Δομές Δεδομένων – Εργασία 3

ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

(**Όνομα:** Φοίβος Παπαθανασίου, **ΑΜ:** 3200138, **Email:** p3200138@aueb.gr)

ΥΛΟΠΟΙΗΣΗ ΜΕΘΟΔΩΝ

Σημείωση: Η μέθοδος load() απαιτεί το **absolute path** του .txt αρχείου.

Παρακάτω αναλύεται η υλοποίηση των μεθόδων,

update() – Η πιο σημαντική μέθοδος, λαμβάνει ως όρισμα έναν κόμβο (TreeNode) και ενημερώνει το πεδίο subtreeSize με το άθροισμα των subtreeSize των παιδιών του +1 (προφανώς αναθέτει την τιμή 1 εάν το όρισμα είναι εξωτερικός κόμβος). Η μέθοδος αυτή καλείται κάθε φορά στους κόμβους που αλλάζουν οι δείκτες τους και σε κάποιες άλλες περιπτώσεις.

insert() – Λαμβάνει ως όρισμα μία λέξη, κάνει χρήση της **insertR()** η οποία είναι αναδρομική μέθοδος που λαμβάνει ως όρισμα έναν κόμβο και τη λέξη. Περνάμε στην αναδρομική μέθοδο ως όρισμα τη ρίζα του δέντρου καθώς επίσης τη λέξη αφού την κάνουμε strip() και toLowerCase().

insertR() – Αναζητά αναδρομικά την κατάλληλη θέση για να εισαχθεί η λέξη, εάν η λέξη ήδη υπάρχει τότε δεν κάνει εισαγωγή και απλώς αυξάνει το πεδίο freq της λέξης κατά ένα (κάνωντας χρήση της μεθόδου increaseFreq() του αντικειμένου WordFreq). Στην περίπτωση που δεν υπάρχει η λέξη, κατασκευάζει νέο κόμβο που περιέχει τη νέα λέξη και την εισάγει στην κενή θέση.

Σημαντικό είναι να τονιστεί, ότι κάθε φορά πριν επιστραφεί η ρίζα κάποιου υποδέντρου, καλείται η μέθοδος update() σε αυτήν ώστε στην περίπτωση που έγινε εισαγωγή να ενημερωθεί σωστά το πεδίο subtreeSize όλων των εμπλεκόμενων κόμβων.

search() – Κάνουμε πάλι χρήση αναδρομικής μεθόδου, της **searchR()**. Καταρχάς, εάν η ρίζα είναι null επιστρέφεται null. Επίσης, ελέγχεται η περίπτωση όπου η ρίζα περιέχει το αντικείμενο ώστε να μη χρειαστεί να κληθεί η μέθοδος getMeanFrequency() (αφού θα είναι ήδη στη ρίζα). Η searchR() κλασικά επιστρέφει τον κόμβο εφόσον βρεθεί, διαφορετικά null. Ακολούθως, στη search παίρνουμε το αντικείμενο WordFreq που περιέχει ο κόμβος και εξετάζουμε εάν έχει μεγαλύτερη συχνότητα από τη μέση συχνότητα, κάνωντας κλήση της μεθόδου getMeanFrequency(). Εαν ναι, τότε κάνουμε κλήση της μεθόδου rotateToRoot() και με περιστροφές φέρνουμε τον κόμβο στη ρίζα. Τέλος, επιστρέφουμε το αντικείμενο εφόσον βρέθηκε.

getMeanFrequency() – Επιστρέφει 0 εάν το δέντρο είναι άδειο, διαφορετικά, επιστρέφει (κάνωντας cast σε double) τον λόγο του αθροίσματος όλων των συχνοτήτων (κάνωντας χρήση της αναδρομικής μεθόδου sumOfFreqR) προς το μέγεθος του δέντρου (root.subtreeSize).

sumOfFreqR() – Υπολογίζει αναδρομικά (με ενδοδιατεταγμένη διάσχιση) και επιστρέφει το άθροισμα των συχνοτήτων των αντικειμένων WordFreq ενός (υπο)δέντρου.

