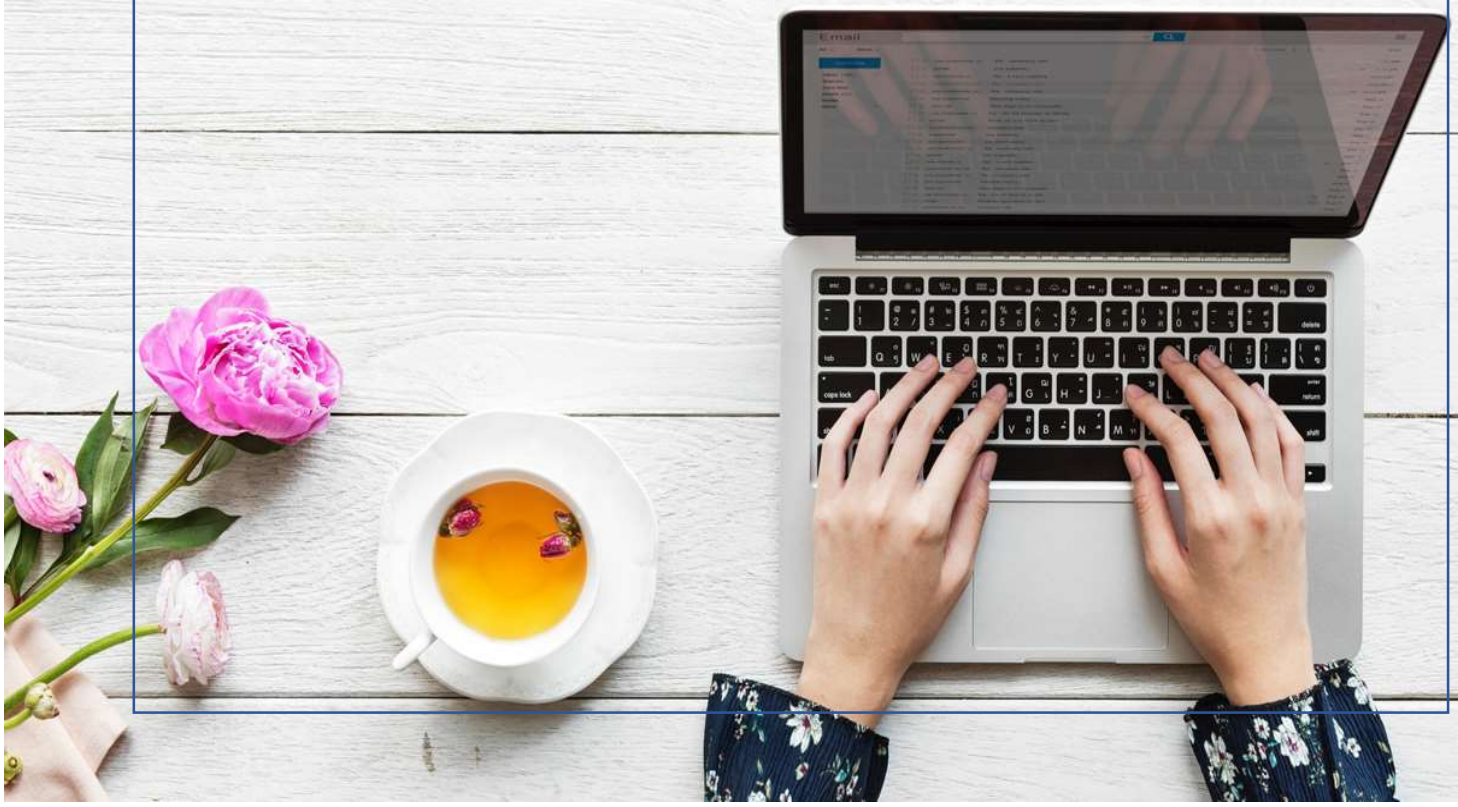
Πέτσα Γεωργία 3200155
Παναγιώτης Τριανταφυλλίδης 3200199

Τρίτη εργασία στις Δομές Δεδομένων 2021-2022



ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Α.

α) void insert(String w)

Για να υλοποιήσουμε την μέθοδο αυτή ελέγχουμε αν υπάρχει ήδη το στοιχείο μέσα στο ΔΔΑ, και αν υπάρχει αυξάνουμε το NumberOfTimes του ήδη υπάρχον κόμβου. Αλλιώς ελέγχουμε αν η λέξη πρέπει να «μπει» αριστερά ή δεξιά του τρέχοντα κόμβου (current) και την βάζουμε στη σωστή θέση. Έπειτα ξανά διασχίζουμε το ΔΔΑ μέχρι και τον κόμβο που μόλις εισήλθε για να ενημερώσουμε σωστά το subtreeSize (ουσιαστικά δηλαδή αυξάνουμε το subtreeSize κάθε πατέρα από τη ρίζα μέχρι το πατέρα του κόμβου που μόλις μπήκε).

