Δομές Δεδομένων – Εργασία 3

Τμήμα Πληροφορικής
Χειμερινό Εξάμηνο 2021-2022
Υπεύθυνοι φοιτητές:
Μαρία Κονταράτου (3200078)
Γεώργιος Κουμουνδούρος (3200083)
Διδάσκων: Ε.Μαρκάκης

ΑΘΗΝΑ, ΙΑΝΟΥΑΡΙΟΣ 2021

ΜΕΡΟΣ Γ – ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

Α. Επεξήγηση υλοποίησης μεθόδων και πολυπλοκότητάς τους

public void insert(String w):
Σε πρώτο στάδιο τσεκάρουμε αν η λέξη βρίσκεται στη λίστα ο κόμβος με κλειδί w μέσω της μεθόδου isInList(). Αν δεν βρίσκεται, καλούμε την insertR(), ενώ αυξάνουμε την μεταβλητή totalWords και θέτουμε την μεταβλητή isLeaf σε false, δηλαδή ότι δεν θα γίνει εισαγωγή σε πρώτη φάση ως φύλλο. Ως προς την insertR(). Αν το head είναι null, σημαίνει ότι χρειάζεται να δημιουργήσουμε ένα νέο item TreeNode και να το εισάγουμε ως φύλλο. Στη συνέχεια, ψάχνουμε το υπό-δεντρο που θα εισάγουμε το item. Αν η λέξη είναι ήδη στο BST, δεν την εισάγουμε, απλά αυξάνουμε τις φορές που την βρίσκουμε στο πρόγραμμα ενώ παράλληλα τσεκάρουμε αν το αντικείμενο έχει μεγαλύτερη συχνότητα από το ήδη υπάρχων με μέγιστη συχνότητα και ενημερώνουμε με το αντίστοιχο item την maxFW. Αν η λέξη δεν βρίσκεται στο δέντρο, την εισάγουμε στο δεξί ή αριστερό δέντρο και τελικά επιστρέφουμε την «ρίζα» του δέντρου.

Η πολυπλοκότητα της μεθόδου είναι O(N). Σε περίπτωση που το δέντρο μας πχ δεν είναι ισοσταθμισμένο (balanced) θα χρειαστεί να διασχίσουμε όλο το δέντρο για να βρούμε πού θα εισάγουμε το item.

public WordFreq search(String w):
Καλούμε αρχικά την συνάρτηση find που δημιουργήσαμε με σκοπό να δούμε αν το string όντως υπάρχει στο συγκεκριμένο δέντρο ή όχι. Στην find(), παίρνουμε το εκάστοτε string και σε πρώτο στάδιο το συγκρίνουμε με το στοιχείο του συγκεκριμένου υπο-δεντρου ενώ στη συνέχεια το βάζουμε δεξιά ή αριστερά ανάλογα με το αποτέλεσμα της σύγκρισης. Αν το βρούμε στο δέντρο, τσεκάρουμε αν έχει συχνότητα μεγαλύτερη της μέσης και τότε μέσω της insertRoot() τοποθετούμε το node στο head του δέντρου, εκτελώντας τις σωστές περιστροφές (εισαγωγή στη ρίζα για το συγκεκριμένο κόμβο). Γενικός σκοπός μας σαφώς είναι λέξεις με μεγάλη συχνότητα - δηλαδή λέξεις για τις οποίες μπορεί να γίνουν πολλές αναζητήσεις - να τις έχουμε όσο το δυνατόν πιο ψηλά στο δέντρο.

Η πολυπλοκότητα της μεθόδου είναι O(N). Αφού χρειάζεται να διασχίσουμε όλα τα στοιχεία ένα προς ένα του εκάστοτε δέντρου σε μια συγκεκριμένη σειρά.

public void remove(String w):
Όσο υπάρχει το αντικείμενο που θέλουμε να διαγράψουμε – δηλαδή όσο find(w)!=null – καλούμε την removeR() (recursive) την οποία είδαμε και στις διαλέξεις. Ουσιαστικά στην removeR() «διασχίζουμε» το αριστερό ή δεξί υπο-δεντρο και ακολουθούν και ανάλογα το kay()θα έχουμε τρεις περιπτώσεις:
- Το node να είναι μέγιστης συχνότητας, όπου εκτελείται κανονικά η remove καθιστώντας το maxFW ως null.
- Το αντικείμενο προς αφαίρεση να έχει ένα ή κανένα παιδί, όπου συνδέουμε τον πατέρα με το παιδί, αφού κάνουμε τον πατέρα να δείξει το παιδί του
- Το αντικείμενο προς αφαίρεση να έχει δύο παιδιά , όπου θα χρειαστεί να βρούμε το μικρότερο στοιχείο από το δεξί υπο-δεντρο, να το αποθηκεύσουμε εκεί που είναι το στοιχείο προς αφαίρεση, και όταν αποκτήσει και ρόλο φύλλου να το διαγράψουμε.

Αν η maxFW είναι ίση με το string w που θέλουμε να αφαιρέσουμε, τότε τη βάζουμε να βρει το node μέγιστης συχνότητας εντός του δέντρου.

