

Παρατηρήσαμε πως στην αρχή πως με 1 thread ο χρόνος ήταν πιο αργός απ’ όταν το πρόγραμμα έτρεχε σειριακά (714 ms έναντι ~600ms το σειριακό). Στη συνέχεια, αυξάνοντας των αριθμό των threads που χρησιμοποιήσαμε για την παραλληλοποίηση του προγράμματος, ο χρόνος μειωνόταν αισθητά μέχρι τα 4-5-6 threads που έφτασε στον μικρότερο χρόνο εκτέλεσης 280 ms!! Μετά, όσο αυξάνονταν τα threads, αυξανόταν και ο χρόνος εκτέλεσης που είχε ανοδική πορεία μέχρι και την τελευταία δοκιμή στα 10.000 threads με χρόνο 1.082 sec.



Για την μνήμη, όπως φαίνεται και στο παραπάνω screenshot οι τιμές κυμαίνονται περίπου στα 4 δισεκατομμύρια δεσμευμένα bytes. Δεν υπάρχουν leaks.

**Σχεδιαστικές επιλογές πολυνηματισμού-multithreading:**

Επιχειρήσαμε να παραλληλοποιήσουμε και τις συναρτήσεις StartQuery και EndQuery ωστόσο, οι χρόνοι εκτέλεσης αυξάνονταν περίπου στο διπλάσιο αντιστοίχως για κάθε αριθμό thread. Επομένως η καλύτερη σχεδιαστική επιλογή ήταν να παραλληλοποιήσουμε την Match Document και όλες οι υπόλοιπες συναρτήσεις να εκτελούνται σειριακά.

**Τελικές παρατηρήσεις**

Για την εκτέλεση του 3ου μέρους τις εργασίας χρησιμοποιήθηκαν οι δομές τους 2ου μέρους, οι οποίες ενημερώθηκαν ώστε να δέχονται και το thread που είχε αναλάβει την εισαγωγή και την επεξεργασία της κάθε δομής.

Οι συγκεκριμένες δομές μας έδωσαν τους καλύτερους χρόνους εκτέλεσης.

Αρχικά, η εργασία δοκιμάστηκε τοπικά στα μηχανήματα του καθενός από εμάς . Όμως, οι χρόνοι και οι τελικές εκτελέσεις έγιναν στους υπολογιστές του εργαστηρίου linux του τμήματος Πληροφορικής και Τηλεπικοινωνιών.