β) WordFreq search(String w)

Διατρέχουμε τους κόμβους και κάθε φορά ελέγχουμε αν το στοιχείο στον τρέχοντα κόμβο (current) ισούται με το στοιχείο που ψάχνουμε. Αν βρεθεί αυτό το στοιχείο παίρνουμε την συχνότητα του (current.getItem().getNumOfTimes()) και τη συγκρίνουμε αν είναι μεγαλύτερη από τη μέση συχνότητα. Αν αυτό ισχύει, αφαιρούμε τη λεξη-κομβο από το ΔΔΑ και την βάζουμε στην ρίζα καλώντας την insertAtRoot.

γ) void remove(String w)

Αρχικά διατρέχουμε όλο το δέντρο ώστε να εντοπίσουμε αν υπάρχει η λέξη. Αν αυτό ισχύει, κάνουμε την ιδιά διαδικασία, και κρατάμε τον πατέρα ενώ ταυτόχρονα μειώνουμε το subtreesize κάθε κόμβου που περάσαμε.

- Αν ο κόμβος προς διαγραφή έχει μόνο 1 παιδί (είτε δεξιά είτε αριστερά) τότε αποθηκεύουμε το παιδί αυτό στο replace. Έπειτα το replace θα εισαχθεί στη θέση του κόμβου που διαγράφουμε.
- Αν ο κόμβος έχει 2 παιδιά, τότε εντοπίζουμε το αριστερότερο παιδί (findCurrent) του δεξιού παιδιού του προς διαγραφή κόμβου και ταυτόχρονα μειώνουμε τα subtreeSize των κόμβων που περνάμε μέχρι το αριστερότερο παιδί. Έπειτα αφαιρούμε το παιδί αυτό (findCurrent) με την εξής διαδικασία:

- Αν ο findCurrent έχει δεξιό παιδί τότε αυτό θα μπει στην θέση του findcurrent.

Έπειτα αναθέτουμε στο subtreesize του findCurrent το subtreeSize του κόμβου current(αυτόν που αρχικά θέλουμε να αφαιρέσουμε) και το μειώνουμε κατά 1. Μετά ενημερώνουμε τα παιδιά και τον πατέρα του findCurrent

δ) void load(String filename)

Αρχικά, αν το δέντρο δεν είναι κενό, διατρέχουμε την λίστα με τα stopwords και αφαιρούμε τους κόμβους του ΔΔΑ που έχουν αυτές τις λέξεις. Έπειτα, έχουμε έναν πίνακα με τα Accepted characters, και για κάθε λέξη που υπάρχει στο κείμενο αφαιρούμε από αυτήν οτιδήποτε invalid υπάρχει μπροστά και πίσω από την λέξη και την προσθέτουμε στο ΔΔΑ, αν δεν είναι stopWord. Σε αυτό σύμφωνα με την εκφώνηση δεν πρέπει να συμπεριλαμβάνονται λέξεις που έχουν έστω και έναν αριθμό.

ε) int getTotalWords()

Διατρέχουμε το ΔΔΑ μέσω της μεθόδου Traverse που έχουμε υλοποιήσει για διάσχιση του δέντρου ανά επίπεδο και επιστρέφουμε το άθροισμα του numberOfTimes (δηλαδή συχνότητα) του κάθε κόμβου που υπάρχει στο ΔΔΑ.

στ) int getDistinctWords()

Οι διαφορετικές λέξεις του δέντρου είναι ο αριθμός του subtreesize της ρίζας + 1 (την ιδιά τη ρίζα).

ζ) int getFrequency(String w)

Ουσιαστικά κάνουμε μία αναζήτηση στο ΔΔΑ και αν η λέξη βρεθεί επιστρέφουμε την συχνότητά της (numberOfTimes) αλλιώς επιστρέφουμε 0.

η) WordFreq getMaximumFrequency()

Υλοποιήσαμε την ταξινόμηση QuickSort δημιουργώντας έναν πίνακα και επιστρέφοντας το τελευταίο του στοιχείο. (Αφού έχουμε ταξινόμηση σε αύξουσα σειρά)

θ) double getMeanFrequency()

Πάλι χρησιμοποιώντας την Traverse παίρνουμε το άθροισμα όλων των λέξεων που υπάρχουν στο κείμενο και έπειτα διαιρούμε με το πλήθος των διαφορετικών λέξεων που υπάρχουν στο κείμενο και επιστρέφουμε αυτόν τον αριθμό.

Ι) void addStopWord(String w)

Έχουμε υλοποιήσει μια λίστα με τα stopwords και βάζουμε με τη μέθοδο insertAtBack τη λέξη w.

ΙΑ) void removeStopWord(String w)

Έχουμε υλοποιήσει στην κλάση της λίστας (LinkedList) την μέθοδο remove και έτσι αφαιρούμε τη λέξη w.