Η removeR(), συνεπώς και η remove() έχουν πολυπλοκότητα O(N). Αυτό συμβαίνει αφού χρειάζεται πάλι να διασχίσουμε όλα τα στοιχεία για να βρούμε εκείνο που θέλουμε να διαγράψουμε.

public void load(String filename):
Στη συγκεκριμένη περίπτωση, δουλέψαμε δημιουργώντας 3 boolean μεταβλητές: isOK, isStringWithOther και build. Διαβάζοντας κάθε χαρακτήρα, εξετάζουμε κάθε περίπτωση ξεχωριστά και αλλάζουμε ανάλογα τη συνθήκη την μεταβλητή σε true/false.

Στην isOK, εξετάζουμε αν ο χαρακτήρας μας είναι γράμμα ή απόστροφος, αν είναι κενό ή κάποια μορφή σημείου στίξης ή αριθμός. Αν ανήκει στην πρώτη περίπτωση, τότε μπορούμε να θέσουμε το build σε true καθώς είναι γράμμα και θέλουμε να το λάβουμε υπ'όψιν.

Στην isStringWithOther, ήδη από την πορηγούμενη κατάσταση έχουμε θέσει ως true την isStringWithOther όταν βρίσκουμε αριθμό. Τώρα αν σε αυτό το στάδιο ο χαρακτήρας μας είναι είτε γράμμα είτε απόστροφος είτε πάλι αριθμός, απλά συνεχίζουμε ενώ αν βρούμε κενό ή σημείο στίξης πάμε στην προηγούμενη κατάσταση.

Στην build, απλά μέσω του StringBuilder ουσιαστικά χτίζουμε ένα string με όλες τις «σωστές» λέξεις που θέλουμε. Αν ο χαρακτήρας μας είναι γράμμα ή απόστροφος, κάνουμε append μέχρι να βρούμε κενό ή σημείο στίξης όπου τότε κάνουμε update το string και πάμε στην κατάσταση isOK. Αν πριν το κενό ή το σημείο στίξης βρούμε πρώτα αριθμό πάμε στην κατάσταση isStringWithOther.

Η πολυπλοκότητα της μεθόδου μας είναι O(N), αφού ελέγχουμε κάθε χαρακτήρα του αρχείου με το for-loop μας και μετά έχουμε απλά if-statements.

public int getTotalWords():
Επιστρέφει τον ακέραιο totalWords ο οποίος ενημερώνεται στο πρόγραμμα καθώς τρέχει και συνεπώς έχει πολυπλοκότητα O(1).

pubic int getDistinctWords():
Εάν ο ακέραιος totalWords είναι 0, επιστρέφει 0. Μέσω της μεταβλητής int subtreeSize, οποία μας δίνει το σύνολο των κόμβων που βρίσκονται από κάτω από τον τρέχοντα κόμβο επιστρέφουμε όλες τις διαφορετικές λέξεις, δηλαδή το head.subtreeSize αθροισμένο με τον τρέχοντα κόμβο head. Η πολυπλοκότητα είναι και σε αυτή την περίπτωση O(1).

public int getFrequency():
Σε πρώτη φάση, ψάχνουμε μέσω της search() αν υπάρχει το string. Αν δεν υπάρχει, επιστρέφουμε 0. Σε αντίθετη περίπτωση, επιστρέφουμε αντικείμενα WordFreq που από την getTimesFound() βλέπουμε πόσο συχνά έχουν εμφανιστεί. Αφού η search() είναι πολυπλοκότητας O(N), τότε και η getFrequency() θα είναι και εκείνη ίδιας πολυπλοκότητας.

public int getMaximumFrequency():
Εάν ο ακέραιος totalWords είναι 0, επιστρέφει 0. Αν όχι, επιστρέφει τη μεταβλητή maxFW η οποία ανανεώνεται καθώς τρέχει το πρόγραμμα και συνεπώς έχει πολυπλοκότητα O(1).

public int getMeanFrequency():
Επιστρέφουμε την το αποτέλεσμα της διαίρεσης του getTotalWords() με το getDistinctWords(). Οι δύο παραπάνω μέθοδοι έχουν πολυπλοκότητα O(1) συνεπώς και το πηλίκο τους θα έχει ίδιας τάξης πολυπλοκότητα.

public addStopWord(String w):
Μέσω της μεθόδου insertAtLIst(), τοποθετούμε στη λίστα stopWords το w εάν δεν είναι κενή. Όπως ζητήθηκε, η μέθοδος είναι πολυπλοκότητας O(1).

public removeStopWord(String w):
Μέσω της μεθόδου removeFromList αφαιρούμε από τη λίστα stopWords το w. Η πολυπλοκότητά της θα είναι O(stopWords.getSize()) καθώς τρέχει όλα τα στοιχεία της λίστας για να βρει αν υπάρχει η συγκεκριμένη stopWord.

public void printTreeAlphabetically(PrintStream stream):
Στο δέντρο μας κάνουμε μία διάσχιση τύπου inOrder, δηλαδή πρώτα εμφανίζουμε το αριστερό παιδί, στη συνέχεια τον γονέα και στο τέλος το δεξί παιδί, τυπώνοντας έτσι τις λέξεις σε αλφαβητική σειρά.

Η πολυπλοκότητα της μεθόδου είναι O(N) καθώς η ίδια η διάσχιση in-order είναι τέτοιας πολυπλοκότητας και πρέπει να διασχίσει όλα τα στοιχεία.

public void printTreebyFrequency(PrintStream stream): -

Β. Επιπλέον επεξηγηματικά σχόλια

Δεν μπορέσαμε να υλοποιήσουμε εν τέλει την **printTreebyFrequency()**, παρ'όλη την προσπάθειά μας.