ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 3

Τμήμα Πληροφορικής

Χειμερινό Εξάμηνο 2021- 2022

Διδάσκων: Ε. Μαρκάκης

Διονύσιος Ρηγάτος (3200262)

ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

Η εργασία ζητούσε την υλοποίηση ενός μετρητή λέξεων από αρχεία με την χρήση ενός Δυαδικού Δένδρου Αναζήτησης ως πίνακα συμβόλων.

Στην υλοποίηση έγινε η χρήση AVL ΔΔΑ (max height log(n)) σύμφωνα με τις διαφάνειες του μαθήματος, και για sorting χρησιμοποιήθηκε βελτιωμένη Merge Sort με την βοήθεια Insertion Sort.[1]

Γενικές σημειώσεις για τις μεθόδους:

- Η μέθοδος balance στα insert/remove έχει πολυπλοκότητα O(1) και επομένως δεν επιβαρύνει σημαντικά την ταχύτητα long-term, όπως και τα rotateLeft/rotateRight.
- Η mergeLR (που είναι στην remove) τρέχει σε χρόνο ανάλογο του ύψους αφου καλεί την αναδρομική partition η οποία σιγά σιγά φέρνει ένα Node στην ρίζα.

Ανάλυση Μεθόδων:

```
void insert(String w)
```

Η insert αναδρομικά φτάνει στην σωστή θέση που πρέπει να εισαχθεί το κλειδί Word. Στην περίπτωση όπου υπάρχει ήδη Node με το κλειδί μας, αυξάνει το frequency της λέξης κατά 1. Αλλιώς εισάγει καινούργιο Node με το κλειδί στην σωστή θέση (default Freq == 1) . Τέλος, κάνει update το subTreeSize και ισοζυγίζει το δέντρο (διατήρηση AVL ιδιότητας) καθώς γυρνάει από την αναδρομή και ανανεώνει τους συνδέσμους μεταξύ των Nodes μέχρι και την ρίζα. Το άνω όριο για insert είναι το ύψος του δέντρου, που λόγω της AVL ιδιότητας είναι logn, άρα η μέθοδος τρέχει σε O(logN).

```
TreeNode insertRoot(TreeNode current, WordFreq item)
```

Η insertRoot εισάγει έναν κόμβο στην ρίζα του δέντρου και έχει υλοποιηθεί έτσι ώστε να ξαναεισάγει συχνά χρησιμοποιημένες λέξεις στην ρίζα του δέντρου για καλό caching κάθε φορά που γίνεται αναζήτηση και μία λέξη έχει μεγαλύτερη συχνότητα από την μέση.

```
void remove(String w)
```

Η remove αναδρομικά αφαιρεί το Node με το κλειδί w από το δέντρο κατεβαίνοντας μέχρι να βρεθεί το κλειδί που θέλουμε να αφαιρέσουμε. Όταν βρεθεί, συγχωνεύει (merge) τα δύο υπόδέντρα του αφαιρετέου Node και επιστρέφει την ρίζα των υποδέντρων στην θέση του αφαιρετέου. Τέλος, ανανεώνει το subTreeSize και κάθε φορα κανει balance καθώς ανανεώνονται οι σύνδεσμοι και η αναδρομή γυρνάει πίσω. Τρέχει σε χρονο ανάλογο του ύψους

```
WordFreq search(String w)
```

Η search αναδρομικά ψάχνει το Node με το κλειδί w. Εάν βρεθεί, συγκρίνεται το frequency του Node με το meanFrequency όλου του δέντρου και στην περίπτωση που είναι μεγαλύτερο, αφαιρεί τον κόμβο από τον δέντρο και τον ξαναεισάγει στην ρίζα. Το κόστος για μια επιτυχή αναζήτηση είναι 3 φορές το ύψος του δέντρου, μια για την αναζήτηση και δύο για την αφαίρεση και επαναεισαγωγή.

```
void load(String filename)
```

Η load ανοίγει ένα buffer stream για την ανάγνωση του αρχείου. Στην συνέχεια δημιουργείται ένα RegEx pattern[2] το οποίο αναγνωρίζει μόνο λέξεις με γράμματα και αποστρόφους. Όσο υπάρχουν λέξεις που ταιριάζουν στο pattern, ελέγχει εάν βρίσκεται στο stopword list και στην συνέχεια το εισάγει (εάν δεν είναι stopword) στο ΔΔΑ. Το κόστος της συνάρτησης αυτής είναι ανάλογο του μεγέθους του αρχείου.

```
int getTotalWords()
```

