ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Data Structures Project Report

Τεχνική έκθεση περιγραφής του design και του implementation

Υλοποιήθηκε από: Παπαδόπουλο Αναστάσιο

Μυρίδη Θεόδωρο

Διδάσκων: Παπαδόπουλος Απόστολος

Περιεχόμενα

1. Γενική Περιγραφή Προδιαγραφών

Για το συγκεκριμένο Project ζητήθηκε να υλοποιηθούν τρεις δομές δεδομένων: 1) απλό δυαδικό δένδρο αναζήτησης, 2) δυαδικό δένδρο αναζήτησης τύπου AVL και 3) πίνακας κατακερματισμού με ανοικτή διεύθυνση. Οι δομές αυτές θα πρέπει να αποθηκεύουν τις διαφορετικές λέξεις ενός αρχείου κειμένου και το πλήθος εμφανίσεων της κάθε λέξης μέσα στο αρχείο, ώστε έπειτα να συγκριθεί η απόδοση της κάθε δομής.

2. Εισαγωγή

2.1 Project's Background

Στόχος του project αυτού είναι η υλοποίηση σε γλώσσα C++ μερικών βασικών δομών δεδομένων, με σκοπό την αποθήκευση πληροφορίας κειμένου. Η εφαρμογή έχει υλοποιηθεί για εκδόσεις της γλώσσας από 11 και πάνω και έχει δοκιμαστεί με τα παρακάτω:

- **Dev-C++**
- **Code::Blocks IDE**
- **Visual Studio Code**

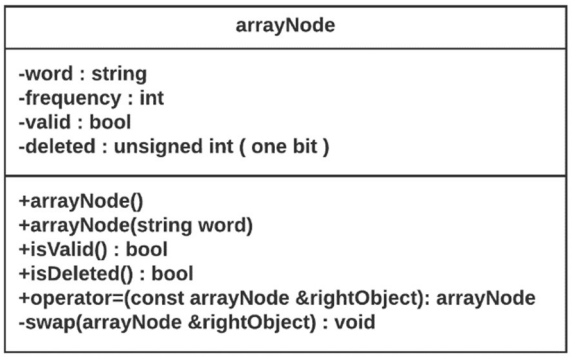
2.2 Σύντομη Περιγραφή Λειτουργικότητας

Το συγκεκριμένο project είναι ένα console application στο οποίο εισάγεται ένα αρχείο κειμένου, το application το επεξεργάζεται και έπειτα μπορούν να εκτελεστούν πράξεις εισαγωγής, αναζήτησης και διαγραφής των λέξεων ή και όχι του κειμένου στις τρεις δομές δεδομένων, καθώς και preOrder,inOrder και postOrder διασχίσεις στις δύο δομές των δένδρων. Επίσης αν το επιθυμεί ο χρήστης κάθε δομή μπορεί να χρησιμοποιηθεί ξεχωριστά ως library που υλοποιεί την εκάστοτε δομή δεδομένων για συμβολοσειρές. Χρειάζεται να γίνει **include** μόνο το αρχείο **dsHandler.h.** Επίσης τα txt αρχεία που συνοδεύουν τους πηγαίους κώδικες πρέπει να τοποθετηθούν στον ίδιο φάκελο με το εκτελέσιμο αρχείο, εκτός αν ο χρήστης αλλάξει τις τιμές των συμβολικών σταθερών που σχετίζονται με αυτά τα αρχεία. Τέλος, το αρχείο **train.cpp** περιέχει τον κώδικα που χρησιμοποιήθηκε για την εκπαίδευση του μοντέλου Γραμμικής Παλινδρόμησης ενώ το **main.cpp** την ενδεικτική συνάρτηση main που ζητήθηκε.

3. Τεχνικές Λεπτομέρειες Εφαρμογής

3.1 Περιγραφή Κλάσεων

3.1.1 arrayNode



Η κλάση **arrayNode** περιγράφει μία θέση του πίνακα κατακερματισμού.

Ιδιότητες:

word : η λέξη που αποθηκεύεται στην συγκεκριμένη θέση

frequency : η συχνότητα της word

valid : μεταβλητή που δείχνει αν η θέση είναι κατειλημμένη (ακόμα και αν η λέξη word έχει διαγραφεί valid=true)

deleted : μεταβλητή μεγέθους ενός bit που δείχνει αν η λέξη έχει διαγραφεί (για deleted = 0 η λέξη δεν έχει διαγραφεί, ενώ για deleted = 1 η λέξη έχει διαγραφεί)

Μέθοδοι:

arrayNode() : default constructor της κλάσης που αρχικοποιεί τα ιδιωτικά μέλη σε τετριμμένες τιμές

arrayNode(string word) : constructor της κλάσης που αρχικοποιεί τα μέλη με κατάλληλες τιμές (ιδιωτικό μέλος word = παράμετρος word , valid = true , frequency = 0 , deleted = 0)

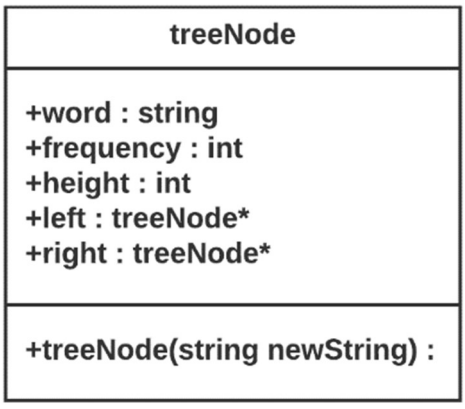
isValid() : επιστρέφει το αποτέλεσμα της σύγκρισης valid == true

isDeleted() : επιστρέφει το αποτέλεσμα της σύγκρισης deleted == 1

operator=(const arrayNode &rightObject) : υπερφόρτωση του τελεστή =

swap(arrayNode &rightObject) : μέθοδος που ανταλλάσσει τις τιμές δύο αντικειμένων της κλάσης

3.1.2 treeNode



Η κλάση **treeNode** περιγράφει έναν κόμβο είτε του δυαδικού δένδρου αναζήτησης είτε του δένδρου AVL.

Ιδιότητες:

word : η λέξη που αποθηκεύεται στον συγκεκριμένο κόμβο

frequency : η συχνότητα της word

height : το ύψος του υποδένδρου με ρίζα τον συγκεκριμένο κόμβο

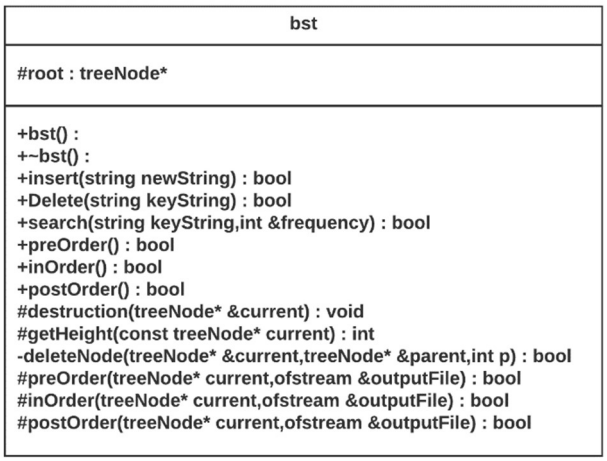
left : δείκτης στο αριστερό παιδί του συγκεκριμένου κόμβου

right : δείκτης στο δεξί παιδί του συγκεκριμένου κόμβου

Μέθοδοι:

treeNode(string newString) : constructor της κλάσης που αρχικοποιεί τα μέλη με κατάλληλες τιμές (word = newString , frequency = 1 , height = 1 , left=NULL , right = NULL)

3.1.3 bst



Η κλάση **bst**

περιγράφει την δομή ενός

δυαδικού δένδρου αναζήτησης.