ΙΒ) printAlphabetically(PrintStream stream)

Έχουμε υλοποιησει την μεθοδο InOrder οπου διατρεχει αναδρομικα τους κομβους του ΔΔΑ με αυξουσα σειρα (από αριστερα δηλαδη προς τα δεξια και από κατω προς τα πανω) πρωτα για το αριστερο υποδεντρο της ριζας μετα για τη ριζα του υποδεντρου και υστερα για το δεξιο.

ΙΓ) printByFrequency(PrintStream stream)

Εδώ και παλι βαζουμε τους κομβους του ΔΔΑ σε έναν πινακα και τον ταξινομουμε με τη μεθοδο quicksort, και υστερα τυπωνουμε τα στοιχεια του πινακα

Β.

Όνομα μεθόδου	Πολυπλοκότητα	Σχόλια
insert	$O(n)$	Worst case: το BST να είναι αλυσιδωτό μόνο προς μία κατεύθυνση (δλδ οι λέξεις να είναι σε αύξουσα ή φθίσουσα σειρά μέσα στο txt) και ο κόμβος να χρειαστεί να μπει στο τελευταίο παιδί οπότε θα διασχίσει όλο το δέντρο μέχρι να εισαχθεί ο καινούριος κόμβος.
search	$O(n)$	Worst case: το BST να είναι αλυσιδωτό μόνο προς μία κατεύθυνση (δλδ οι λέξεις να είναι σε αύξουσα ή φθίσουσα σειρά μέσα στο txt) και ο κόμβος που ψάχνουμε να μην υπάρχει στο BST, οπότε θα ψάξει όλο το δέντρο με ανεπιτυχή αναζήτηση
remove	$O(n)$	Worst case: το BST να είναι αλυσιδωτό, ο κομβος προς αφαίρεση να είναι προς το τέλος και να έχει 2 παιδιά, εκ των οποίων το δεξιο του να έχει μόνο αριστερά παιδιά.
load	$O(n^2)$	Εδώ διατρέχουμε όλο το κείμενο χωρισμενο σε tokens, από αυτά αφαιρείται οτιδηποτε invalid και στο τελικό ΔΔΑ εισέρχονται οι επιτρεπτές λέξεις. Θα τρέξει σε $O(n^2)$
getTotalWords	$O(n)$	Θα τρέξει σίγουρα σε O(n) καθώς πρέπει να διατρέξει όλους τους κόμβους για να πάρει το frequency (numberOfTimes) κάθε κόμβου. Για τον λόγο αυτό υλοποιήσαμε την Traverse που διατρέχει το ΔΔΑ ανά επίπεδο.
getDistinctsWords	$O(1)$	Απλα επιστρέφεται το subtreesize της ρίζας + 1, οπότε τρέχει σε O(1)
getFrequency	$O(n)$	Worst Case: Η λέξη για την οποία ψάχνουμε τη συχνότητα να μην υπάρχει, και τότε θα διατρέξει τόσους κόμβους του ΔΔΑ όσους θα οδηγούσαν στο σημείο όπου θα έπρεπε να εισαχθεί. Αυτό θα είναι n μόνο αν το δέντρο είναι αλυσιδωτό μονο προς τα αριστερα ή δεξια.

getMeanFrequency	$O(n)$	Επιλέξαμε να χρησιμοποιήσουμε την Traverse και εδώ και θα τρέξει σίγουρα n φορές καθώς πρέπει να πάρει τη συχνότητα κάθε λέξης (κόμβου) ούτως ώστε να βρει τη μέση συχνότητα.
addStopWord	$O(1)$	Καλείται η insertAtBack της LinkedList η οποία τρέχει και αυτή σε O(1)
removeStopWord	$O(n)$	Καλείται η remove (της LinkedList) μέσα στην οποία διατρέχουμε την λίστα μέχρι να βρουμε τη λέξη που αναζητούμε. Αν αυτή βρεθεί την αφαιρούμε. Η χειρότερη περίπτωση είναι να βρίσκεται στο τέλος ενώ η καλύτερη να βρίσκεται στην αρχή.
printTreeAlphabetically	$O(n)$	Υλοποιήσαμε την inOrder η οποία διατρέχει τους κόμβους του ΔΔΑ από τον μικρότερο στον μεγαλύτερο, και έτσι κρατήσαμε την πολυπλοκότητα σε O(n)
printTreeByFrequency	$O(n^2)$	Εισάγουμε τους κόμβους σε έναν πίνακα και στην συνέχεια καλούμε την Quicksort η οποία τρέχει σε $O(n^2)$.
getMaximumFrequency	$O(n^2)$	Λόγω της quickSort στην χειρότρερη περίπτωση (η οποία είναι να είναι ήδη ταξινομημένοι οι κόμβοι) η μέθοδος θα τρέξει σε $O(n^2)$.