rotateToRoot() – Λαμβάνει ως όρισμα μία λέξη, καλεί την αναδρομική μέθοδο rotateToRootR() η οποία βρίσκει αναδρομικά την λέξη (στο σημείο που την καλούμε είναι βέβαιο ότι θα βρεί τη λέξη) και με διαδοχικές περιστροφές τη φέρνει στη ρίζα. Δεν απαιτείται ενημέρωση των subtreeSize διότι γίνεται μέσα στις rotate.

rotL() και rotR() – Υλοποιούν δεξιά και αριστερή περιστροφή, καλείται η μέθοδος update() στους εμπλεκόμενους κόμβους, πρώτα ενημερώνεται ο κόμβος που θα μπεί αριστερά/δεξιά του άλλου, και στη συνέχεια ο νέος κόμβος-ρίζα του υποδέντρου.

remove() – Εντελώς αντίστοιχη υλοποίηση με του μαθήματος, δύο μικρές προσθήκες: (1) το string που δίδεται για να αφαιρεθέι γίνεται strip() και toLowerCase(), (2) κάθε φορά, στην αναδρομική **removeR()** ενημερώνεται (κλήση της update()) ο κόμβος-ρίζα του εκάστοτε υποδέντρου πρωτού επιστραφεί. Επίσης στη βοηθητική μέθοδο **joinLR()** γίνεται κλήση της μεθόδου update() στην περίπτωση όπου “τίθεται” στον κόμβο που θα επιστραφεί αριστερό υποδέντρο.

load() – Αρχικά, εάν το αρχείο ανοίξει επιτυχώς, κατασκευάζει με τη βοήθεια της μεθόδου buildForRegex() του αντικειμένου <StringList> stopWords, ένα string το οποίο είναι το **regular expression** που θα χρησιμοποιηθεί για να αφαιρεθούν τα stopwords από τις γραμμές. Στη συνέχεια, κάνει parse της γραμμές του .txt αρχείου (του οποίου δίδεται **το absolute path**) και καλεί με αυτές ως όρισμα τη μέθοδο loadLine()

Η μέθοδος buildForRegex() της κλάσης <StringList>

Προσπελαύνει τους κόμβους που περιέχουν τα stopwords και κατασκευάζει και επιστρέφει ένα string της μορφής:
“stopword1|stopword2|stopword3|stopword2…|”

Φυσικά, εάν είναι κενή επιστρέφει “”. Επίσης, εάν τοποθετηθεί δύο φορές το ίδιο stopword, δεν υπάρχει κάποιο θέμα στο regex και άρα απλά μπορεί να υπάρξει και δύο φορές όπως και παραπάνω.

loadLine() – Η μέθοδος αυτή, αρχικά καλεί τη βοηθητική μέθοδο **filter()** με ορίσματα τη γραμμή και το stopword regex που κατασκευάστηκε στη load() και επιστρέφει και αναθέτει στη γραμμή, τη νέα γραμμή «φιλτραρισμένη». Στη συνέχεια κάνει tokenize τη γραμμή (με whitespace ως delimiter) και κάνει insert τα tokens (τις λέξεις δηλαδή) στο ΔΔΑ.

filter() – Η μέθοδος filter() λαμβάνει ως είσοδο τη γραμμή καθώς και το stopword regex και με χρήση τεσσάρων **regular expressions** καταφέρνει να φιλτράρει μία γραμμή ώστε τελικά οι λέξεις που θα εξαχθούν μετέπειτα ως tokens να **πληρούν τις απαιτούμενες προυποθέσεις** που αναγράφονται στην εκφώνηση της εργασίας.

Εφαρμόζει στη γραμμή 4 φορές τη μέθοδο replaceAll() του <String> με διαφορετικά regular expressions και ως replacement το “”.

Καταρχάς ορίζουμε το παρακάτω για διευκόλυνση:

block: το ορίζουμε ώς ένα σύνολο από non-word boundary χαρακτήρες, π.χ. στο string “som3;ae well 93ak’e” έχουμε 3 “blocks”.

Στη γραμμή πραγματοποιούνται τα εξής με τη συγκεκριμένη σειρά:

1) Αφαιρούνται όλα τα σημεία στίξης που βρίσκονται μπροστά ή πίσω από κάποιο block. π.χ. “*&@exa)mple;? some&*” -> “exa)mple some”

2) Οποιοδήποτε block περιέχει σημείο/α στίξης όπου δεν είναι ακριβώς μία μονή απόστροφος, αφαιρείται επίσης. π.χ. “that’s anoth&*er exam)ple y’e’a” ->

“that’s”

3) Οποιοδήποτε block περιέχει τουλάχιστον έναν αριθμό αφαιρείται.

