Όνομα: Αναστάσιος Παπαπαναγιώτου
ΑΜ: 3200143

Ερώτημα Β

- Για την insert χρησιμοποιώ μια νέα συνάρτηση (insertHelper) η οποία καλείται αναδρομικά. Συγκεκριμένα, ελέγχει σε ποιο υπόδεντρο πρέπει να γίνει το insert και καλεί τη συνάρτηση σε αυτό το υπόδεντρο. Αν η λέξη που θέλουμε να εισάγουμε υπάρχει ήδη τότε απλά αυξάνει τον αριθμό εμφανίσεων. Καθώς περνάει από κόμβους αυξάνω το μέγεθος του υπόεντρου καθώς θα είναι σίγουρα σε ένα από τα υπόδεντρά του. Όμως αν υπάρχει ήδη η λέξη τότε καλείται (updatePreviousSize) μια συνάρτηση για να μειώσει κατά 1 το μέγεθος των προηγούμενων αφού εν τέλει δεν προστέθηκε νέος κόμβος. Η insert τρέχει σε χρόνο O(logN) average case κι O(N) worst case.
- Η μέθοδος search ψάχνει για μια λέξη συγκρίνοντας την με τον τωρινό κόμβο και αν είναι μικρότερη τοτε ψάχνει στο αριστερό αλλιώς στο δεξί. Αν ο κόμβος είναι null τότε σημαίνει ότι η λέξη δεν υπάρχει, αλλιώς όταν την βρεί ελέγχει ο μετρητής της λέξης αυτής είναι μεγαλύτερος από την μέση τιμή όλων των μετρητών. Αν είναι, τότε αφαιρεί τον κόμβο από το δέντρο και τον προσθέτει ως ρίζα στο δέντρο χρησιμοποιώντας left/right rotations. Η αφαίρεση τρέχει σε O(logN) average, O(N) worst case, η insertAsHead τρέχει σε O(logN) average, O(N) worst case αφού τα rotations γίνονται σε O(1), άρα η search τρέχει σε O(logN) average, O(N) worst case.
- Η remove αφαιρεί μια λέξη από το δέντρο. Βρίσκει αρχικά που βρίσκεται η λέξη στο δέντρο και μετά αλλάζει τη τιμή του κόμβου με τη λέξη που θέλουμε να αφαιρέσουμε με την μικρότερη τιμή του δεξιού υπόδεντρου και αυτό γιατί θα είναι μεγαλύτερη σίγουρα από όλες τις τιμές του αριστερού υπόδεντρου και μικρότερη από όλες τις τιμές του δεξιού. και τέλος αφαιρεί τη μικρότερη λέξη του δεξιού υπόδεντρου αφού θα υπάρχει δύο φορές. Η remove τρέχει σε O(logN) average, O(N) worst case
- Η load διαβάζει ένα αρχείο που η διεύθυνσή του δίνεται ως παράμετρος. Κάνει parse λέξη-λέξη και για κάθε λέξη διαβάζει τους χαρακτήρες και ελέγχει αν είναι γράμμα. Αν δεν είναι γράμμα, ελέγχει αν είναι η απόστροφος στο σωστό σημείο (δηλαδή ανάμεσα σε γράμματα), αλλιώς ελέγχει αν ο χαρακτήρας είναι στην αρχή ή στο τέλος, αλλιώς την αγνοεί. Αν είναι γράμμα, το προσθέτει στον StringBuilder και αφού διαβάσει όλους τους χαρακτήρες, ελέγχει αν δεν είναι stop word και μετά τη προσθέτει. Η load τρέχει σε O(N) όπου N ο αριθμός όλων των χαρακτήρων του αρχείου
- Η getTotalWords επιστρέφει το σύνολο όλων των λέξεων κάνει μια Depth First Search στο δέντρο και από κάθε κόμβο επιστρέφει τον αριθμό εμφανίσεων κάθε λέξης και αυτό το κάνει μέσω recursive calls στην getTotalWordsHelper. Η getTotalWords τρέχει σε O(N) αφού περνάει από όλους τους κόμβους μια φορά.
- Η getDistinctWords επιστρέφει το σύνολο των λέξεων (χωρίς τις εμφανίσεις τους) το οποίο είναι ίσο με τον αριθμό των κόμβων όλου του δέντρου, και επειδή κρατάμε το μέγεθος κάθε υπόδεντρου ως μεταβλητή του TreeNode αρκεί να επιστρέψουμε το μέγεθος της ρίζας + 1 που θα είναι η ίδια η ρίζα, άρα τρέχει σε O(1)