Ιδιότητες:

root : δείκτης στην ρίζα του δένδρου

Μέθοδοι:

bst() : default constructor της κλάσης που αρχικοποιεί τα μέλη με κατάλληλες τιμές (root = NULL)

~bst() : destructor της κλάσης

destruction(treeNode* &current) : μέθοδος που χρησιμοποιείται από τον destructor της κλάσης για την αποδέσμευση της μνήμης του δυαδικού δένδρου. Για την αποδέσμευση χρησιμοποιείται ο παρακάτω αναδρομικός αλγόριθμος(postOrder διάσχιση με την διαφορά ότι αντί για εκτύπωση των περιεχομένων του κόμβου αποδεσμεύουμε τον κόμβο):

- Αποδέσμευσε το αριστερό παιδί
- Αποδέσμευσε το δεξί παιδί
- Αποδέσμευσε τον τωρινό κόμβο

insert(string newString) : μέθοδος που εισάγει την λέξη newString στο δυαδικό δένδρο. Για την εισαγωγή της λέξης στο δυαδικό δένδρο χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Ελέγχουμε αν η τωρινός κόμβος αποθηκεύει την ίδια λέξη με την παράμετρο newString.Αν ΝΑΙ τότε αυξάνουμε την ιδιότητα frequency του κόμβου και επιστρέφουμε τον έλεγχο στην καλούσα συνάρτηση, αλλιώς συνεχίζουμε.
- Αν η λέξη newString είναι λεξικογραφικά μικρότερη από την λέξη του τωρινού κόμβου επαναλαμβάνουμε τον αλγόριθμο για το αριστερό υποδένδρο του κόμβου. Αλλιώς, επαναλαμβάνουμε τον αλγόριθμο για το δεξί υποδένδρο του κόμβου.
- Αν σε κάποιο σημείο ο δείκτης στον κόμβο είναι ίσος με NULL εισάγουμε την λέξη σε ένα νέο κόμβο και τον συνδέουμε με το υπόλοιπο δένδρο στη σωστή κατεύθυνση(αριστερό ή δεξί παιδί του προγόνου του).

Delete(string keyString) : μέθοδος που διαγράφει την λέξη keyString από το δυαδικό δένδρο. Κάθε φορά που διαγράφεται μία λέξη ο κόμβος διαγράφεται εντελώς από το δένδρο. Για την διαγραφή της λέξης από το δυαδικό δένδρο χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Αναζητούμε την λέξη keyString στο δυαδικό δένδρο. Σε κάθε βήμα αποθηκεύουμε την διεύθυνση του γονιού του τωρινού κόμβου και αν ο τωρινός κόμβος είναι δεξί ή αριστερό παιδί του. Αν το δένδρο είναι άδειο ή σε κάποιο σημείο ο δείκτης στον κόμβο είναι ίσος με NULL, που σημαίνει ότι η λέξη δεν υπάρχει στο δένδρο , επιστρέφουμε false.
- Αν ο κόμβος που περιέχει την λέξη keyString βρεθεί τότε έχουμε 4 περιπτώσεις οι οποίες ελέγχονται στην μέθοδο **deleteNode**

deleteNode(treeNode* &current,treeNode* &parent,int p) : μέθοδος που διαγράφει έναν κόμβο από το δυαδικό δένδρο δοθέντος της διεύθυνσης του, της διεύθυνσης του γονέα του και αν είναι αριστερό (p=1) ή δεξί (p=2) παιδί του γονέα. Όπως αναφέρθηκε στην μέθοδο Delete υπάρχουν τέσσερις περιπτώσεις:

- Ο κόμβος που διαγράφουμε δεν έχει παιδιά. Τότε απλώς διαγράφουμε τον κόμβο και ανάλογα με το αν ήταν δεξί ή αριστερό παιδί θέτουμε τον αντίστοιχο δείκτη του γονιού ίσο με NULL
- Ο κόμβος που διαγράφουμε έχει μόνο αριστερό παιδί. Τότε ανάλογα με το αν είναι αριστερό ή δεξί παιδί θέτουμε τον αντίστοιχο δείκτη του γονιού να δείχνει στο αριστερό παιδί του τωρινού κόμβου και τον διαγράφουμε.
- Ο κόμβος που διαγράφουμε έχει μόνο δεξί παιδί. Παρόμοια με το αν ο κόμβος έχει μόνο αριστερό παιδί θέτουμε τον αντίστοιχο δείκτη του γονιού να δείχνει στο δεξί παιδί του τωρινού κόμβου και τον διαγράφουμε.
- Ο κόμβος έχει δύο παιδιά. Τότε βρίσκουμε τον κόμβο(έστω **nextOrdered)** που είναι ο επόμενος του κόμβου που διαγράφουμε(έστω **current**) στην inOrder διάσχιση, αντιγράφουμε τα δεδομένα του **nextOrdered**(word και frequency) στα δεδομένα του **current** και τελικά διαγράφουμε τον **nextOrdered** (επειδή ο κόμβος αυτός είναι ο επόμενος του **current** στην inOrder διάσχιση η διαγραφή του ανάγεται στις περιπτώσεις διαγραφής που ο κόμβος έχει είτε μόνο ένα παιδί είτε δεν έχει κανένα).

search(string keyString,int &frequency) : μέθοδος που αναζητά την λέξη keyString στο δυαδικό δένδρο και αποθηκεύει το πόσες φορές εμφανίζεται μέσα στο κείμενο στην

μεταβλητή frequency. .Για την αναζήτηση της λέξης στο δυαδικό δένδρο χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Ελέγχουμε αν ο τωρινός κόμβος περιέχει την λέξη keyString.Αν ΝΑΙ τότε αποθηκεύουμε πόσες φορές εμφανίζεται στην αναφορική μεταβλητή frequency και επιστρέφουμε true. Αλλιώς συνεχίζουμε.
- Αν η λέξη που αναζητούμε είναι λεξικογραφικά μικρότερη επαναλαμβάνουμε τον αλγόριθμο για το αριστερό υποδένδρο του κόμβου, αλλιώς επαναλαμβάνουμε τον αλγόριθμο για το δεξί υποδένδρο. Αν σε κάποιο σημείο ο δείκτης στον κόμβο είναι ίσος με NULL τότε θέτουμε την αναφορική μεταβλητή ίση με 0 και επιστρέφουμε false γιατί η λέξη keyString δεν υπάρχει στο δυαδικό δένδρο

preOrder() : μέθοδος που τυπώνει την preOrder διάσχιση του δυαδικού δένδρου. Η εκτύπωση της preOrder πραγματοποιείται στο αρχείο με όνομα **BST_PRE_ORDER_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **BST.h**). Επίσης χρησιμοποιείται η private μέθοδος preOrder για την αναδρομική εκτέλεση του παρακάτω αλγορίθμου:

- Εκτύπωσε το περιεχόμενο του τωρινού κόμβου
- Εκτύπωσε το περιεχόμενο του αριστερού παιδιού
- Εκτύπωσε το περιεχόμενο του δεξιού παιδιού

preOrder(treeNode* current,ofstream &outputFile) : μέθοδος που χρησιμοποιείται για την εκτύπωση της preOrder διάσχισης του δυαδικού δένδρου σε txt αρχείο

inOrder() : μέθοδος που τυπώνει την inOrder διάσχιση του δυαδικού δένδρου. Η εκτύπωση της inOrder πραγματοποιείται στο αρχείο με όνομα **BST_IN_ORDER_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **BST.h**). Επίσης χρησιμοποιείται η private μέθοδος inOrder για την αναδρομική εκτέλεση του παρακάτω αλγορίθμου:

- Εκτύπωσε το περιεχόμενο του αριστερού παιδιού
- Εκτύπωσε το περιεχόμενο του τωρινού κόμβου
- Εκτύπωσε το περιεχόμενο του δεξιού παιδιού

inOrder(treeNode* current,ofstream &outputFile) : μέθοδος που χρησιμοποιείται για την εκτύπωση της inOrder διάσχισης του δυαδικού δένδρου σε txt αρχείο

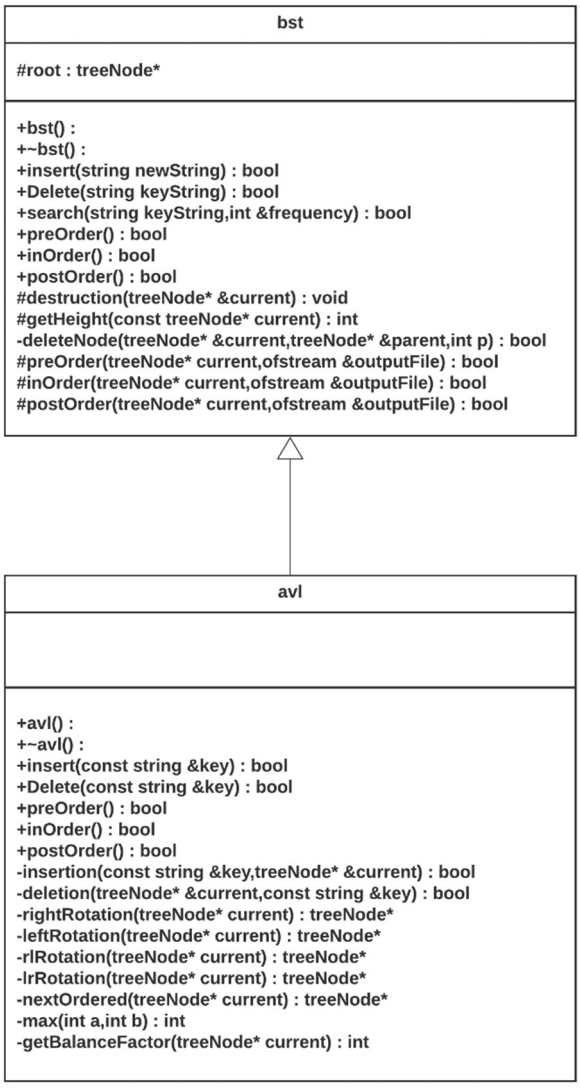
postOrder() : μέθοδος που τυπώνει την postOrder διάσχιση του δυαδικού δένδρου. Η εκτύπωση της postOrder πραγματοποιείται στο αρχείο με όνομα **BST_POST_ORDER_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **BST.h**). Επίσης χρησιμοποιείται η private μέθοδος postOrder για την αναδρομική εκτέλεση του παρακάτω αλγορίθμου:

