Κωνσταντίνος Τσιάρας AM: 1115201500165

Προγραμματισμός Συστήματος

Εργασία 1η

Μεταγλώττιση: $make$

Εκτέλεση: i) $./minisearch\ -i\ docfile\ -k\ K$

ii) $./minisearch\ -k\ K\ -i\ docfile$

iii) $./minisearch\ -i\ docfile$

όπου $docfile$ το όνομα του αρχείου κειμένων και K ένας θετικός ακέραιος

Στην περίπτωση (iii) όπου το K δεν δίνεται από τη γραμμή εντολών τίθεται από προεπιλογή ίσο με 10.

Χρησιμοποιούμενες Δομές:

1. Map: Είναι απλά ένας πίνακας θέσεων *doc_count* από strings {*char *docs[doc_count]}* - όπου *doc_count το* πλήθος των κειμένων, το οποίο βρίσκω διατρέχοντας αρχικά ολόκληρο το *docfile*. Η αντιστοίχιση των id με τα κείμενα γίνεται μέσω του index: π.χ. το κείμενο με *id 42* θα είναι το *docs[42]* .

2. Inverted Index: Υλοποιείται με ένα Trie, ακριβώς όπως περιγράφεται από την εκφώνηση. Κάθε κόμβος του Trie (*TrieNode*) περιλαμβάνει ακριβώς ένα χαρακτήρα, δείκτη *next* σε *TrieNode* του ίδιου επιπέδου, δείκτη *child* σε *TrieNode* του παρακάτω επιπέδου και έναν δείκτη σε *PostingList*. Για λόγους επιτάχυνσης των αναζητήσεων αλλά και εκτυπώσεων με αλφαβητική σειρά (κατά την κλήση *"/df"*) οι κόμβοι σε κάθε επίπεδο του Trie διατηρούνται πάντα ταξινομημένοι.
   PostingList: Διατηρεί το document frequency της λέξης η οποία καταλήγει σε αυτό (διατρέχοντας το Trie) καθώς και έναν δείκτη στο πρώτο *ListNode* για τη λέξη αυτή. Εκμεταλλευόμενος το γεγονός ότι τα docs διαβάζονται με τη σειρά και άρα οι κόμβοι *ListNode* εισάγονται ταξινομημένοι ως προς το *id*, επίσης διατηρώ και ένα δείκτη στο τελευταίο *ListNode*, έτσι ώστε να μην χρειάζεται να διατρέχω τη λίστα σε κάθε εισαγωγή λέξης. Για τους χαρακτήρες που δεν αντιστοιχούν σε κάποια λέξη οι δύο αυτοί δείκτες είναι ίσοι με *NULL* και επίσης ισχύει *df=0*.

ListNode: Κόμβος μιας απλά συνδεδεμένης λίστας, που περιέχει έναν πίνακα δύο ακεραίων {*id_times[2]*}, το πρώτο στοιχείο του οποίου είναι κάποιο doc id και το δεύτερο το πλήθος των εμφανίσεων της συγκεκριμένης λέξης σε αυτό το doc.

3. Pairing Heap: Για την αποτελεσματική αποθήκευση των αποτελεσμάτων ενός search query και εύρεση των top-K εξ αυτών επέλεξα την υλοποίηση ενός pairing heap. Ο λόγος που προτίμησα τη συγκεκριμένη δομή (έναντι, π.χ., κάποιου binary heap) ήταν η πολυπλοκότητά του. Πιο συγκεκριμένα, έχει χρόνο εισαγωγής **O(1)**, όπου οι εισαγωγές δύνανται να έχουν πλήθος *doc_count* ενώ οι εξαγωγές με **O(logn)** αναμένονται να είναι λιγότερες – για την ακρίβεια $min(K, doc\_count)$, όπου συχνά θα ισχύει *K << doc_count.*

Παραδοχές:

- Κατά την εκτύπωση των αποτελεσμάτων, εάν μια λέξη δεν χωράει ολόκληρη σε μία γραμμή αντί να «σπάει» τυπώνεται εξ ολοκλήρου στην επόμενη γραμμή. Συνεπώς εάν (έχοντας αφήσει τα απαραίτητα κενά στην αρχή της γραμμής για ευθυγραμμισμένη εκτύπωση όπως ζητείται) μια λέξη δεν χωράει ολόκληρη σε κενή γραμμή του terminal {*curr_word_len > cols_to_write – 1*} τότε δεν μπορεί να πραγματοποιηθεί εκτύπωση των αποτελεσμάτων του συγκεκριμένου search query.
- Σε περίπτωση που τα κείμενα περιέχουν non-ASCII χαρακτήρες (π.χ. ‘£’) το πρόγραμμα εξακολουθεί να συμπεριφέρεται όπως είναι αναμενόμενο, με την εξαίρεση ότι η υπογράμμιση με ‘^’ των αποτελεσμάτων κατά την εκτύπωση κάποιου search query ενδέχεται να μην είναι ευθυγραμμισμένη με του προς αναζήτηση όρους κοντά στους non-ASCII χαρακτήρες.

Πρόσθετες Λειτουργίες:

- Εντολή *“/df word1 word2 ...”*: Επέκταση της “/df” για ταυτόχρονη προβολή του document frequency περισσότερων από μιας συγκεκριμένων λέξεων.
- Εντολή *“/k K”*: Για αλλαγή του *K* κατά την εκτέλεση του προγράμματος.
- Εντολή *“/help”*: Εκτυπώνει στο τερματικό τις διαθέσιμες εντολές του προγράμματος καθώς και τον τρόπο με τον οποίο αυτές συντάσσονται.