4) Εντοπίζει και αφαιρεί όλα τα stopwords, case insensitive.

Σημείωση: Η δομή των regular expressions παρουσιάζεται πολύ αναλυτικά σε σχόλιο στον κώδικα.

getTotalWords() – Καλεί με όρισμα τη ρίζα του ΔΔΑ τη μέθοδο **sumOfFreqR()** και επιστρέφει την τιμή που επιστρέφεται.

getDistinctWords() – Επιστρέφει root.subtreeSize - το πλήθος κόμβων του δέντρου.

getFrequency() – Απλή αναζήτηση με χρήση αντίστοιχης αναδρομικής μεθόδου (getFrequencyR()).

getMaximumFrequency() – Εάν η ρίζα είναι άδεια, επιστρέφουμε null. Θέτουμε ως max τη ρίζα και διασχίζουμε το δέντρο ανά επίπεδο (για τον λόγο αυτό υλοποιήσαμε και μία απλή ουρά) κάνωντας συγκρίσεις. Εαν υπάρχει αντικείμενο με μεγαλύτερο freq, τότε δείχνει σε αυτό η max κ.ο.κ

removeStopWord() – Καλεί στο αντικείμενο <StringList> stopWords τη μέθοδο remove() με όρισμα την λέξη αφού γίνει strip() και toLowerCase(). Η remove() διατρέχει τη λίστα και αφαιρεί όλους τους κόμβους που περιέχουν τη λέξη που δόθηκε.

addStopWord() – Καλεί στο αντικείμενο <StringList> stopWords τη μέθοδο insert() με όρισμα την λέξη (αφού γίνει strip() και toLowerCase()). Η insert() απλώς εισάγει στην κεφαλή νέο κόμβο που περιέχει τη λέξη.

printAlphabetically() – Απλή ενδοδιατεταγμένη διάσχιση σε συνδυασμό με την υπερφόρτωση της μεθόδου toString() της κλάσης <WordFreq>.

printByFrequency() – Για τη μέθοδο αυτή, φτιάξαμε μία κλάση <WordFreqComparator> και υλοποιήσαμε τη μέθοδο compare όπου επιστρέφει τη διαφορά των frequencies δύο <WordFreq> αντικειμένων.

Επίσης, φτιάξαμε έναν private κατασκευαστή ο οποίος παίρνει ως όρισμα τον comparator. Ακόμη, φτιάξαμε τις private μεθόδους insertByFreq() (και insertByFreqR()), copyToFreqBST() (και copyToFreqBSTR()).

Πώς λειτουργεί:

Μέσα στην printByFrequency() κατασκευάζουμε ένα προσωρινό ΔΔΑ με comparator όμως ένα αντικείμενο <WordFreqComparator> όπως ορίσαμε παραπάνω. Ακολούθως, καλείται η copyToFreqBST() η οποία με χρήση της αναδρομικής αντίστοιχης προσπελαύνει το αλφαβητικά «διατεταγμένο δέντρο» με ενδοδιατεταγμένη διάσχιση και εισάγει στο κατά άυξουσα συχνότητα «διατεταγμένο δέντρο» τα αντικείμενα <WordFreq> με τη μέθοδο insertByFreq().

Μια λεπτομέρεια για τη διατήρηση σχετικότητας:

Επιλέξαμε να χρησιμοποιήσουμε ενδοδιατεταγμένη διάσχιση κατα την εισαγωγή, σε συνδυασμό με την εξής λεπτομέρεια: εάν κατά την insertByFreq() το νέο αντικείμενο που πάει να γίνει insert έχει ίδιο freq με κάποιο άλλο, τότε θα εισαχθεί στα **δεξιά** του. Έτσι εάν π.χ. έχουμε τις λέξεις “are” και “about” στο αλφαβητικά «διατεταγμένο δέντρο» και έχουν την ίδια συχνότητα, τότε πρώτα θα εισαχθεί η “about” και μετά η “are” δεξιά της στο άλλο δέντρο.