- Εκτύπωσε το περιεχόμενο του αριστερού παιδιού
- Εκτύπωσε το περιεχόμενο του δεξιού παιδιού
- Εκτύπωσε το περιεχόμενο του τωρινού κόμβου

postOrder(treeNode* current,ofstream &outputFile) : μέθοδος που χρησιμοποιείται για την εκτύπωση της postOrder διάσχισης του δυαδικού δένδρου σε txt αρχείο

getHeight(const treeNode* current) : μέθοδος που επιστρέφει το ύψος του κόμβου current

3.1.4 avl



Η κλάση **avl**

περιγράφει την δομή ενός

δένδρου AVL.

Η κλάση **avl** κληρονομεί public από την κλάση **bst.**Επομένως, έχει πρόσβαση στις public και protected ιδιότητες και μεθόδους της κλάσης **bst** και κάνει **override** κάποιες από αυτές τις μεθόδους. Οι μέθοδοι αυτές είναι δηλωμένες στην κλάση **bst** ως virtual ώστε να εκτελούνται οι σωστές μέθοδοι της παράγωγης κλάσης σε περίπτωση που χρησιμοποιηθούν δείκτες στην βασική κλάση για την δημιουργία αντικειμένων **avl.**

Μέθοδοι:

avl() : default constructor της κλάσης

~avl() : destructor της κλάσης

preOrder() : μέθοδος που τυπώνει την preOrder διάσχιση του δένδρου AVL. Η εκτύπωση της preOrder πραγματοποιείται στο αρχείο με όνομα **AVL_PRE_ORDER_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **AVL.h**)

inOrder() : μέθοδος που τυπώνει την inOrder διάσχιση του δένδρου AVL. Η εκτύπωση της inOrder πραγματοποιείται στο αρχείο με όνομα **AVL_IN_ORDER_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **AVL.h**)

postOrder() : μέθοδος που τυπώνει την postOrder διάσχιση του δένδρου AVL. Η εκτύπωση της postOrder πραγματοποιείται στο αρχείο με όνομα **AVL_POST_ORDER_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **AVL.h**)

insert(const string &key) : μέθοδος που εισάγει την λέξη key στο δένδρο avl χρησιμοποιώντας την private μέθοδο insertion

insertion(const string &key,treeNode* &current) : private μέθοδος που χρησιμοποιείται από την public insert. Για την εισαγωγή της λέξης στο δένδρο avl χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Ελέγχουμε αν ο τωρινός κόμβος αποθηκεύει την ίδια λέξη με την παράμετρο key. Αν ΝΑΙ τότε αυξάνουμε την ιδιότητα frequency του κόμβου, κάνουμε update το height του και επιστρέφουμε τον έλεγχο στην καλούσα συνάρτηση.
- Αν η λέξη newString είναι λεξικογραφικά μικρότερη από την λέξη του τωρινού κόμβου επαναλαμβάνουμε τον αλγόριθμο για το αριστερό υποδένδρο του κόμβου. Αλλιώς, επαναλαμβάνουμε τον αλγόριθμο για το δεξί υποδένδρο του κόμβου.
- Αν σε κάποιο σημείο ο δείκτης στον κόμβο είναι ίσος με NULL εισάγουμε την λέξη σε ένα νέο κόμβο και τον συνδέουμε με το υπόλοιπο δένδρο στη σωστή κατεύθυνση(αριστερό ή δεξί παιδί του προγόνου του).
- Όταν τελειώσει η εισαγωγή της λέξης στο δένδρο avl κάνουμε update το height του κόμβου που έγινε η εισαγωγή και ελέγχουμε το μονοπάτι των κόμβων από αυτόν μέχρι την ρίζα για τυχόν ανισορροπίες που μπορεί να έχουν δημιουργηθεί και τις διορθώνουμε. Ορίζουμε αρχικά τον παράγοντα ισορροπίας ενός κόμβου ως την διαφορά του ύψους του αριστερού του παιδιού μείον το ύψος του δεξιού του παιδιού(Παρακάτω για ευκολία αναφέρεται ως **balanceFactor)**. Οι περιπτώσεις είναι οι εξής τέσσερις για έναν τυχαίο κόμβο (έστω **current)** :
    - Αν **balanceFactor** == 2 και η λέξη key είναι λεξικογραφικά μικρότερη από την λέξη του αριστερού παιδιού του κόμβου **current** τότε έχουμε μια ανισορροπία left-left και την διορθώνουμε με μία right περιστροφή. Αλλιώς έχουμε μια ανισορροπία left-right και την διορθώνουμε με μία left-right περιστροφή.
    - Αν **balanceFactor** == -2 και η λέξη key είναι λεξικογραφικά μεγαλύτερη από την λέξη του δεξιού παιδιού του κόμβου **current** τότε έχουμε μια ανισορροπία right-right και την διορθώνουμε με μία left περιστροφή. Αλλιώς έχουμε μια ανισορροπία right-left και την διορθώνουμε με μία right-left περιστροφή.

Delete(const string &key) : μέθοδος που διαγράφει την λέξη key από το δένδρο avl χρησιμοποιώντας την private μέθοδο deletion

deletion(treeNode* & current,const string &key) : private μέθοδος που χρησιμοποιείται από την public Delete. Κάθε φορά που διαγράφεται μία λέξη ο κόμβος διαγράφεται εντελώς από το δένδρο. Για την διαγραφή της λέξης από το δένδρο avl χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Αναζητούμε την λέξη στο δένδρο avl. Αν το δένδρο είναι άδειο ή σε κάποιο σημείο ο δείκτης στον κόμβο είναι ίσος με NULL, που σημαίνει ότι η λέξη δεν υπάρχει στο δένδρο , επιστρέφουμε false.
- Αν ο κόμβος που περιέχει την λέξη keyString βρεθεί τότε πραγματοποιούμε την διαγραφή όπως στο απλό δυαδικό δένδρο. Αφού ο κόμβος διαγραφεί κάνουμε update το height του κόμβου που ίσως έχει αντικαταστήσει τον προηγούμενο που περιείχε την λέξη key και ελέγχουμε το μονοπάτι των κόμβων από αυτόν (μπορεί να μην έχει αντικατασταθεί από κανέναν κόμβο ανάλογα ποια περίπτωση ίσχυε από αυτές που αναφέρονται στην μέθοδο **deleteNode** της κλάσης **bst)** μέχρι την ρίζα για τυχόν ανισορροπίες που μπορεί να έχουν δημιουργηθεί και τις διορθώνουμε. Οι περιπτώσεις είναι τέσσερις για έναν τυχαίο κόμβο (έστω **current**):
    - Αν **balanceFactor** > 1 και ο **balanceFactor** του αριστερού παιδιού του **current** είναι μεγαλύτερος ή ίσος του μηδέν τότε έχουμε μια ανισορροπία left-left και την διορθώνουμε με μία right περιστροφή. Αλλιώς αν ο **balanceFactor** του αριστερού παιδιού είναι αρνητικός έχουμε μία left-right ανισορροπία και την διορθώνουμε με μία left-right περιστροφή.
    - Αν **balanceFactor** < -1 και ο **balanceFactor** του δεξιού παιδιού του **current** είναι μικρότερος ή ίσος του μηδέν τότε έχουμε μια ανισορροπία right-right και την διορθώνουμε με μία left περιστροφή. Αλλιώς αν ο **balanceFactor** του δεξιού παιδιού είναι θετικός έχουμε μία right-left ανισορροπία και τη διορθώνουμε με μία right-left περιστροφή.

rightRotation(treeNode* current) : μέθοδος που πραγματοποιεί δεξιά περιστροφή στον κόμβο current

leftRotation(treeNode* current) : μέθοδος που πραγματοποιεί αριστερή περιστροφή στον κόμβο current

rlRotation(treeNode* current) : μέθοδος που πραγματοποιεί διπλή δεξιά-αριστερή περιστροφή στον κόμβο current

