ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

Μάθημα
Προγραμματισμός Συστήματος

Project 1

Όνομα
Κυλάφη Χριστίνα-Θεανώ

ΑΜ
1115201200077

Αθήνα, 2018

Η άσκηση έχει υλοποιηθεί σε γλώσσα C, στο πρόγραμμα “sublime text” ενώ παράλληλα δοκιμαζόταν σε περιβάλλον linux μέσω απομακρυσμένης σύνδεσης μέσω τερματικού macOS.

Στο φάκελο, περιλαμβάνονται τα εξής αρχεία εκτός από το readme :
makefile, main.c, structs.h, trie.c, trie.h, funs.c, funs.h, sorting.c, sorting.h, extras.c, extras.h .

Με την εντολή “make” παράγεται το εκτελέσιμο “**minisearch**”. Η σύνταξη για την εκτέλεσή του, είναι:
./minisearch -i <**όνομα αρχείου προς ανάγνωση**> **-k** <**πλήθος αποτελεσμάτων προς εμφάνιση**>
ή
./minisearch -k <**πλήθος αποτελεσμάτων προς εμφάνιση**> **-i** <**όνομα αρχείου προς ανάγνωση**>
ή
./minisearch -k -i <**όνομα αρχείου προς ανάγνωση**> (αποκτά default τιμή αποτελεσμάτων = 10)
ή
./minisearch -i <**όνομα αρχείου προς ανάγνωση**> **-k** (αποκτά default τιμή αποτελεσμάτων = 10)

Έχω προσθέσει ελέγχους λανθασμένης σύνταξης, για λιγότερα ή λανθασμένα ορίσματα.

Το πρόγραμμα, ανοίγει το συγκεκριμένο αρχείο που έχει επιλέξει ο χρήστης, αποθηκεύει σε έναν πίνακα τα κείμενα (κάθε γραμμή κι ένα κείμενο του αρχείου) και στη συνέχεια, ανάλογα με τις εντολές του χρήστη (/search, /df ή /df λέξης, /tf ή /exit), παράγει τα αντίστοιχα αποτελέσματα, τυπώνοντας τέλος μήνυμα επιτυχημένης απελευθέρωσης της μνήμης αν έχει επιλεγεί η εντολή “/exit”.

Δομές:

Για την επίλυση του προβλήματος της εργασίας, δημιουργήθηκαν και χρησιμοποιήθηκαν :

- **Πίνακας** αποθήκευσης κειμένων (char *keimena[lines]), μεγέθους όσες και οι γραμμές(κείμενα) του αρχείου, το οποίο σάρωσα αρχικά για να τις μετρήσω. Στη δεύτερη σάρωση του αρχείου, δημιουργώ δυναμικά χώρο για κάθε i -οστό κείμενο του αρχείου στον πίνακα (keimena[i]) και το αντιγράφω, με τη συνάρτηση create_map( ).

- **Δομή Trie** που λειτουργεί σαν λεξικό του αρχείου. Συγκεκριμένα, αποτελείται απο κόμβους τύπου struct Trie_t, οι οποίοι περιέχουν ένα χαρακτήρα, δείκτες σε επόμενο κόμβο ή / και παιδί, flag για τέλος λέξης, δείκτη σε posting list αν είναι τέλος λέξης και μετρητή της document frequency(σε πόσα κείμενα εμφανίζεται) για τη συγκεκριμένη λέξη. Έχουν παιδιά(child) και γείτονες(next) για σύνδεση μεταξύ τους. Κάθε παιδί αποτελεί επόμενο γράμμα μιας λέξης και κάθε γείτονας επόμενο γράμμα της λέξης μέχρι και τον πατέρα του κόμβου του οποίου είναι γείτονας(ένα level πιο πριν). Σχηματικά, είναι ακριβώς όπως έχει ζητηθεί στην εκφώνηση της εργασίας. Δημιουργείται με τη συνάρτηση create_trie( ) και όταν επιλεγεί η εντολή “/exit”, ελευθερώνεται η μνήμη που έχει δεσμευτεί, με τη συνάρτηση trie_destr( ).