Έτσι υπάρχει και ενός είδους αλφαβητική διάταξη και όταν ταξινομούνται ως προς τη συχνότητα εμφάνισης.

ΠΟΛΥΠΛΟΚΟΤΗΤΑ (worst-case)

Σημείωση: η πολυπλοκότητα της update() είναι O(1) επομένως δεν φέρει επιρροή στην πολυπλοκότητα των μεθόδων αυτούσια.

insert() – Εξαρτάται από την πολυπλοκότητα της insertR(), η οποία είναι ανάλογη του ύψους h του δέντρου για το οποίο ισχύει logN <= h <= N-1. Δεδομένου ότι το δέντρο μπορεί να εκφυλιστεί σε λίστα η πολυπλοκότητα είναι O(N).

search() – Εξαρτάται από την πολυπλοκότητα της searchR(), η οποία είναι ανάλογη του ύψους h του δέντρου για το οποίο ισχύει logN <= h <= N-1. Δεδομένου ότι το δέντρο μπορεί να εκφυλιστεί σε λίστα η πολυπλοκότητα είναι

O(N). Σημειώνουμε ότι η πολυπλοκότητα της getMeanFrequency() είναι ίση με O(N), επίσης η πολυπλοκότητα των rotate είναι O(1) άρα στην χειρότερη περίπτωση έχουμε ~ O(N) + O(N) + (N-1)O(1) = O(N).

remove() – Εξαρτάται από την πολυπλοκότητα της removeR(), η οποία εάν ληφθούν υπόψιν και οι βοηθητικές μεθόδοι (joinLR(), partR() και τα rotates) δεν ξεπερνάει το O(N). Καθώς στη χειρότερη περίπτωση θα γίνουν αθροιστικά ~N συγκρίσεις από τις removeR() και partR(), ενώ επίσης στη χειρότερη περίπτωση θα πραγματοποιηθούν N-1 περιστροφές. Συμπερασματικά O(N).

load() – Στη χειρότερη περίπτωση, κάθε λέξη που θα υπάρχει θα είναι διαφορετική και το δέντρο θα εκφυλίσσεται σε λίστα (π.χ. αν οι λέξεις έρχονται ταξινομημένες) άρα έχουμε 1 + 2 + 3 + … + (N-1) συγκρίσεις = N(N-1)/2 = O(N^2).

getTotalWords() – Υπολογίσθηκε με τη βοηθητική μέθοδο sumOfFreqR() η οποία πραγματοποιεί ενδοδιατεταγμένη διάσχιση στο δέντρο, συνεπώς O(N).

getDistinctWords() – Επιστρέφει root.subtreeSize, O(1).

getFrequency() – Εξαρτάται από την πολυπλοκότητα της getFrequencyR(), είναι μία απλή αναζήτηση με πολυπλοκότηα O(N) στη χειρότερη περίπτωση.

getMaximumFrequency() – Μία απλή διάσχιση ανά-επίπεδο ξεκινώντας από τη ρίζα, επομένως O(N).

getMeanFrequency() – Εξαρτάται από την πολυπλοκότητα της sumOfFreqR(), η οποία είναι O(N), συμπερασματικά O(N).

addStopWord() – Πάντα εισάγουμε στην κεφαλή, άρα O(1).

removeStopWord() – Προσπελαύνεται ολόκληρη η λίστα και αφαιρείται κάθε εμφάνιση της συγκεκριμένης λέξης μέχρι να μην υπάρχει καμία. Άρα O(N).

printAlphabetically() – Απλή ενδοδιατεταγμένη διάσχιση, άρα O(N).

printByFrequency() – Η copyToFreqBSTR() διασχίζει ολόκληρο το δέντρο (με ενδοδιατεταγμένη διάσχιση) και εισάγει κάθε στοιχείο του σε ένα νέο δέντρο το οποίο μπορεί και να εκφυλιστεί σε λίστα άρα μπορεί να έχουμε ~ 1 + 2 + 3 + … + (N-1) συγκρίσεις = N(N-1)/2 = O(N^2). Έχουμε επίσης τη διάσχιση του νέου δέντρου για την εκτύπωση, όπου έχει πολυπλοκότητα O(N) άρα τελικά στη χειρότερη περίπτωση O(N^2).