Φίλιππος Δουραχαλής **12/1/2019** **p3170045**

Κλάση ST

Μέθοδοι :

1) Void **insert** (WordFreq item) : Η μέθοδος αρχικά ελέγχει αν το στοιχείο item δεν είναι **null** και κατόπιν καλεί την insert(TreeNode node , TreeNode parent , TreeNode newNode) η οποία αναλαμβάνει την αναδρομική εισαγωγή του σαν φύλλο του δέντρου ,ξεκινώντας από τη ρίζα του. Αν ο κόμβος ο οποίος εξετάζεται είναι **null** ,σημαίνει ότι δεν υπάρχει στοιχείο εκεί ,οπότε προσθέτει τον newNode που περιέχει το item ,διαφορετικά συνεχίζει στον σωστό κόμβο-παιδί.

2) Void **update**(String word) : Στην αρχή της μεθόδου ,καλείται η μέθοδος **find** που δέχεται μια λέξη και επιστρέφει τον κόμβο που την περιέχει ,αν υπάρχει . Αν ο κόμβος (δλδ η λέξη) υπάρχει στο δέντρο ,τότε αυξάνεται η συχνότητα της ,διαφορετικά καλείται η **insert** για να εισαχθεί στο σωστό σημείο .

3) WordFreq **search**(String word) : και πάλι καλείται η find για να βρεθεί ο κόμβος που περιέχει την λέξη . Αν υπάρχει ,ελέγχεται η συχνότητα της και αν αυτή είναι μεγαλύτερη της μέσης συχνότητας ,ο κόμβος αφαιρείται μέσω της remove και ξανά εισάγεται στην κορυφή μέσω της **insertR .**

4) Void **remove**(String word) : Αναζητεί τη λέξη μέσω της find .Αν η λέξη βρεθεί ,καλείται η remove(TreeNode node) ώστε να γίνει αφαίρεση της. Αν ο κόμβος είναι φύλλο του δέντρου ,απλά αφαιρείται . Αν έχει δύο παιδία ,θα πρέπει να βρεθεί ο διάδοχός του (δηλαδή ο πιο αριστερός κόμβος του δεξιού υποδέντρου) και να τον αντικαταστήσει ώστε να διατηρηθεί η δομή του δέντρου. Αν έχει ένα παιδί ,τότε απλά τον αντικαθιστούμε με αυτό.

5) Void **load**(String filename) : δέχεται ένα αρχείο και το διαβάζει λέξη προς λέξη . Στην Αρχή ,δημιουργείται ένα String το οποίο περιέχει τα delimiters που είναι αποθηκευμένα στην λίστα stopwords για να δοθεί στην συνέχεια στο StringTokenizer που επιστρέφει τις λέξεις. Καθώς διαβάζεται κάθε λέξη ,ελέγχεται αν περιέχει νούμερα (με την **hasDigit**) . Αν δεν περιέχει ,δίνεται στην update για να ενημερώσει το δέντρο.

6) Int **getTotalWords**() : Χρησιμοποιεί την μέθοδο **count**(TreeNode node) που μετράει την συχνότητα κάθε λέξης του υποδέντρου με ρίζα το node. Αφού ολοκληρωθεί η διάσχιση ,επιστρέφεται το άθροισμα.

7) Int **getDistinctWords**() : επιστρέφει την μεταβλητή στιγμιότυπου distinctWords που ενημερώνεται κάθε φορά που εισάγεται ή αφαιρείται από το δέντρο ένας κόμβος (δλδ μια λέξη).

8) Int **getFrequency**(String word): Χρησιμοποιεί την find για να βρεί την λέξη και επιστρέφει την συχνότητα της αν υπάρχει .

9) Int **getMeanFrequency**() : Καλεί την **getTotalWords**() για να βρει το σύνολο των λέξεων και το διαιρεί με την μεταβλητή distinctWords για να επιστρέψει τον Μέσο Όρο.

10) Int **getMaxFrequency**() : Δημιουργεί ένα καινούργιο δέντρο με κλειδί την συχνότητα εμφάνισης της κάθε λέξης και επιστρέφει την λέξη που βρίσκεται στο πιο δεξί κόμβο ,που θα περιέχει την λέξη με την μεγαλύτερη συχνότητα.

11) Void **printAlphabetically**(PrintStream stream) : καλεί την **printInorder**(TreeNode node, PrintStream stream) η οποία τυπώνει το δέντρο από τα αριστερά προς τα δεξιά.

12) Void **printByFrequency**(PrintStream stream) : δημιουργεί ένα νέο δέντρο με κλειδί την συχνότητα μέσω της **copyTree** και στη συνέχεια καλή την **printRtoL** για να εκτυπώσει το δέντρο από τα δεξιά προς τα αριστερά ,σε φθίνουσα σειρά.

13) Void **addStopWord**(String word): Ελέγχει αν η λίστα περιέχει τη λίστα και αν δεν υπάρχει ,την προσθέτει .

14) Void **removeStopWord**(String word): Καλεί την remove μέθοδο της λίστας που αναζητά και αφαιρεί την λέξη από τη λίστα.