- **Απλά συνδεδεμένη Postings List** για τους κόμβους που περιέχουν τελευταίο γράμμα κάποιας λέξης του αρχείου. Σε όσους κόμβους του λεξικού Trie έχουμε τελευταίο γράμμα λέξης, τότε αυτός ο κόμβος δείχνει σε μια λίστα (απλά συνδεδεμένη) που περιέχει όλα τα ζεύγη κειμένου-εμφανίσεων της συγκεκριμένης λέξης, την postings list. Όπως και με το Trie, δημιουργείται με τη συνάρτηση posting_list σε κάθε κόμβο που συναντάται τέλος λέξης, και όταν επιλεχθεί η “/exit”, καλείται η συνάρτηση trie_destr( ), η οποία ελευθερώνει τη μνήμη που δέσμευσε η δομή postings list συνολικά για όλους τους κόμβους που έχουν λίστα.

- **Πίνακας με scores** για κάθε κείμενο. Κάθε θέση είναι δείκτης σε struct idscore, που πρόκειται για ζευγάρι id κειμένου και score. Κάθε φορά που ο χρήστης επιλέγει την εντολή “/search” , τότε δημιουργώ έναν πίνακα (τον οποίο ελευθερώνω πριν ξαναπάρω νέα εντολή από το χρήστη) και βρίσκω για κάθε μία λέξη που αναζητώ (αν υπάρχει στο αρχείο) το σκορ των κειμένων που τις περιέχουν(αυτό γίνεται με τη συνάρτηση scorecalc βαθμιαία ανά λέξη - εξηγείται παρακάτω στη συνάρτηση scorecalc( )) όλες ή κάποιες από αυτές. Τελικά θα έχω στο τέλος έναν πίνακα όπου σε κάθε θέση, αν εχω το id ίσο με τη θέση του πίνακα, τότε υπάρχει σκορ και το κείμενο θα είναι υποψήφιο για να μπει στον επόμενο πίνακα, τον topk.

- **Πίνακας με τα topk scores** τα οποία τελικά θα εκτυπωθούν κατά φθίνουσα σειρά (με βάση τα σκορ), μαζί με στοίχιση και υπογράμμιση. Αυτή τη φορά έχω μόνο k θέσεις στον πίνακα ή αν έχω λιγότερα κείμενα από “kresults”, δημιουργώ μικρότερο πίνακα, ακριβώς τόσων θέσεων (αν πχ. Ο χρήστης ζητήσει 100 αποτελέσματα και έχω μόνο στα 5 κείμενα score, τοτε θα εχω πίνακα topk με 5 μόνο θέσεις).

- **Συμβολοσειρά με max length λέξης αρχείου** η οποία χρησιμοποιείται στην εκτύπωση των λέξεων του Trie κατά την df_all. Συγκεκριμένα, κατά την create_trie, κρατάω το μέγιστο μέγεθος λέξης, κι έπειτα πριν αρχίσω να εκτελώ την df_all δεσμεύω μνήμη για τη συμβολοσειρά με μέγεθος = max lenght λέξης που έχω κρατήσει από πριν. Αυτή η πρακτική ενώ φαίνεται να σπαταλά μνήμη, εν τέλει, αν το σκεφτεί κανείς, θα εκτυπωθούν όλες οι λέξεις, μαζί κα η λεξη μεγαλύτερου μήκους, οπότε η συμβολοσειρά θα χρησιμοποιηθεί ολόκληρη σε κάποιο σημείο της df_all κάτι που κάνει αυτή την πρακτική(που θα περιγράψω στην df_all) αποδοτική .

- **Πίνακας argarray** στον οποίο κρατώνται τα ορίσματα της εντολής “/search” ώστε να γίνεται αργότερα η σύγκριση για την υπογράμμιση αυτών στην εκτύπωση.