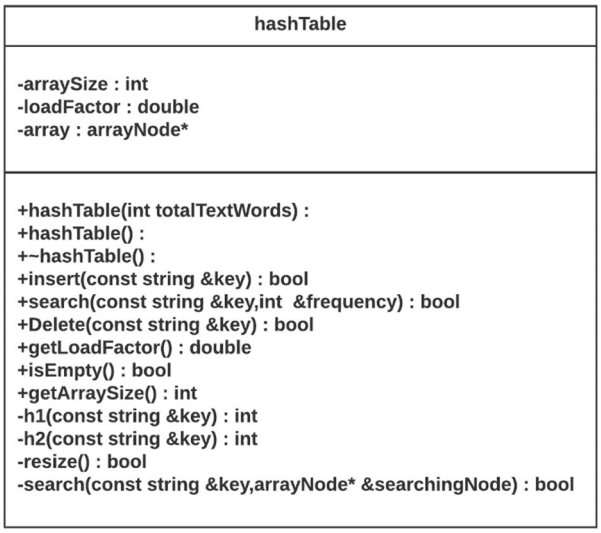
lrRotation(treeNode* current) : μέθοδος που πραγματοποιεί διπλή αριστερή-δεξιά περιστροφή στον κόμβο current

nextOrdered(treeNode* current) : μέθοδος που επιστρέφει τον κόμβο που είναι ο επόμενος του κόμβου current στην inOrder διάσχιση του δένδρου avl

max(int a,int b) : μέθοδος που επιστρέφει τον μεγαλύτερο από τους αριθμούς a και b

getBalanceFactor(treeNode* current) : μέθοδος που επιστρέφει την διαφορά του ύψους του αριστερού υποδένδρου του κόμβου current με το δεξί του υποδένδρο

3.1.5 hashTable



Η κλάση **hashTable** περιγράφει την δομή ενός πίνακα κατακερματισμού με ανοικτή διεύθυνση που χρησιμοποιεί την μέθοδο του double hashing για την αντιμετώπιση των collisions

Ιδιότητες:

arraySize : το μέγεθος του πίνακα κατακερμαστισμού

loadFactor : ο συντελεστής πληρότητας του πίνακα κατακερματισμού

array : δείκτης για την αποθήκευση του πίνακα κατακερματισμού

Μέθοδοι:

hashTable(int totalTextWords) : constructor που δεσμεύει χώρο για τον πίνακα κατακερματισμού χρησιμοποιώντας τις συνολικές λέξεις του κειμένου που δέχεται σαν όρισμα και το μοντέλο Γραμμικής Παλινδρόμησης (βλέπε 3.4) για να υπολογίσει το μέγεθος του πίνακα και να πραγματοποιήσει τις απαραίτητες αρχικοποιήσεις

hashTable() : default constructor που δεσμεύει χώρο σε μια συγκεκριμένη αρχική τιμή και αν ο πίνακας κατακερματισμού χρησιμοποιηθεί για παραπάνω λέξεις θα αυξηθεί το μέγεθος του αργότερα

~hashTable() : destructor της κλάσης

insert(const string &key) : μέθοδος που εισάγει την λέξη key στον πίνακα κατακερματισμού. Για την εισαγωγή χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Υπολογίζουμε την τιμή **h1(key)** και διατρέχουμε τον πίνακα κατά **i*h2(key)** θέσεις κάθε φορά (**i>=0**) όσο οι θέσεις που ελέγχουμε είναι κατειλημμένες (δηλαδή την ιδιότητα **valid** της θέσης, ακόμα κι αν οι λέξεις που υπάρχουν σε κάποιες από αυτές τις θέσεις που ελέγχουμε έχουν διαγραφεί).
- Αν κάποια από τις θέσεις που ελέγχουμε αποθηκεύει λέξη λεξικογραφικά μεγαλύτερη από την λέξη **key** τότε αντιμεταθέτουμε τα στοιχεία και πλέον συνεχίζουμε την εισαγωγή με την λέξη που υπήρχε σε εκείνη την θέση. Με τον τρόπο αυτό διατηρούμε κάθε ακολουθία κλειδιών σε αύξουσα ταξινομημένη

διάταξη, έτσι μειώνουμε τον χρόνο που απαιτείται για μία μη επιτυχημένη αναζήτηση περίπου στον χρόνο που απαιτείται για μία επιτυχημένη αναζήτηση.

- Αν σε κάποια θέση υπάρχει ήδη αποθηκευμένη η λέξη **key** τότε θέτουμε την ιδιότητα **deleted** της θέσης σε 0 και αυξάνουμε την ιδιότητα **frequency** κατά 1. Έτσι αν η λέξη **key** είχε διαγραφεί και τώρα ξαναεισάγεται το **frequency** της θέσης θα γίνει 1 (αφού είχε γίνει 0 στην μέθοδο **Delete**) και δεν θα εμφανίζεται πλέον σαν μία λέξη που έχει διαγραφεί, ενώ σε διαφορετική περίπτωση απλώς αυξάνεται το **frequency** της θέσης.
- Αν σε κάποιο σημείο η θέση είναι άδεια τότε εισάγουμε την λέξη **key** στην συγκεκριμένη θέση και αυξάνουμε τον **loadFactor** του πίνακα κατακερματισμού.
- Αν επιστρέψουμε στην ίδια θέση από εκεί που ξεκινήσαμε τότε ο πίνακας έχει γεμίσει.

search(const string &key,int &frequency) : μέθοδος που αναζητά την λέξη key στον πίνακα κατακερματισμού και αποθηκεύει το πόσες φορές εμφανίζεται στο κείμενο στην αναφορική μεταβλητή frequency.Για την αναζήτηση χρησιμοποιείται ο παρακάτω αλγόριθμος:

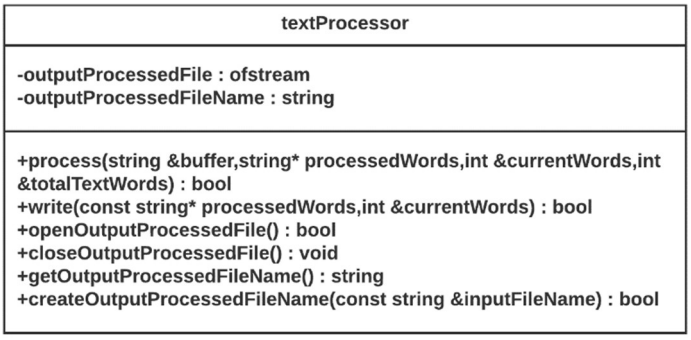
- Υπολογίζουμε τις τιμές **h1(key)** και **h2(key)** και διατρέχουμε τον πίνακα κατά **i*h2(key)** θέσεις κάθε φορά (**i>=0**) όσο οι θέσεις που ελέγχουμε είναι κατειλημμένες (δηλαδή την ιδιότητα **valid** της θέσης).
- Αν σε κάποια θέση βρούμε την λέξη **key** αποθηκεύουμε το **frequency** της θέσης στην αναφορική μεταβλητή και επιστρέφουμε το **!deleted** της θέσης. Έτσι, αν η λέξη έχει διαγραφεί η αναφορική μεταβλητή θα έχει την τιμή 0 και θα επιστραφεί false, ενώ σε διαφορετική περίπτωση η αναφορική μεταβλητή θα έχει το πόσες φορές υπάρχει η λέξη **key** στο κείμενο μέχρι την στιγμή της αναζήτησης και θα επιστραφεί true.
- Αν επιστρέψουμε στην αρχική θέση, βρούμε κάποια θέση με λεξικογραφικά μεγαλύτερη λέξη από την λέξη **key** ή μία άδεια θέση θέτουμε την αναφορική μεταβλητή ίση με 0 και επιστρέφουμε false, γιατί αν η λέξη **key** υπήρχε στον πίνακα θα έπρεπε να είχε βρεθεί σε κάποια από τις προηγούμενες θέσεις.

Delete(const string &key) (**Δεν έχει ζητηθεί στις προδιαγραφές της εφαρμογής αλλά έχει υλοποιηθεί**) **:** μέθοδος που διαγράφει την λέξη key από τον πίνακα κατακερματισμού. Κάθε φορά που διαγράφεται μία λέξη το frequency της μηδενίζεται. Για την διαγραφή της λέξης χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Υπολογίζουμε τις τιμές **h1(key)** και **h2(key)** και διατρέχουμε τον πίνακα κατά **i*h2(key)** θέσεις κάθε φορά (**i>=0**) όσο οι θέσεις που ελέγχουμε είναι κατειλημμένες (δηλαδή την ιδιότητα **valid** της θέσης) μέχρι να βρούμε είτε την θέση που περιέχει την λέξη **key** ή μια θέση που περιέχει μία λεξικογραφικά μεγαλύτερη λέξη. Αν επιστρέψουμε στην αρχική θέση αυτό σημαίνει ότι η λέξη **key** δεν υπάρχει στον πίνακα και συνεπώς επιστρέφουμε false.
- Σε διαφορετική περίπτωση ελέγχουμε αν η θέση που σταματήσαμε την αναζήτηση περιέχει την λέξη **key** και αν ναι την διαγράφουμε θέτοντας την ιδιότητα **deleted** σε 1, την ιδιότητα **frequency** σε 0 και επιστρέφουμε true, αλλιώς επιστρέφουμε false γιατί η λέξη δεν υπάρχει στον πίνακα κατακερματισμού.

