Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Πληροφορικής

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

2ο Εξάμηνο

Εργασία Εαρινού Εξαμήνου 2020-2021

Αλεξία Νταντουρή (ΑΕΜ: 3871)

Καλλιόπη Πλιόγκα (ΑΕΜ: 3961)

ΠΕΡΙΕΧΟΜΕΝΑ

1. Θέμα Εργασίας

Μας ζητήθηκε να υλοποιήσουμε τις παρακάτω 5 δομές δεδομένων σε γλώσσα C++, για αποθήκευση των διαφορετικών λέξεων ενός κειμένου, καθώς και του πλήθους εμφανίσεων της κάθε λέξης μέσα στο κείμενο.

- αταξινόμητος πίνακας (Unsorted Array)
- ταξινομημένος πίνακας (Sorted Array)
- απλό δυαδικό δένδρο αναζήτησης (BST)
- δυαδικό δένδρο αναζήτησης τύπου AVL (AVL)
- πίνακας κατακερματισμού με ανοιχτή διεύθυνση (HashTable)

Για κάθε δομή θα πρέπει να υποστηρίζεται εισαγωγή, διαγραφή, αναζήτηση, για τις δενδρικές δομές επιπλέον inorder, preorder, postorder και για τον πίνακα κατακερματισμού απαιτείται μόνο εισαγωγή και αναζήτηση.

Για τον έλεγχο του κώδικα θα πρέπει να χρησιμοποιηθούν τα αρχεία κειμένου:

- small-file.txt (~250.000 λέξεις, 18.874 διαφορετικές λέξεις)
- gutenberg.txt (~390.000.000 λέξεις, 1.724.393 διαφορετικές λέξεις)

2. Αρχεία

2.1 Source Code

Το πρόγραμμα αναπτύσσεται σε 13 αρχεία (6 header files και 7 .cpp files)

.h files:

- UnsortedArray.h
- SortedArray.h
- Node.h
- BST.h
- AVL.h
- HashTable.h

.cpp files:

- UnsortedArray.cpp
- SortedArray.cpp
- Node.cpp
- BST.cpp
- AVL.cpp
- HashTable.cpp
- main.cpp

Για κάθε δομή το header file περιλαμβάνει τις δηλώσεις των μεθόδων της, οι υλοποιήσεις των οποίων βρίσκονται μέσα στα αντίστοιχα implementation files (.cpp). Επιπλέον, δημιουργήσαμε την κλάση Node, η οποία αναπαριστά κόμβο του δυαδικού δένδρου αναζήτησης (Node.h και Node.cpp).

2.2 Output Files

Το πρόγραμμα δημιουργεί 6 αρχεία κειμένου

1. UnsortedArray.txt
2. SortedArray.txt
3. BST.txt
4. AVL.txt
5. HashTable.txt
6. output.txt

Για κάθε μία δομή δεδομένων δημιουργείται ένα αρχείο κειμένου (αρχεία 1-5 στην παραπάνω λίστα), που περιλαμβάνει τις 1000 τυχαίες λέξεις που χρησιμοποιούνται για τον έλεγχο λειτουργίας της αναζήτησης σε κάθε δομή και το πλήθος εμφανίσεων τους στο κείμενο. Φυσικά, μετά την εκτέλεση του προγράμματος, το περιεχόμενο των 5 αρχείων είναι ίδιο μεταξύ τους.

Στο αρχείο output.txt αποθηκεύεται ο συνολικός χρόνος αναζήτησης για κάθε δομή.

Η main εκτελείται για ένα από τα δύο αρχεία εισόδου (small-file.txt ή gutenberg.txt), το οποίο ορίζεται μέσα στη main. Δεν ζητείται ως είσοδος το όνομα του αρχείου από το χρήστη, σύμφωνα με την εκφώνηση της εργασίας.

