Δομές Δεδομένων
Εργασία 3
ΤΟΣΚΟΛΛΑΡΙ ΡΟΝΑΛΝΤ
p3160244

α)

-search(String w): Αρχικά θέτουμε στην μεταβλητή WordFreq word το αποτέλεσμα της searchR με αρχικά ορίσματα το TreeNode root και String w. Τώρα η searchR θα μας επιστρέψει ένα αντικείμενο τύπου WordFreq το οποίο θα είναι η λέξει που ψάχνουμε ή null εάν δεν υπάρχει. Μέσα στην searchR κοιτάμε τον κόμβο που θα εξετάσουμε εάν είναι null(δεν δείχνει πουθενά στην μνήμη). Εάν υπάρχει κάπου στην μνήμη τότε συγκρίνουμε το WordFreq που περιέχει με ένα νέο WordFreq που περιέχει την λέξη που ψάχνουμε. Ο comparator συγκρίνει τις λέξεις των WordFreq pWord και wWord. Εάν είναι ίδιες τότε ο comparator επιστρέφει 0, άρα γυρνάμε το TreeNode που έχει την λέξη. Αλλιώς εάν η wWord είναι αλφαβητικά μικρότερη από την pWord τότε καλούμε αναδρομικά την searchR με 'ορισμα το αριστερό παιδί αυτή την φορά και την λέξη που ψάχνουμε. Ομοίως στην αντίθετη περίπτωση ψάχνουμε στο δεξί παιδί. Μόλις τελειώσει η searchR βλέπουμε αν επέστρεψε κάτι. Εάν δεν επέστρεψε κάτι γυρνάμε null(η λέξη δεν υπάρχει). Εάν υπάρχει το πάμε στην ρίζα του δέντρου και επιστρέφουμε το WordFreq αντικείμενο.

-insert(WordFreq w): Αρχικά ελέγχουμε εάν η λέξη υπάρχει ήδη στο δέντρο. Εάν δεν υπάρχει τότε καλούμε την insertR η οποία παίρνει 3 ορίσματα, την ρίζα αρχικά, το WordFreq που θα μπεί στο δέντρο, και τον πατέρα της root που αρχικά είναι null προφανώς. Στην insertR τώρα πρώτα κοιτάμε το TreeNode που βρισκόμαστε εάν είναι null, σε περίπτωση που είναι φτιάχνουμε το TreeNode node, ορίζουμε τον πατέρα του node και αυξάνουμε τα μεγέθη που θέλουμε.(Ουσιαστικά εδώ γίνεται η εισαγωγή στο δέντρο). Αλλιώς γίνεται η επιλογή του κατάλληλου comparator και προχωράμε στο αντίστοιχο παιδί έτσι ώστε να διατηρούμε την ιδιότητα της σωρού.

-remove(String w): Πρώτα κοιτάμε εάν υπάρχει η λέξη που θέλουμε να αφαιρέσουμε στο δέντρο. Εάν υπάρχει τότε καλούμε την remove(TreeNode p) η οποία κοιτάει όλες τις περιπτώσεις(εάν έχουμε 1 παιδί, 2 παιδία κ.ο.κ). Εάν έχουμε 2 παιδία βρίσκει το successor του p και θέτει p = succ(p). Μετά κοιτάει εάν το TreeNode που θα αφαιρέσουμε είναι το δεξί/αριστερό παιδί ή εάν είναι η ρίζα. Κάνει την αφαίρεση του p και μειώνει το μέγεθος κατά 1.

-getDistinctWords(): Επιστρέφουμε το μέγεθος του δέντρο που ενημερώνεται με κάθε insert/remove.

-getTotalWords(): Επιστρέφουμε το συνολικό αριθμό των ορθών λέξεων που διαβάσαμε στο .txt αρχείο μας, ο οποίος ενημερώνεται μέσα στην load.

-update(String w): Κοιτάμε ένα υπάρχει η λέξη μέσω της search(w). Εάν δεν υπάρχει την εισάγουμε στο δέντρο αλλιώς εάν υπάρχει ενημερώνουμε την συχνότητα της.

-**getMeanFrequency():** Επιστρέφουμε το άθροισμα από τις συχνότητες όλων των λέξεων(διάσχιση όλου του δέντρου και ενημέρωση ενός αθροίσματος) διαιρώντας το με το μέγεθος του δέντρου.

-getMaximumFrequency(): Επιστρέφουμε το WordFreq της ρίζας του δέντρου στο οποίο έχει υλοποιηθεί σίγουρα μια φορά η search.

-removeStopWord(String w): Αφαιρεί από την λίστα με τα stop words το String w.

-getFrequency(String w): Αρχικά ελέγχουμε εάν υπάρχει η λέξη μέσω της search, εάν δεν υπάρχει ενημερώνουμε τον χρήστη ότι δεν υπάρχει η λέξη και επιστρέφουμε 0.Αλλιώς επιστρέφουμε την συχνότητα εμφάνισης της λέξης.

-addStopWord(String w): Ενημερώνει την λίστα με το String w.