- **getLoadFactor() :** μέθοδος που επιστρέφει τον συντελεστή πληρότητας του πίνακα κατακερματισμού
- **isEmpty() :** μέθοδος που ελέγχει αν ο πίνακας είναι άδειος
- **getArraySize() :** μέθοδος που επιστρέφει το μέγεθος του πίνακα κατακερματισμού
- **h1(const string &key) :** 1η συνάρτηση κατακερματισμού
- **h2(const string &key) :** 2η συνάρτηση κατακερματισμού
- **resize() :** μέθοδος που αυξάνει το μέγεθος του πίνακα κατακερματισμού
- **search(const string &key,arrayNode* &searchingNode) :** μέθοδος που χρησιμοποιείται από την μέθοδο **resize** για να ελέγξει αν έχει πραγματοποιηθεί σωστά η αύξηση του μεγέθους του πίνακα κατακερματισμού και για να αποθηκευτεί η σωστή τιμή της ιδιότητας **frequency κάθε** θέσης στον νέο μεγαλύτερο πίνακα κατακερματισμού.

3.1.6 textProcessor



Η κλάση **textProcessor** περιγράφει έναν επεξεργαστή κειμένου ο οποίος υλοποιεί την επεξεργασία του αρχικού txt αρχείου και την δημιουργία του νέου που περιέχει τις επεξεργασμένες λέξεις.

Ιδιότητες:

outputProcessedFile : το αρχείο στο οποίο θα αποθηκευτούν οι επεξεργασμένες λέξεις

outputProcessedFileName : το όνομα του αρχείου στο οποίο θα αποθηκευτούν οι επεξεργασμένες λέξεις

Μέθοδοι:

process(string &buffer,string* processedWords,int &currentWords,int &totalTextWords) : μέθοδος που δέχεται ως είσοδο μια συμβολοσειρά buffer η οποία έχει διαβαστεί από το αρχικό txt αρχείο, την επεξεργάζεται, αποθηκεύει τις επιμέρους λέξεις στον πίνακα processedWords και ενημερώνει κατάλληλα τις μεταβλητές currentWords (για να πραγματοποιηθεί η αποθήκευση των λέξεων στο νέο αρχείο) και totalTextWords (ώστε να γνωρίζει ο fileHandler (βλέπε 3.1.7) πόσες λέξεις περιέχει το εκάστοτε αρχείο με το οποίο

πραγματοποιήθηκε η εκτέλεση). Ως λέξη θεωρούμε οποιαδήποτε ακολουθία που περιέχει μόνο χαρακτήρες του αγγλικού αλφαβήτου, αν κάποια λέξη περιέχει κεφαλαίους χαρακτήρες τότε αυτοί μετατρέπονται σε μικρούς (Για παράδειγμα η λέξη protected θεωρείται ίδια με την λέξη Protected ή την λέξη ProtecteD). Για την επεξεργασία των λέξεων χρησιμοποιείται ο παρακάτω αλγόριθμος:

- Μετατρέπουμε όλους τους κεφαλαίους χαρακτήρες (αν αυτοί υπάρχουν) στους αντίστοιχος μικρούς
- Σαρώνουμε το **buffer** μέχρι να βρούμε την εμφάνιση του πρώτου αγγλικού χαρακτήρα, αποθηκεύουμε αυτή την θέση (έστω **startingIndex**) και συνεχίζουμε τη σάρωση όσο συναντάμε αγγλικούς χαρακτήρες.
- Αν βρεθεί κάποιος μη αγγλικός χαρακτήρας σταματάμε την σάρωση και αποθηκεύουμε την συμβολοσειρά από την θέση **startingIndex** μέχρι την θέση που σταματήσαμε και επαναλαμβάνουμε τον αλγόριθμο για το υπόλοιπο μέρος του **buffer**.

write(const string* processedWords,int &currentWords) : μέθοδος που αποθηκεύει τις επεξεργασμένες λέξεις στο νέο txt αρχείο

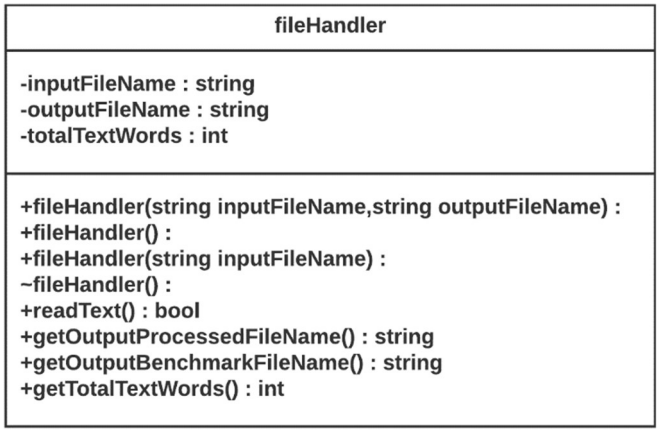
openOutputProcessedFile() : μέθοδος που ανοίγει το νέο αρχείο txt στο οποίο θα αποθηκευτούν οι επεξεργασμένες λέξεις

closeOutputProcessedFile() : μέθοδος που κλείνει το νέο txt αρχείο

getOutputProcessedFileName() : μέθοδος που επιστρέφει το όνομα του txt αρχείου που περιέχει τις επεξεργασμένες λέξεις

createOutputProcessedFileName(const string &inputFileName) : μέθοδος που δημιουργεί το όνομα του αρχείου στο οποίο θα αποθηκευτούν οι επεξεργασμένες λέξεις προσθέτοντας στο τέλος του ονόματος την λέξη processed (π.χ. αν το αρχικό αρχείο ονομάζεται “**small-file.txt**"» τότε το νέο αρχείο με τις επεξεργασμένες λέξεις θα ονομάζεται “**small-fileprocessed.txt**")

3.1.7 fileHandler



Η κλάση **fileHandler** περιγράφει έναν διαχειριστή αρχείων ο οποίος χειρίζεται τα διάφορα αρχεία που χρησιμοποιούνται κατά την διάρκεια της εκτέλεσης της εφαρμογής

Ιδιότητες:

inputFileName : το όνομα του αρχικού txt αρχείου

outputFileName : το όνομα του txt αρχείου στο οποίο θα αποθηκευτούν οι επιδόσεις των δομών και ο αριθμός των εμφανίσεων της κάθε λέξης που αναζητήθηκε μέσω της κλάσης dsHandler (βλέπε 3.1.8)

totalTextWords : οι συνολικές επεξεργασμένες λέξεις του αρχικού txt αρχείου

Μέθοδοι:

fileHandler(string inputFileName,string outputFileName) : constructor της κλάσης που αρχικοποιεί τα ιδιωτικά μέλη με τις τιμές των αντίστοιχων παραμέτρων που δέχεται

fileHandler() : default constructor της κλάσης που διαβάζει από το πληκτρολόγιο τις τιμές των ιδιωτικών μελών

fileHandler(string inputFileName) : constructor της κλάσης που δέχεται σαν όρισμα μόνο το όνομα του αρχικού txt αρχείου και αρχικοποιεί το όνομα του αρχείου εξόδου ίσο με την συμβολική σταθερά **OUTPUT_FILE_NAME** που δηλώνεται μέσα στο αρχείο **fileHandler.h**

~fileHandler() : destructor της κλάσης που διαγράφει το αρχείο με τις επεξεργασμένες λέξεις

readText() : μέθοδος που διαβάζει το αρχικό αρχείο για να το επεξεργαστεί και αποθηκεύει τις επεξεργασμένες λέξεις στο νέο txt αρχείο. Για την ανάγνωση και επεξεργασία του αρχείου χρησιμοποιείται ο παρακάτω αλγόριθμος:

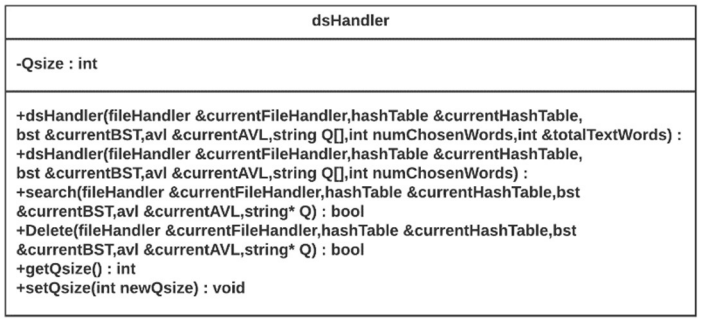
- Δημιουργούμε το όνομα του νέου αρχείου (βλέπε 3.1.6) , ανοίγουμε το αρχείο εισόδου και το αρχείο στο οποίο θα αποθηκευτούν οι επεξεργασμένες λέξεις και χρησιμοποιώντας έναν **textProcessor** και την μέθοδο **process** (βλέπε 3.1.6) πραγματοποιούμε την επεξεργασία του αρχείου. Κατά την διάρκεια της επεξεργασίας υπολογίζουμε τον χρόνο που χρειάστηκε και τον εκτυπώνουμε στο αρχείο με όνομα **outputFileName** (ιδιότητα της κλάσης **fileHandler**)

getOutputProcessedFileName() : μέθοδος που επιστρέφει το όνομα του αρχείου που αποθηκεύονται οι επεξεργασμένες λέξεις

getOutputBenchmarkFileName() : μέθοδος που επιστρέφει το όνομα του αρχείου στο οποίο θα αποθηκευτούν οι αποδόσεις των δομών

getTotalTextWords() : μέθοδος που επιστρέφει τον συνολικό αριθμό επεξεργασμένων λέξεων που περιείχε το αρχείο με το οποίο έγινε η εκτέλεση

3.1.8 dsHandler



Η κλάση **dsHandler** περιγράφει έναν διαχειριστή των δομών ο οποίος εκτελεί σε αυτές, τις πράξεις της εισαγωγής, αναζήτησης και διαγραφής

Ιδιότητες:

Qsize : το μέγεθος του συνόλου Q το οποίο περιέχει τις λέξεις με τις οποίες θα εκτελεστούν οι διάφορες πράξεις στις δομές

Μέθοδοι:

dsHandler(fileHandler &currentFileHandler,hashTable &currentHashTable,bst &currentBST,avl &currentAVL,string Q[],int numChosenWords,int &totalTextWords :

constructor της κλάσης που πραγματοποιεί τις εισαγωγές στις τρεις δομές χρησιμοποιώντας τις αντίστοιχες **insert** μεθόδους της κάθε κλάσης, αποθηκεύει τις τυχαίες λέξεις στον πίνακα Q και αποθηκεύει τον συνολικό αριθμό των επεξεργασμένων λέξεων στην παράμετρο **totalTextWords**. Κατά την διάρκεια της εισαγωγής υπολογίζουμε τον χρόνο που χρειάστηκε και τον εκτυπώνουμε στο αρχείο με όνομα **outputFileName** (ιδιότητα της κλάσης **fileHandler**)

dsHandler(fileHandler &currentFileHandler,hashTable &currentHashTable,bst &currentBST,avl &currentAVL,string Q[],int numChosenWords) : constructor της κλάσης παρόμοιος με τον προηγούμενο με την διαφορά ότι δεν αποθηκεύεται σε αναφορική μεταβλητή ο συνολικός αριθμός των επεξεργασμένων λέξεων

search(fileHandler &currentFileHandler,hashTable &currentHashTable,bst &currentBST,avl &currentAVL,string* Q) : μέθοδος που αναζητά τις λέξεις του πίνακα Q στις τρεις δομές χρησιμοποιώντας τις αντίστοιχες **search** μεθόδους της κάθε κλάσης και εκτυπώνει την κάθε λέξη που αναζητείται και το **frequency** της στο αρχείο με όνομα **outputFileName** (ιδιότητα της κλάσης **fileHandler**). Επίσης εκτυπώνεται ο χρόνος που χρειάστηκε κάθε δομή για να ολοκληρώσει την αναζήτηση όλων των λέξεων του συνόλου Q.

Delete(fileHandler &currentFileHandler,hashTable &currentHashTable,bst &currentBST,avl &currentAVL,string* Q) : μέθοδος που διαγράφει τις λέξεις του πίνακα Q από τις τρεις δομές χρησιμοποιώντας τις αντίστοιχες **Delete** μεθόδους της κάθε κλάσης και εκτυπώνει τον χρόνο που χρειάστηκε κάθε δομή για να ολοκληρώσει την διαγραφή όλων των λέξεων.

getQsize() : μέθοδος που επιστρέφει το μέγεθος του συνόλου Q

setQsize(int newQsize) : setter για το ιδιωτικό μέλος Qsize

3.2 Περιγραφή Συναρτήσεων Εκτός Κλάσεων

3.2.1 fileHandler.h

printError(int errorCode) : Συνάρτηση που εκτυπώνει στο αρχείο με όνομα **LOG_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **fileHandler.h**) ανάλογα την τιμή του errorCode (συμβολικές σταθερές δηλωμένες στο **fileHandler.h**) τι **error** συνέβη κατά την διάρκεια της εκτέλεσης (αν έχει συμβεί).

3.2.2 hashTable.h

calculateSize(int totalTextWords) : συνάρτηση που χρησιμοποιώντας το μοντέλο Γραμμικής Παλινδρόμησης και την συνάρτηση **calculateNextPrime** (βλέπε 3.4) υπολογίζει το μέγεθος του **hashTable** για το αρχείο που επιλέχθηκε για την τωρινή εκτέλεση της εφαρμογής.

calculateNextPrime(int approxSize) : συνάρτηση που βρίσκει τον αμέσως επόμενο πρώτο αριθμό που είναι μεγαλύτερος ή ίσος του **approxSize** χρησιμοποιώντας το αρχείο με όνομα **PRIMES_FILE_NAME** (συμβολική σταθερά δηλωμένη στο **hashTable.h,** το αρχικό αρχείο που έχει φτιαχτεί με πρώτους αριθμούς μέχρι και το 1001003 έχει όνομα **primes.txt**).

3.2.3 textProcessor.h

convertToLower(string &currentWord) : συνάρτηση που μετατρέπει όλους τους κεφαλαίους (αν υπάρχουν) αγγλικούς χαρακτήρες της συμβολοσειράς σε μικρούς

isEnglishLetter(char element) : συνάρτηση που ελέγχει αν ο χαρακτήρας **element** είναι χαρακτήρας του αγγλικού αλφαβήτου (είτε κεφαλαίος είτε μικρός).

3.3 Επιλογή Συνδυασμού Συναρτήσεων Κατακερματισμού

Για την επιλογή Συναρτήσεων Κατακερματισμού δοκιμάστηκαν τέσσερις συνδυασμοί της συνάρτησης κατακερματισμού **sdbm** (βλέπε παρακάτω) με άλλες 4 διαφορετικές συναρτήσεις κατακερματισμού.

sdbm

```
unsigned long long hash =0;
int i=0;
while ( key[i]!='\0' )
    hash = ((int)key[i++]) + ( hash<<6 ) + ( hash<<16 ) - hash;
return hash % arraySize;
```

djb2 version 1

```
unsigned long long int hash = 5381;
int i=0;
while ( key[i]!='\0' )
    hash = ( ( hash<<5 ) + hash ) + ((int)key[i++]);


return hash % arraySize;
```

djb2 version 2

```
unsigned long long int hash = 5381;
int i=0;
while ( key[i]!='\0' )
    hash = ( ( hash<<5 ) + hash ) ^ ((int)key[i++]);

return hash % arraySize;
```