- Η getFrequency επιστρέφει πόσες φορές έχει εμφανιστεί μια λέξη και δηλαδή κάνει αναζήτηση της λέξης αυτής στο δέντρο και επιστρέφει τον αριθμό εμφανίσεών της, άρα τρέχει σε O(logN) average, O(N) worst case.
- Η getMaximumFrequency επιστρέφει τη λέξη με τις περισσότερες εμφανίσεις, και κάνει μια Depth First Search και ανανεώνει σε κάθε κόμβο το max εμφανίσεων το οποίο το οποίο είναι παράμετρος της συνάρτησης, αρα αφού περνάει από κάθε κόμβο τρέχει σε O(N)
- Η getMeanFrequency επιστρέφει τον μέσο όρο εμφανίσεων ο οποίος θα είναι συνολικές λέξεις / μοναδικές λέξεις, και από πριν η συνολικές λέξεις είναι σε O(N) ενώ μοναδικές λέξεις σε O(1) άρα συνολικά θα είναι O(N)
- Η addStopWord προσθέτει μια λέξη σε μια linkedlist η οποία την προσθέτει ως head της linkedlist αρα είναι O(1) operation
- Η removeStopWord αφαιρεί μια συγκεκριμένη λέξη από τη linkedlist. Κάνει traverse όλα τα nodes της και αν την βρει την αφαιρεί, η προηγούμενη λέξη κάνει point στην επόμενη, αρα τρέχει σε O(N)
- Η printTreeAlphabetically κάνει print τις λέξεις με αλφαβητική σειρά και αυτό γίνεται μέσω in-order traversal, αφού: αριστερό node < parent node < δεξί node, αρα αφού περνάει από κάθε node μια φορά θα είναι O(N)
- Η printTreeByFrequency κάνει print τις λέξεις με βάση τον αριθμό εμφανίσεών τους. Αρχικά, κάνει μια in-order traverse για να προσθέσει τα στοιχεία του δέντρου σε μια array, O(N) operation. Μετά χρησιμοποιώντας την quicksort, κάνει sort το array με αύξουσα σειρά, με βάση τη συχνότητα εμφάνισης, O(NlogN) operation. Άρα η printTreeByFrequency τρέχει συνολικά σε O(NlogN)

Ερώτημα Δ

Το TestingFiles περιέχει τα αρχεία που χρειάζονται για να γίνει έλεγχος της σωστής λειτουργίας. Επίσης το src περιέχει δύο .jar αρχεία που χρησιμοποιούνται για να τρέξω JUnit Tests από cmd και τα test περιέχονται στα αρχεία: BSTTest.java, LinkedListTest.java, QuicksortTest.java, τα οποία είναι test cases και τρέχουν στο cmd με τις εντολές (πρέπει να το τρέξω από το directory που είναι το src π.χ. ../…/…/src/):

- javac -cp junit-4.13.1.jar;. BSTTest.java LinkedListTest.java QuicksortTest.java
- java -cp junit-4.13.1.jar;hamcrest-core-1.3.jar;. org.junit.runner.JUnitCore BSTTest LinkedListTest QuicksortTest

Τα αρχεία που χρειάζονται όμως δουλεύουν κανονικά και με το cmd χωρίς JUnit. Π.χ. για το Greedy

- javac Main.java
- java Greedy

Διάφορες σημειώσεις

- Στο menu για να γίνει load ένα αρχείο πρέπει να δοθεί όλη η διεύθυνση χωρίς “ ”, δηλαδή: C:\Users\...\...\...\TestingFiles\test1.txt

- Στην printTreeAlphabetically, printTreeByFrequency, έχει τεστραριστεί μόνο για PrintStream printStream = System.out