15) TreeNode **getHead**(): Επιστρέφει τη ρίζα του δέντρου.

16) TreeNode **InsertR**(TreeNode node ,TreeNode parent ,TreeNode newNode): εισάγει τον κόμβο newNode σαν φύλλο και με διαδοχικές περιστροφές τον φέρνει στην κορυφή του δέντρου.

Πολυπλοκότητα μεθόδων

1) Insert : Η πολυπλοκότητα της insert στη μέση περίπτωση είναι O(logn) ενώ στη χειρότερη περίπτωση O(N) ,αν το δέντρο είναι ταξινομημένο ,που σημαίνει ότι θα πρέπει να περάσουμε από κάθε κόμβο πρωτού εισαχθεί ο νέος.

2) Update : Η μέθοδος update καλεί δύο μεθόδους ,την find και την insert ,άρα η πολυπλοκότητα της προκύπτει συναρτήσει αυτών των δύο . Η find στη χειρότερη περίπτωση θα έχει πολυπλοκότητα O(N) και στη μέση περίπτωση O(logn) ,για τους ίδιους λόγους με την insert . Άρα και η μέση περίπτωση της update είναι Cn = logn + logn = 2logn δηλαδή O(logn) .Αντίστοιχα στη χειρότερη περίπτωση θα τρέχει σε O(N).
3) Search : Η search χρησιμοποιεί και αυτή την find , σε συνδυασμό με την remove και την insertR .
    - Remove : Στη χειρότερη περίπτωση θα καλέσουμε την μέθοδο successor η οποία θα τρέχει σε O(h) ,με h το ύψος του δέντρου (αν ο successor βρίσκεται στο τελευταίο επίπεδο του δέντρου). Η remove δεν περιέχει άλλους βρόχους ή αναδρομές και άρα η πολυπλοκότητα της εξαρτάται από την successor ,δηλαδή O(h) στη μέση περίπτωση .
    - InsertR : Η insertR κάνει τις ίδιες πράξεις με την απλή εισαγωγή ,με τη διαφορά ότι κάνει διαδοχικές κλήσεις της rotateLeft ή rotateRight όταν εισάγει τον νέο κόμβο για να τον φέρει στην κορυφή . Κάθε μέθοδος rotate εκτελείται σε χρόνο ανεξάρτητο των κόμβων του δέντρου ,άρα σε O(1) .Όμως στην insert οι μέθοδοι αυτοί καλούνται στη μέση περίπτωση h φορές ,ώστε από φύλλο ο κόμβος να γίνει ρίζα. Η πολυπλοκότητα της συνεπώς θα είναι στη Μέση περίπτωση O(logn + h) = O(logn) (Αφού σε τέτοια περίπτωση το h θα είναι ίσο με logn) .Στη χειρότερη περίπτωση θα είναι O(N + h) = O(N) ,αφού το δέντρο θα έχει μορφή λίστας και h = N.

    Συνολικά η πολυπλοκότητα της search στην χειρότερη περίπτωση θα είναι O(N) + O(N) = O(N + N) = O(N) .

4) getTotalWords : Η μέθοδος καλεί την count που διασχίζει ολόκληρο το δέντρο για να υπολογίσει τον αριθμό εμφάνισης κάθε λέξης . Άρα η πολυπλοκότητα της είναι O(N).
5) getDistinctWords : Η μέθοδος έχει χρόνο εκτέλεσης O(1)
6) getFrequency : Όπως και με προηγούμενες μεθόδους που καλούν την find ,η πολυπλοκότητα της μεθόδου αυτής ταυτίζεται με την πολυπλοκότητα της find.
7) getMeanFrequency : Η μέθοδος καλεί την getTotalWords ,η οποία έχει πάντα πολυπλοκότητα O(N) .Επομένως αυτή θα είναι η πολυπλοκότητα της getMeanFrequency .
8) getMaxFrequency : Διασχίζει ολόκληρο το δέντρο για να βρει το στοιχείο με την μεγαλύτερη πολυπλοκότητα ,αρα έχει χρόνο εκτέλεσης O(N).
9) printAlphabetically : Όπως και πριν ,διασχίζεται όλο το δέντρο για να εκτυπώσει το αποτέλεσμα .
10) addStopWord : Η μέθοδος καλεί την DoubleLinkedList μέθοδο contains που - στη χειρότερη περίπτωση - διασχίζει ολόκληρη τη λίστα για να αποφασίσει αν υπάρχει ή όχι η λέξη. Η insertAtBack έχει υλοποιηθεί σε O(1) και άρα η πολυπλοκότητα της μεθόδου είναι O(N).

11) printByFrequency : Σε αυτή τη μέθοδο δημιουργείται ένα νέο δέντρο μέσω της copyTree ,σε O(N) και στη συνέχεια τυπώνεται αυτό το δέντρο πάλι σε O(N) . Άρα η πολυπλοκότητα της είναι O(2N) = O(N).
12) removeStopWord : Καλεί την remove της DoubleLinkedList που έχει πολυπλοκότητα O(N) ,συνεπώς και η removeStopWord έχει ίδια πολυπλοκότητα.