hash Function 6

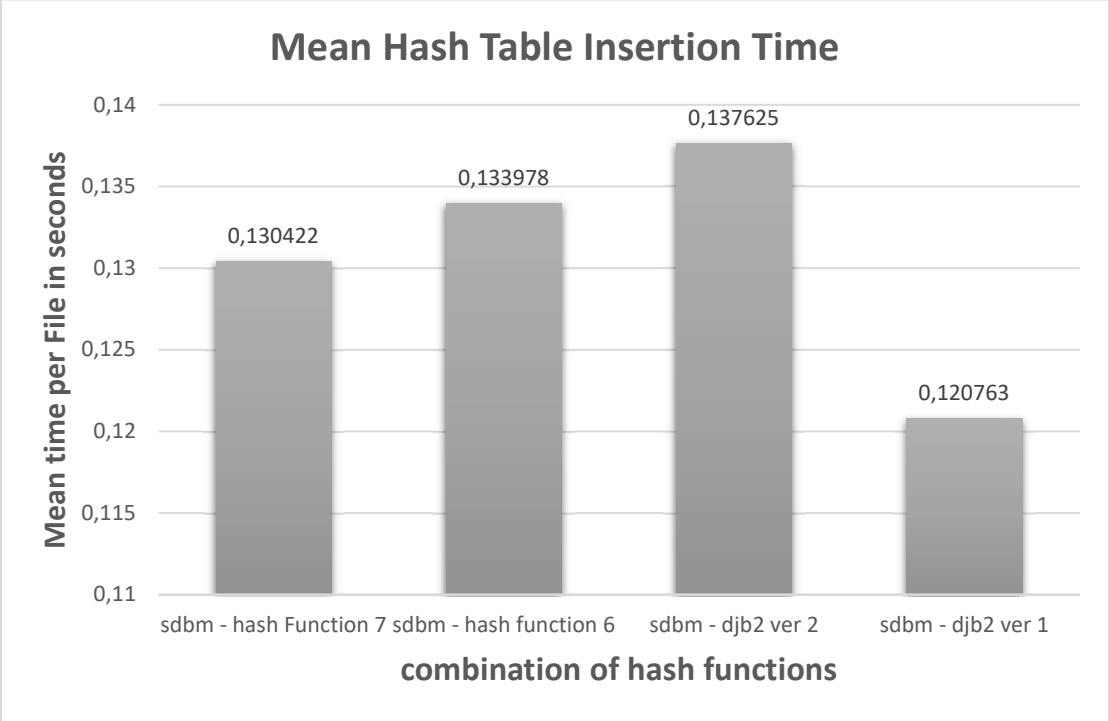
```
unsigned long long int h=0;
int i=0;
while( key[i]!='\0' )
    h=31*h + ((int)key[i++]);

return h % arraySize;
```

hash Function 7

```
static unsigned long long int p[13]={1,31,961,29791,923521 ,28629151,887503681,27512614111,852891037441,26439622160671
                                     819628286980801,25408476896404831,787662783788549761};
unsigned long long int sum=0;
unsigned long long int prime=1;
int i=0;
while( key[i] != '\0' ) // in case the word has length more than 13 characters we use as prime the 13th power of 31
{
    if( i>=13 )
        prime=p[12];
    else
        prime=p[i];
    sum += ( (int)key[i] ) * prime;
    i++;
}

return sum % arraySize;
```



Κατά μέσο όρο (το πλήθος των αρχείων που δοκιμάστηκαν ήταν 190 και χρησιμοποιήθηκαν τα ίδια αρχεία που χρησιμοποιήθηκαν και στο Μοντέλο Γραμμικής Παλινδρόμησης) για την εισαγωγή όλων των λέξεων στον πίνακα κατακερματισμού με τον συνδυασμό συναρτήσεων κατακερματισμού **sdbm – djb2 version 1** χρειάστηκαν 0.12 δευτερόλεπτα (η μικρότερη τιμή) συνεπώς επιλέχθηκε αυτός ο συνδυασμός.

3.4 Μοντέλο Γραμμικής Παλινδρόμησης

3.4.1 Υπολογισμός Βέλτιστης Ευθείας

Για την δημιουργία ενός μοντέλου Γραμμικής Παλινδρόμησης χρησιμοποιήθηκε η μέθοδος των ελάχιστων τετραγώνων (Least Squares) και 190 txt αρχεία από το Gutenberg Project (https://www.gutenberg.org/). Για τον υπολογισμό της ευθείας (της μορφής **h=a*x+b**, όπου **x** η ανεξάρτητη μεταβλητή είναι ο αριθμός των επεξεργασμένων λέξεων ενός txt αρχείου και **h** η εξαρτημένη μεταβλητή είναι οι μοναδικές λέξεις που περιέχει το αρχείο) που ταιριάζει καλύτερα στα δεδομένα πρέπει να βρούμε εκείνα τα **a** και **b** για τα οποία η συνάρτηση σφάλματος ελαχιστοποιείται, όπου η συνάρτηση σφάλματος είναι η:

$$E(a,b) = \sum_{i=1}^{m} [(ax_i + b) - h_i]^2$$

Η συνάρτηση είναι μία παραβολή που είναι παντού θετική και αν παρουσιάζει ολικό ελάχιστο τότε αυτό θα είναι σε κάποιο από τα σημεία που ικανοποιούν την παρακάτω εξίσωση:

$$\nabla E(a,b) = (\partial E/\partial a, \partial E/\partial b) = (0,0)$$

Έτσι μετά την παραγώγιση καταλήγουμε στο παρακάτω σύστημα γραμμικών εξισώσεων:

$$0 = \partial E/\partial a = 2\sum_{i=1}^{m} [(ax_i + b) - h_i]x_i$$
$$0 = \partial E/\partial b = 2\sum_{i=1}^{m} [(ax_i + b) - h_i]$$

Ή με την μορφή πινάκων:

$$\begin{bmatrix} \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} x_i h_i \\ \sum_{i=1}^{m} h_i \end{bmatrix}$$

Το σύστημα αυτό μπορεί να λυθεί από αριθμητικούς αλγόριθμους αν όμως λυθεί σε αυτή την μορφή μπορεί να οδηγήσει σε ένα **ill-conditioned** γραμμικό σύστημα(εν ολίγης ένα **ill-conditioned** γραμμικό σύστημα είναι ένα σύστημα στο οποίο μπορεί μικρά σφάλματα στα δεδομένα να επιφέρουν μεγάλα σφάλματα στην λύση). Για τον λόγο αυτό πρέπει πρώτα να

υπολογίσουμε τους μέσους όρους των δεδομένων και να τους αφαιρέσουμε από τα δεδομένα. Τελικά το σύστημα που καθορίζει τα **a** και **b** είναι το παρακάτω:

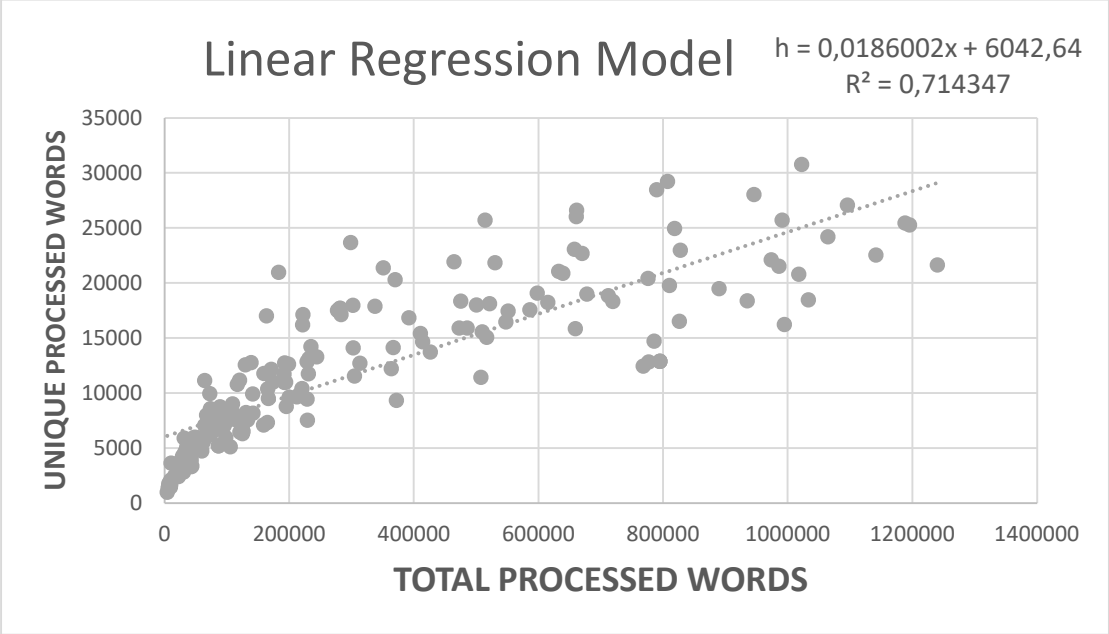
$$\begin{bmatrix} \sum_{i=1}^{m}(x_i-\bar{x})^2 & 0 \\ 0 & m \end{bmatrix} \begin{bmatrix} \bar{a} \\ \bar{b} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m}(x_i-\bar{x})(h_i-\bar{h}) \\ 0 \end{bmatrix}$$

Το οποίο έχει λύση:

$$\bar{a} = \frac{\sum_{i=1}^{m}(x_i-\bar{x})(h_i-\bar{h})}{\sum_{i=1}^{m}(x_i-\bar{x})^2}, \quad \bar{b} = 0$$