3. Υλοποίηση των Δομών (Κλάσεις)

3.1 Αταξινόμητος Πίνακας (Unsorted Array)

Στο private μέρος της κλάσης UnsortedArray δηλώνεται ένα *struct* **wordcnt**, ένας δυναμικός πίνακας *wordcnt* **unsorted_array**, ένας ακέραιος (*int*) **numofwords** κι ένας ακέραιος (*int*) **maxsize**.

Το struct **wordcnt** αποτελείται από ένα *string* **word** κι ένα ακέραιο (*int*) **cnt**. Στο string word αποθηκεύεται μία λέξη και στον ακέραιο cnt αποθηκεύεται το πλήθος εμφανίσεων της.

Ο δυναμικός πίνακας **unsorted_array** αποτελείται από structs wordcnt, δηλαδή κάθε κελί του είναι ένα struct wordcnt, στο οποίο αποθηκεύεται μια λέξη και το πλήθος εμφανίσεων της.

Ο ακέραιος **numofwords** αποθηκεύει τον αριθμό των διαφορετικών λέξεων που είναι αποθηκευμένες στον αταξινόμητο πίνακα, ενώ ο ακέραιος **maxsize** αποθηκεύει το μέγεθος

του πίνακα, δηλαδή τον μέγιστο αριθμό διαφορετικών λέξεων που μπορούν να αποθηκευτούν στον πίνακα εκείνη τη στιγμή.

Στο public μέρος της κλάσης UnsortedArray δηλώνεται ένας κατασκευαστής **UnsortedArray()** και οι μέθοδοι:

- **bool resize_array():** για τον διπλασιασμό του μεγέθους του δυαδικού πίνακα
- **bool linearSearch(const string&, int&, int&):** για την γραμμική αναζήτηση μιας λέξης στον πίνακα
- **bool insert_word(const string&):** για την εισαγωγή μιας λέξης στον πίνακα
- **bool delete_word(const string&):** για τη διαγραφή μιας λέξης από τον πίνακα
- **int getNumofwords():** που επιστρέφει τον αριθμό των διαφορετικών λέξεων του πίνακα
- **void show():** εμφανίζει όλες τις λέξεις του πίνακα και το πλήθος εμφανίσεων τους

ΑΝΑΛΥΤΙΚΟΤΕΡΑ

Αναζήτηση λέξης στον πίνακα (Search):

- Μέσω της **linearSearch(const string &keyword, int &counter, int &index_pos)** γίνεται σειριακή αναζήτηση στις κατειλημμένες θέσεις (numofwords) του πίνακα μέχρις ότου να βρεθεί η λέξη ή να ελεγχθούν όλες οι πιθανές θέσεις που θα μπορούσε να βρίσκεται. Σε περίπτωση που βρεθεί η συγκεκριμένη λέξη τότε επιστρέφεται η τιμή true, το πλήθος εμφανίσεων της λέξης και ο δείκτης θέσης της λέξης μέσω των παραμέτρων counter και index_pos. Σε αντίθετη περίπτωση επιστρέφεται η τιμή false.

Εισαγωγή λέξης στον πίνακα (Insertion):

- Μέσω της **insert_word(const string &word)** γίνεται η εισαγωγή της λέξης word στον πίνακα. Αρχικά, γίνεται σειριακή αναζήτηση της λέξης word στον αταξινόμητο πίνακα. Αν βρεθεί, τότε αυξάνεται το πλήθος εμφανίσεων της λέξης (cnt). Αλλιώς, ελέγχεται αν ο πίνακας είναι γεμάτος. Στην περίπτωση που είναι γεμάτος, τότε καλείται η συνάρτηση **resize_array()** για τον διπλασιασμό του μεγέθους του πίνακα. Έπειτα η λέξη προστίθεται στην πρώτη ελεύθερη θέση του πίνακα.
- Μέσω της **resize_array()** ελέγχουμε αν υπάρχει διαθέσιμος χώρος για τον διπλασιασμό του μεγέθους του αταξινόμητου πίνακα. Σε περίπτωση που υπάρχει, διπλασιάζεται το μέγεθος του πίνακα προκειμένου να εισαχθούν με επιτυχία κι άλλες λέξεις.