-load(String filename): Αρχικά θέτουμε τις μεταβλητές που θα χρειαστούμε στην load, ξεκινάμε να διαβάζουμε το αρχείο filename γραμμή προς γραμμή αγνοώντας σύμβολα, ειδικούς χαρακτήρες κλπ. Στο στάδιο αυτό δημιουργούμε και την λίστα με τα stop words(**θεωρώ ότι λέξεις με μέγεθος <= 2 πρέπει να μπουν στην λίστα**) . Μόλις φτιάξουμε την λίστα ξεκινάμε να διαβάζουμε ξανά το αρχείο γραμμή προς γραμμή, αγνοούμε ξανά τους ειδικούς χαρακτήρες όμως αυτή την φορά λέξεις που υπάρχουν στην λίστα με τα stop words **δεν εισάγονται στο δέντρο.** Με κάθε εισαγωγή ενημερώνουμε και το σύνολο των λέξεων που διαβάσαμε συνολικά.

-printAlphabetically(PrintStream stream): Εδώ απλά κάνουμε inOrder διάσχιση στο δέντρο μας και εκτυπώνουμε τις λέξεις αλφαβητικά.

-**printByFrequency(PrintStream stream):** Στην μέθοδο αυτή φτιάχνουμε ένα νέο Δέντρο το οποίο όμως θα χρησιμοποιεί τον FreqComparator για να συγκρίνει τα στοιχεία και να τα βάλει αναλόγως στο δέντρο. Καλούμε την μέθοδο copy, η οποία αντιγράφει τα στοιχεία του υπάρχον δέντρου σε ένα νέο προσωρινό δέντρο. Κάνοντας πάλι μια inOrder διάσχιση στο νέο μας δέντρο θα πάρουμε τα στοιχεία εκτυπωμένα με την συχνότητα εμφάνισης τους.

β)

-insert(WordFreq w): Η πολυπλοκότητα για αυτήν την μέθοδο δεν είναι ιδιαίτερα περίπλοκη και είναι ίση με 0(h) όπου h είναι το ύψος του δέντρου με h να είναι κατά μέσο όρο ίσο με logn.

-search(String w): Ομοίως και αυτή η μέθοδος εξαρτάται από το ύψος του δέντρου οπότε προκύπτει ότι έχει πολυπλοκότητα ίση με O(h) με h να είναι κατά μέσο όρο ίσο με logn.

-remove(String w): Στην μέθοδο αυτή κάνουμε χρήση και της search οπότε έχει σίγουρα O(h) εάν δεν υπάρχει η λέξη και εάν υπάρχει τότε γίνεται O(h^2) με h να είναι κατά μέσο όρο ίσο με logn.

-getDistinctWords(): Είναι μόνο O(1) αφού επιστρέφει μόνο την μεταβλητή.

-getTotalWords(): Είναι μόνο O(1) αφού επιστρέφει μόνο την μεταβλητή.

-update(String w): Η μέθοδος κάνει πρώτα μια αναζήτηση να δει εάν η υπάρχει η λέξη το οποίο είναι ίσο με O(h). Εάν δεν υπάρχει η λέξη τότε έχουμε πολυπλοκότητα συνολικά o(h^2) εφόσον κάνουμε και μια εισαγωγή. Αλλιώς εάν υπάρχει η λέξη τότε η πολυπλοκότητα γίνεται O(h) + O(1) = O(h) με h να είναι κατά μέσο όρο ίσο με logn.

-**getMeanFrequency():** Στη μέθοδο αυτή έχουμε μόνο μια διαίρεση η οποία καλεί δύο μεθόδους που έχουν πολυπλοκότητα O(1), αρα O(1) + O(1) + O(1) = O(1).

-removeStopWord(String w): Κάνουμε μια διάσχιση στη λίστα μέχρι να βρούμε την λέξη μας το οποίο είναι ίσο με O(n).

-getMaximumFrequency(): Επιστρέφουμε την ρίζα του δέντρου το οποίο γίνεται σε χρόνο O(1).

-getFrequency(String w): Γίνεται χρήση της search άρα είναι σίγουρα O(h) και απλά επιστρέφει την συχνότητα εάν υπάρχει η λέξη η null εάν δεν υπάρχει το οποίο είναι O(1). Άρα τελικά έχουμε o(h) με h να είναι κατά μέσο όρο ίσο με logn.

-addStopWord(String w): Κάνει απλά μια εισαγωγή στην λίστα το οποίο είναι O(1).

-load(String filename): Έστω πως K είναι το πλήθος των λέξεων του αρχείου μας επειδή διαβάζουμε δύο φορές το αρχείο μας η πολυπλοκότητα μας είναι ίση με O(K^2).

-printAlphabetically(PrintStream stream): Γίνεται απλά μια διάσχιση(inOrder) στο δέντρο η οποία έχει πολυπλοκότητα ίση με O(n).

-**printByFrequency(PrintStream stream):** Εδώ φτιάχνουμε ένα αντίγραφο του δέντρου μας το οποίο θέλει O(nlogn) και κάνουμε μία διάσχιση στο νέο δέντρο το οποίο θέλει O(n).