Συναρτήσεις:

- **trie.c**
  - **create_trie( ) :** με αυτή τη συνάρτηση δημιουργείται η βασική δομή, το λεξικό του αρχείου που δίνεται σαν όρισμα. Συγκεκριμένα, ξεκινάμε με έναν κόμβο ρίζα, ο οποίος είναι ίδιου τυπου με τα παιδιά (για μελλοντική ευχρηστία σε συναρτήσεις κλπ). Διαβάζω τα κείμενα από το map που τα έχω αποθηκεύσει, ένα ένα και τα αποθηκεύω σε μία προσωρινή δομή για να μην αλλάξει ο map. Αν είναι το πρώτο γράμμα του αρχείου γενικά, δημιουργώ το παιδί κομβο της ρίζας, το οποίο είναι μοναδικό και από το οποίο γίνεται η αρχή σε πολλές συναρτήσεις. Έπειτα, μέχρι να τελειώσει η κάθε λέξη που εισάγω, ψάχνω αν υπάρχει γράμμα στο trie που έχει δημιουργηθεί μέχρι στιγμής(ξεκινώντας κάθε φορά από τον αρχικό εκείνο κόμβο του πρώτου γράμματος), ίδιο με αυτό που διαβάζω. Εαν είναι ίδιο, συνεχίζω να ψάχνω στα παιδιά μέχρι να φτάσω σε null επόμενο παιδί και είτε να ενημερωσω το postings list της λέξης εαν ήδη υπάρχει, είτε να προσθέσω παιδιά και να δημιουργήσω την postings list της νέας λέξης. Αν δεν είναι ίδιο , ψάχνω στους γείτονες μέχρι να βρω ίδιο ή μέχρι να φτάσω στο τελευταίο γείτονα και να προσθέσω πια το γράμμα που έχω μόλις διαβασει από το κείμενο. Συνεχίζω τη διαδικασία αυτή, μέχρι να φτάσω στο τελευταίο γράμμα της λέξης που θέλω να εισάγω, ώστε να δημιουργήσω/ενημερώσω τότε την postings list με την επόμενη συνάρτηση.
  - **posting_list( ) :** όπως περιέγραψα αναλυτικά στην προηγούμενη συνάρτηση , αν έχω τελικό κόμβο λέξης που προϋπάρχει , ανανεώνω τη λίστα με τις εμφανίσεις της λέξης, προσθέτοντας κόμβο εαν έχω νέο κείμενο που την περιέρχει ή αυξάνοντας το μετρητή του κόμβου του ίδιου κειμένου, αλλιώς, αν δεν έχω ήδη λίστα σε εκείνο τον κόμβο, δημιουργώ νέα. Επίσης, κάθε φορά που έχω νέο κείμενο που περιέχει τη λέξη, αφού προσθέσω κόμβο στη λίστα της λέξης, αυξάνω και το μετρητή κειμένων στον κόμβο της λέξης αυτής (σε πόσα κείμενα εμφανίζεται).
  - **trie_destr( ) :** αναδρομικά και ομοια με την df_all που περιγράφω παρακάτω, ελευθερώνω τη μνήμη που έχω δεσμεύσει για τη δομή trie και postings list κάθε κόμβου με τελικό γράμμα λέξης.

- **sorting.c**
  - **topkres( ) :** δέχεται τον πίνακα με τα σκορ όλων των κειμένων. Εαν έχω κείμενο με σκορ γι’ αυτή την αναζήτηση, τότε το id του struct της αντίστοιχης θέσης θα είναι ίσο με τον αριθμό της θέσης του struct στον πίνακα, αλλιώς, έτσι όπως έχω επιλέξει να αρχικοποιήσω το struct, θα είναι -9, άρα δε με απασχολεί στο γέμισμα του πίνακα topk. Γεμίζω λοιπόν τον πίνακα topk με τα πρώτα σκορ που βρίσκω στον πίνακα score, κι έπειτα με τη συνάρτηση set_max, βρίσκω το μέγιστο στοιχείο και το τοποθετώ στο τέλος του πίνακα. Μετά συγκρίνω κάθε σκορ που βρίσκω στον πίνακα score με το μεγαλύτερο κι αν είναι μεγαλύτερο, το θέτω αυτό ως μεγαλύτερο στην τελευταία θέση του πίνακα topk, και ανταλλάσσω το προηγούμενο μεγαλύτερο με το μικρότερο σκορ του topk μέχρι τότε, το οποίο το βρίσκω μεσα από τη συνάρτηση min( ). Αλλιώς, αν είναι μεγαλύτερο από το min στοιχείο