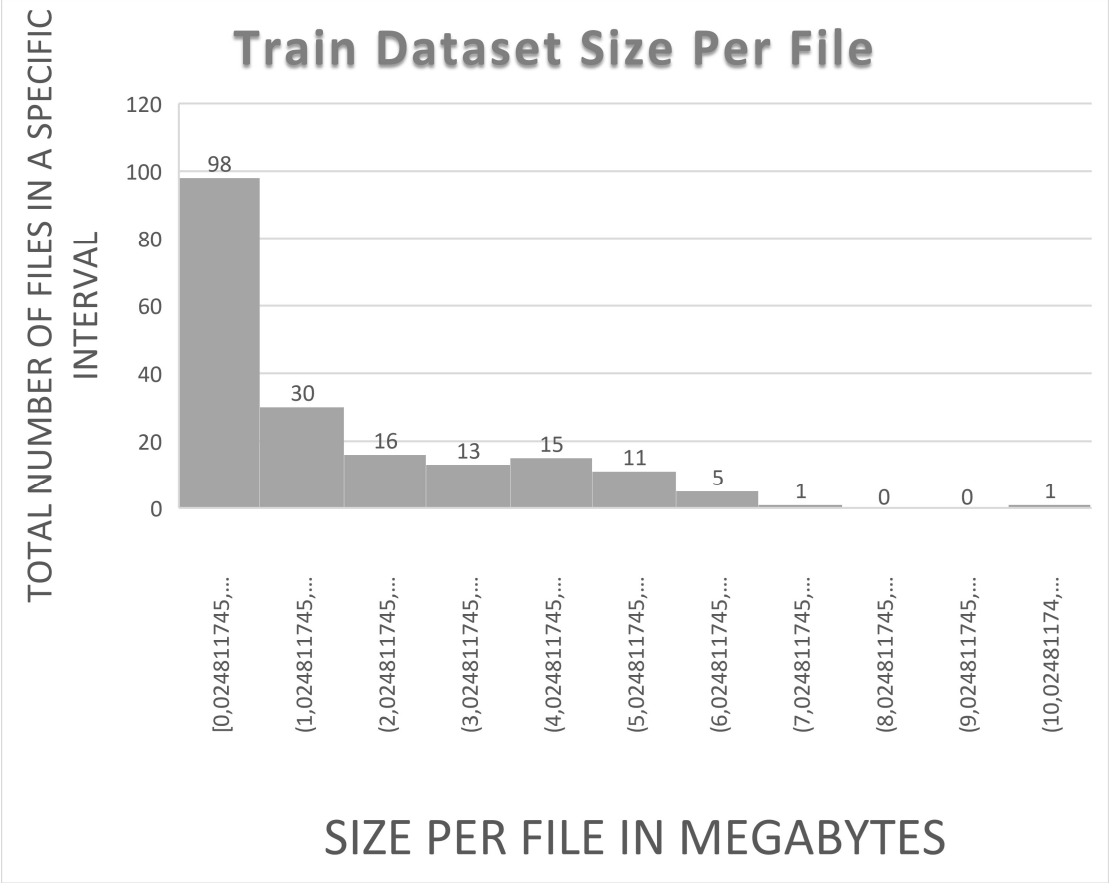
Και τελικά **a =** $\bar{a}$ και **b=** $\bar{h} - \bar{a} * \bar{x}$.

Η εκτέλεση του αλγορίθμου με τα 190 txt αρχεία οδήγησε στην παρακάτω ευθεία με R^2 Error= 0,714347.



3.4.2 Μέγεθος αρχείων του train dataset

Παρακάτω παρουσιάζεται το ιστόγραμμα του μεγέθους των αρχείων που χρησιμοποιήθηκαν στο train dataset:



3.4.3 Train Dataset και RMSE

Το μοντέλο Γραμμικής Παλινδρόμησης δοκιμάστηκε σε test dataset με 20 txt αρχεία (90-10 σχέση train και test dataset) από το οποίο εξάχθηκε RMSE (Root Mean Square Error) = 8862,3122 (μονάδες μέτρησης του RMSE είναι οι μοναδικές λέξεις που υπάρχουν σε ένα αρχείο κειμένου μετά την επεξεργασία του). Το μέγεθος RMSE υποδεικνύει την προσαρμογή του μοντέλου στα δεδομένα, δηλαδή ποσό κοντά βρίσκονται οι αρχικές παρατηρήσεις σε σχέση με τις τιμές που προβλέπει το μοντέλο. Στην εφαρμογή μας, παρατηρούμε ότι το μοντέλο προβλέπει τις μοναδικές λέξεις ενός αρχείου κειμένου με ένα σφάλμα της τάξης $\pm$8862,3122 λέξεων που ίσως θεωρηθεί λογικό καθώς το μοντέλο εκπαιδεύτηκε σε μία αντίστροφη εκθετική κατανομή (βλέπε 3.4.2) και όχι σε μία ομοιόμορφη κατανομή μεγέθους αρχείων κειμένου λόγω των περιορισμένων δεδομένων που υπήρχαν διαθέσιμα για την εκπαίδευση του.

4. Ενδεικτική Χρήση Της Εφαρμογής

Σε μία ενδεχόμενη συνάρτηση main (διαφορετική από αυτή που εμπεριέχεται στα παραδοτέα) θα χρειαστούν να δημιουργηθούν τα παρακάτω αντικείμενα (έτσι ώστε να γίνει και η χρονομέτρηση για την κάθε δομή και όχι απλά να χρησιμοποιηθούν οι δομές δεδομένων):

- Πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης **fileHandler** και να κληθεί η μέθοδος readText()
- Πρέπει να δημιουργηθούν αντικείμενα των κλάσεων **bst**,**avl** και **hashTable**(στον constructor της κλάσης προτείνεται να χρησιμοποιηθεί σαν όρισμα η κλήση της μεθόδου getTotalTextWords μέσω του αντικειμένου της κλάσης **fileHandler**, αλλά ακόμα κι αν για παράδειγμα η δομή χειρίζεται συμβολοσειρές από κάποια άλλη πηγή ή χρησιμοποιηθεί ο default constructor, η μέθοδος resize θα αυξήσει το μέγεθος του πίνακα κατάλληλα, αναμένεται βέβαια χειρότερη απόδοση του hashTable αν δεν χρησιμοποιηθεί ο constructor σε συνδυασμό με το μοντέλο γραμμικής παλινδρόμησης).
- Πρέπει να δημιουργηθεί ένας πίνακας τύπου **string** και να δεσμευθεί ο απαραίτητος χώρος μνήμης για τον πίνακα.
- Τέλος, πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης **dsHandler**(η κατασκευή του αντικειμένου πραγματοποιεί και την εισαγωγή των επεξεργασμένων λέξεων στις δομές).

Στη συνέχεια και αφού έχουν γίνει τα παραπάνω μέσω της κλάσης **dsHandler** πραγματοποιούνται, χρονομετρούνται και αποθηκεύονται στο αρχείο με το όνομα που έχει οριστεί στο αντικείμενο της κλάσης **fileHandler** οι λειτουργίες της αναζήτησης και της διαγραφής, ενώ οι μέθοδοι για τις διασχίσεις των δένδρων καλούνται απευθείας από τα αντικείμενα των κλάσεων **bst** και **avl.** Αν η καλούσα συνάρτηση main κι αυτός που την προγραμματίζει θέλει για παράδειγμα να διαγράψει ή να αναζητήσει λιγότερες, περισσότερες ή διαφορετικές λέξεις από αυτές που αρχικά επιλέχθηκαν για το σύνολο Q θα πρέπει να αλλάξει την τιμή του ιδιωτικού μέλους **Qsize** του αντικειμένου της κλάσης **dsHandler** μέσω του **setter** και να τροποποιήσει επίσης κατάλληλα και το περιεχόμενο του πίνακα που δημιουργήθηκε αρχικά (ή εναλλακτικά μπορεί να χρησιμοποιηθεί ακόμα και διαφορετικός πίνακας) και στην συνέχεια να εκτελέσει τις λειτουργίες που επιθυμεί. Τέλος, αναφέρεται ότι η χρονομέτρηση πραγματοποιείται μόνο μέσω των μεθόδων της κλάσης **dsHandler**, οπότε αν οποιαδήποτε λειτουργία κληθεί άμεσα μέσω των αντικειμένων απλώς θα εκτελεστεί χωρίς να χρονομετρηθεί.

5. Πιθανές Επεκτάσεις

Μία πιθανή επέκταση της εφαρμογής θα μπορούσε να ήταν με κάθε εκτέλεση η προσθήκη στο αρχείο **statistics.txt** των δεδομένων του αρχείου με το οποίο έγινε η εκτέλεση, δηλαδή πόσες συνολικές επεξεργασμένες λέξεις περιείχε και πόσες από αυτές είναι μοναδικές, με σκοπό να βελτιώνεται όλο και περισσότερο το μοντέλο Γραμμικής Παλινδρόμησης. Βέβαια θα χρειαστεί προσοχή σε περιπτώσεις όπου επαναλαμβάνεται η εκτέλεση του ίδιου αρχείου από έναν κακόβουλο χρήστη ή πραγματοποιούνται εκτελέσεις με ένα συγκεκριμένο μέγεθος αρχείου.