Διαγραφή λέξης από τον πίνακα (Deletion):

- Μέσω της **delete_word(const string &word)** διαγράφεται μια λέξη από τον πίνακα. Αν ο πίνακας δεν είναι άδειος τότε γίνεται σειριακή αναζήτηση της λέξης στον πίνακα. Αν η λέξη βρεθεί τότε όλα τα στοιχεία που βρίσκονται μετά από αυτήν στον πίνακα μετακινούνται μια θέση αριστερά και το πλήθος των διαφορετικών λέξεων που βρίσκονται στον πίνακα μειώνεται κατά μία μονάδα (numofwords--).

3.2 Ταξινομημένος Πίνακας (Sorted Array)

Η κλάση SortedArray περιλαμβάνει τις αντίστοιχες μεταβλητές και μεθόδους με την κλάση UnsortedArray.

ΑΝΑΛΥΤΙΚΟΤΕΡΑ

Αναζήτηση λέξης στον πίνακα (Search):

- Μέσω της **binarySearch(const string&, int&, int&)** γίνεται δυαδική αναζήτηση της λέξης σε όλο τον πίνακα. Η διαδικασία αυτή ξεκινά συγκρίνοντας τη λέξη που δέχεται η συναρτηση ως παράμετρο με αυτή που βρίσκεται στο μέσο του πίνακα. Αν είναι ίσες τότε επιστρέφεται η τιμή true, το πλήθος εμφανίσεων της λέξης και ο δείκτης θέσης της λέξης μέσω των παραμέτρων c και m. Αν η λέξη κλειδί είναι μεγαλύτερη τότε η αναζήτηση γίνεται στο δεξί-μισό τμήμα του πίνακα και στη συνέχεια επαναλαμβάνεται η διαδικασία από την αρχη. Σε περίπτωση που η λέξη είναι μικρότερη από αυτή του πίνακα τότε η αναζήτηση συνεχίζεται στο αριστερο-μισο τμήμα του πίνακα και στη συνέχεια επαναλαμβάνεται η διαδικασία από την αρχή. Τέλος αν το εύρος της αναζήτησης μειωθεί στο 0 τότε η λέξη δεν βρέθηκε και επιστρέφεται η τιμή false.

Εισαγωγή λέξης στον πίνακα (Insertion):

- Μέσω της **insert_word(const string &word)** γίνεται η εισαγωγή της λέξης word στον πίνακα. Αρχικά, γίνεται δυαδική αναζήτηση της λέξης word στον αταξινόμητο πίνακα. Ο δείκτης index υποδεικνύει τη θέση στην οποία πρέπει να μπει η λέξη σε περίπτωση που δεν υπάρχει ήδη στον πίνακα. Αν βρεθεί, τότε αυξάνεται το πλήθος εμφανίσεων της λέξης (cnt). Αλλιώς, ελέγχεται αν ο πίνακας είναι γεμάτος. Στην περίπτωση που είναι γεμάτος, τότε καλείται η συνάρτηση **resize_array()** για τον διπλασιασμό του μεγέθους του πίνακα. Έπειτα όλα τα στοιχεία από το index και μετά, μετακινούνται μια θέση δεξιά και η νέα λέξη εισάγεται στη θέση που υποδεικνύει ο δείκτης index.
- Μέσω της **resize_array()** ελέγχουμε αν υπάρχει διαθέσιμος χώρος για την εισαγωγή κι άλλης λέξης στον ταξινομημένο πίνακα. Σε περίπτωση που δεν υπάρχει διπλασιάζεται το μέγεθος του πίνακα προκειμένου να εισαχθούν με επιτυχία κι άλλες λέξεις.