του πίνακα, τότε ανταλλάσσω το νέο στοιχείο με το min του πίνακα που βρίσκω με τη συνάρτηση min( ). Τέλος, αφού προσθέσω με αυτό τον τρόπο τα k μεγαλύτερα σκορ στον πίνακα topk , κάνω μία heapsort για αποδοτική ταξινόμηση k σκορ. Η παραπάνω τακτική είναι πολύ αποδοτική, καθώς οι πίνακες score και topk λειτουργούν συμπληρωματικά οσον αφορά τις συγκρίσεις για min(το μαξ είναι σταθερά 1 σύγκριση καθώς πάντα ξέρω τη θέση του). Δηλαδή, έστω ότι όλα τα κείμενα έχουν κάποιο σκορ, τότε για μεγάλο k θα έχω λιγότερες συγκρίσεις min (θα συμπληρωθεί το μεγαλύτερο μέρος του topk από την πρώτη τυχαία εισαγωγή) και για μικρές τιμές k , θα έχω μικρό συντελεστή πολλαπλασιασμού με N-k σκορ που μένουν μετά την τυχαία εισαγωγή. Γι' αυτό επέλεξα αυτή την τακτική.

- **min( ) :** βρίσκει το μικρότερο στοιχείο του πίνακα που του δίνεται σαν όρισμα κι επιστρέφει τη θέση του μικρότερου αυτού στοιχείου στον πίνακα.
- **set_max( ) :** παίρνει έναν πίνακα και βρίσκει το μεγαλύτερο στοιχείο του, τοποθετώντας το, τέλος, στην τελευταία θέση αυτού του πίνακα.
- **HeapSort( ), Heapify( ), BuildHeap( ) :** heapsort ταξινόμηση του πίνακα με τα topk σκορ και καλείται από την topkres( ).

- **funs.c**
  - **search_trie( ) :** δοθείσας μιας λέξης, διατρέχει το Trie. Αν τα γράμματα της λέξης που ψάχνω έχουν τελειώσει και ο κόμβος στον οποίο βρίσκομαι έχει τελευταίο γράμμα λέξης, τότε επιστρέφω τον κόμβο αυτόν. Σε κάθε άλλη περίπτωση ή συνεχίζω να ψάχνω ή επιστρέφω null pointer , που σημαίνει ότι η λέξη δεν υπάρχει στο αρχείο.
  - **df_w( ) :** για τη λέξη που της δίνεται , ψάχνει αν υπάρχει στο αρχείο και επιστρέφει τον df της.
  - **df_k( ) :** για τον κόμβο που της δίνεται, επιστρέφει τον df .
  - **df_all( ) :** η συνάρτηση αυτή, είναι αναδρομική κι τυπώνει όλες τις λέξεις του αρχείου με τον df της κάθε μιας δίπλα. Διατρέχει το Trie αρχικά κατά child και στη συνέχεια για κάθε child κατά next(διπλανά γράμματα γείτονες). Παράλληλα, κάθε φορά που κατεβαίνει σε child αυξάνεται το επίπεδο (level++) και αντίστοιχα κάθε φορά που επιστρέφει από την αναδρομική στον γονέα που κάλεσε την df_all με το παιδί του, μειώνεται το level. Έτσι , κρατώντας το level σε κάθε κλήση και μια συμβολοσειρά που έχω δεσμεύσει (μεγέθους ίση με μέγιστο μήκος λέξης του αρχείου), εαν έχω τέλος λέξης στον κόμβο που βρίσκομαι, προσθέτω το γράμμα του κόμβου αυτού, τον χαρακτήρα τέλους κι έπειτα εκτυπώνω τη λέξη που έχω βρει. Αυτό γίνεται για κάθε επίπεδο για κάθε κόμβο.
  - **tf( ) :** δίνεται η λέξη και το id κειμένου και αν υπάρχει postings list, ψάχνει έναν έναν κόμβο για να βρει ζευγάρι id - frequency με id ίσο με id του κειμένου που δόθηκε. Εαν βρει, τότε εκτυπώνει το tf της αλλιώς βγάζει μήνυμα αντίστοιχο.
  - **scorecalc( ) :** με αυτή τη συνάρτηση , με βάση τον τύπο που έχει δοθεί στην εκφώνηση, υπολογίζουμε το σκορ του κάθε κειμένου για την κάθε λέξη που ζητάμε και γεμίζουμε τον πίνακα score. Εδώ, εαν βρω αναφορά σε κείμενο και το id του στο struct είναι -9, αυτό σημαίνει ότι είναι η πρώτη φορά που μπαίνουμε σε αυτο το κείμενο και άρα αλλάζουμε το id με τη θέση του στον πίνακα ώστε να ξέρουμε ότι έχουμε σκορ.