Η getTotalWords έπιστρέφει τον συνολικό αριθμό λέξεων που διαβάστηκε από το αρχείο, επομένως προσθέτει όλα τα frequencies των Nodes. Αυτό γίνεται με την χρήση stack και iterative preorder traversal, προσθέτοντας στο total κάθε φορα. Σε περίπτωση άδειου δέντρου επιστρέφει 0. Το κόστος είναι ανάλογο του ύψους, αλλά έχουμε και extra memory cost λόγω του stack, που είναι αμελητέο επειδή γίνεται συνεχώς popping.

```
int getDistinctWords()
```

Η getDistinctWords επιστρέφει το subTreeSize της ρίζας, καθώς κάθε κόμβος είναι μια distinct λέξη. Κόστος O(1).

```
int getFrequency(String w)
```

Η getFrequency αναζητά την λέξη με την χρήση της Search και επιστρέφει το frequency. Πολυπλοκότητα ανάλογη της search.

```
double getMeanFrequency()
```

Η getMeanFrequency επιστρέφει το μέσο frequency των λέξεων στο δέντρο. Με την χρήση stack και iterative preorder traversal προσθέτει όλα τα frequencies των λέξεων και στο τέλος τα διαιρεί με τα Distinct Words που βρίσκονται στο δέντρο μας. Κόστος ανάλογο του ύψους.

```
WordFreq getMaximumFrequency()
```

Η getMaximumFrequency ψάχνει όλο το δέντρο iteratively και με την χρήση stack έτσι ώστε να βρει την λέξη με το μέγιστο frequency. Ανάλογη των προηγούμενων μεθόδων.

```
void addStopWord(String w)
```

Η addStopWord προσθέτει μία λέξη στην συνδεδεμένη λίστα με stopwords. Πολυπλοκότητα O(1).

```
void removeStopWord(String w)
```

Η removeStopWord κάνει traverse την συνδεδεμένη λίστα με stopwords και αφαιρεί τον κόμβο με την λέξη που θέλουμε. Πολυπλοκότητα O(n), ανάλογη του μεγέθους της λίστας με stopwords.

```
void printTreeAlphabetically(PrintStream stream)
```

Η printTreeAlphabetically κάνει traverse το δέντρο inorder με χρήση stack και iterative τρόπο. Έτσι εμφανίζει το δέντρο left-to-right, δηλαδή ταξινομημένο ως προς τα κλειδιά (λέξεις). Κόστος ανάλογο του ύψους, και έξτρα μνήμη ανάλογη του μεγέθους της στόιβας.

```
void printTreeByFrequency(PrintStream stream)
```

Η printTreeByFrequency κάνει traverse το δέντρο inorder και με την χρήση stack προσθέτει όλα τα περιεχόμενα των κόμβων του δέντρου σε έναν πίνακα. Ο πίνακας στην συνέχεια ταξινομείται με improved mergesort και εκτυπώνεται σειριακά, δηλαδή σε αύξουσα σειρά ως προς τον αριθμό εμφανίσεων. Το κόστος εδώ είναι ανάλογο του ύψος + το κόστος της Merge Sort (βλεπε header comments). Σε μνήμη, χρησιμοποιούμε ένα stack που συνήθως θα έχει μέγεθος ανάλογο του ύψους, έναν πίνακα μεγέθους ανάλογου του δέντρου και έναν aux πίνακα επίσης ανάλογου μεγέθους μεσα στην MergeSort.

[1]Την υλοποίηση της improved Merge Sort την διάβασα αναλυτικά από το βιβλίο Algorithms 4th Edition (English Edition) των Robert Sedgewick & Kevin Wayne, καθώς όπως την έδειξαν και στο ανάλογο course δομών δεδομένων. Μιας και αυτή η εργασία δεν εξετάζει την ικανότητά μας να κάνουμε Sort, είπα να εφαρμόσω μια πιο ενδιαφέρουσα υλοποίηση που έμαθα.

[2] [a-zA-Z]+'?[a-zA-Z]*

1) Από το a-Z σε μικρά και κεφαλαία

2) + : Μία η παραπάνω φορές || *: Καμία η παραπάνω φορές

3) Εάν υπάρχει ‘, προαιρετικά