- **extras.c**
  - **create_map( ) :** με αυτή τη συνάρτηση, δεσμεύεται η μνήμη για το map των κειμένων και γίνεται η αντιγραφή τους στη δομή από το αρχείο.
  - **print( ) :** Εδώ γίνεται η εκτύπωση των top k κειμένων με στοιχίση, καθώς και υπογράμμιση των λέξεων που έχουν ζητηθεί από το χρήστη. Η ιδέα της υπογράμμισης, είναι ότι δημιουργώ μία συμβολοσειρά την οποία τελικά θα τυπωσω από κάτω από την αντίστοιχη γραμμή του κάθε κειμένου. Κρατώ λοιπόν 2 flags και με τους κατάλληλους ελέγχους ανάλογα με το αν έχω κενό ή γράμμα ή αν αλλάζω γραμμή ή αν η λέξη που συναντώ υπάρχει στον πίνακα με τα ορίσματα από το χρήστη, τελικά τα flags αυτά μου δείχνουν αν στη συμβολοσειρά υπογράμμισης, θα εκχωρήσω τον κενό χαρακτήρα ή "^", δηλαδή υπογράμμιση.

- **main.c**
  - **main( ) :** η main αποτελεί το σκελετό της οργάνωσης του προγράμματος , μέσα από την οποία καλούνται οι επιμέρους συναρτήσεις για την παραγωγή των ζητουμένων από το χρήστη. Επέλεξα να επιτελεί και η ίδια αρκετές λειτουργίες, με αποτέλεσμα να μη την έχω διασπάσει εντελώς σε συναρτήσεις, όπως έλεγχοι εισόδου , ορισματων, κλπ

Παραδοχές:

1. Εαν δε δοθεί πλήθος αποτελεσμάτων, τότε ορίζεται μια default τιμή (10).

2. Το πρόγραμμα τερματίζει αν συναντήσει στο αρχείο
   - γραμμή μόνο με κενά
   - id που δεν είναι συνεχόμενο
   - εαν το πρώτο κείμενο ξεκινάει με id > 0 ή id < 0
   - εαν βρει μη αριθμητικό χαρακτήρα σε id του αρχείου

3. Η εντολή "/df <q1> <q2> .. " μπορεί να πάρει όσες λέξεις πληκτρολογήσει ο χρήστης για αναζήτηση του df τους.

4. Η εντολή "/tf" μπορεί να πάρει πολλά ορίσματα αλλά τελικά εμφανίζει το tf μόνο της πρώτης λέξης στο συγκεκριμένο κείμενο.

Χωρισμός αρχείων:

Επέλεξα να χωρίσω τα αρχεία στις εξής κατηγορίες:
- trie: δημιουργία και καταστροφή trie και postings list.
- sorting: συναρτήσεις ταξινομησης σκορ
- funs: οι λειτουργίες του προγραμματος df, tf, search, υπολογισμός score
- extras: δημιουργία map για αποθήκευση κειμένων και εκτύπωση με στοιχιση και υπογράμμιση
- main: ο σκελετός του προγράμματος, ανάγνωση από stdin, οργάνωση λειτουργίας, κλπ
- structs: οι structs τύπου λίστας postings, trie και idscore .

Ενδεικτική εκτέλεση:

./minisearch -i bigDataset.txt -k 10
/df users Arts form enters collection

```
Loading...


Select mode:
1. "/search <query1> <query2> ... <query10>"
2. "/df or /df <query>"
3. "/tf <id keimenou> <query1> <query2> ... <queryi>"
4. "/exit"
/df users Arts form enters collection

-Df mode selected-

users  159

Arts  24

form  129

enters  7

collection  26
```

Πηγές :

1. Συνάρτηση ioctl : https://stackoverflow.com/questions/1022957/getting-terminal-width-in-c

2. Heapsort functions (HeapSort( ), Heapify( ), BuildHeap( )) : https://gist.github.com/nefarel/616667