Διαγραφή λέξης από τον πίνακα (Deletion):

- Μέσω της **delete_word(const string &word)** διαγράφεται μια λέξη από τον πίνακα. Αν ο πίνακας δεν είναι άδειος τότε γίνεται δυαδική αναζήτηση της λέξης στον πίνακα. Αν η λέξη βρεθεί τότε όλα τα στοιχεία που βρίσκονται μετά από αυτήν στον πίνακα μετακινούνται μια θέση αριστερά και το πλήθος των διαφορετικών λέξεων που βρίσκονται στον πίνακα μειώνεται κατά μία μονάδα (numofwords--).

3.4 Δυαδικό Δέντρο Αναζήτησης (BST)

Στο private μέρος της κλάσης BST δηλώνεται:

- ένας pointer **Node *root**
- μια μέθοδος **bool insert_word(Node *,const string &):** για την εισαγωγή μιας λέξης στο δυαδικό δένδρο
- μία μέθοδος **Node *search_word(const string&):** για την αναζήτηση μιας λέξης στο δυαδικό δένδρο
- μία μέθοδος **bool delete_word(Node *):** για την διαγραφή μιας λέξης από το δυαδικό δένδρο
- μία μέθοδος **Node *min(Node *):** για την εύρεση του ελαχίστου στοιχείου του δένδρου
- μία μέθοδος **Node *max(Node *):** για την εύρεση του ελαχίστου στοιχείου του δένδρου
- μία μέθοδος **void inOrder(Node *):** για την εμφάνιση των στοιχείων του δένδρου σε ενδοδιατεταγμένη σειρά
- μία μέθοδος **void preOrder(Node *):** για την εμφάνιση των στοιχείων του δένδρου σε προδιατεταγμένη σειρά
- μία μέθοδος **void postOrder(Node *):** για την εμφάνιση των στοιχείων του δένδρου σε μεταδιατεταγμένη σειρά
- μία μέθοδος **void destroyTree(Node *):** για την καταστροφή του δένδρου
- ένας ακέραιος **int numofwords:** στον οποίο αποθηκεύονται πόσες διαφορετικές λέξεις υπάρχουν στο δένδρο

Αντίστοιχα στο public μέρος της κλάσης δηλώνεται:

- ένας κενός κατασκευαστής **BST()**
- μία μέθοδος **void destroyTree():** για τη καταστροφή του δέντρου
- μια μέθοδος **bool insert_word(const string &):** καλεί την insert_word(Node *,const string &) για την εισαγωγή της λέξης στο δυαδικό δέντρο
- μία μέθοδος **bool search_word(string,int &):** καλεί την *search_word(string) για την αναζήτηση μιας λέξης στο δυαδικό δέντρο
- μία μέθοδος **bool delete_word(string):** καλεί την delete_word(Node *) για την διαγραφή μιας λέξεις από το δυαδικό δέντρο
- μία μέθοδος **void printInOrder():** καλεί την inOrder() για την εμφάνιση των στοιχείων του δένδρου σε ενδοδιατεταγμένη σειρά
- μία μέθοδος **void printPreOrder():** καλεί την preOrder() για την εμφάνιση των στοιχείων του δένδρου σε προδιατεταγμένη σειρά
- μία μέθοδος **void printPostOrder():** καλεί την postOrder() για την εμφάνιση των στοιχείων του δένδρου σε μεταδιατεταγμένη σειρά
- μία ακέραια μέθοδο **int getNumofwords():** επιστρέφει τον αριθμό των διαφορετικών λέξεων που υπάρχουν στο δυαδικό δένδρο

Οι συναρτήσεις insert, search, delete, inorder, preorder, postorder είναι διπλές έτσι ώστε η διαχείριση της ρίζας να γίνεται μέσα στην κλάση κι όχι στη main.

ΑΝΑΛΥΤΙΚΟΤΕΡΑ

- Μέσω της **void printInOrder()** καλείται η **void preOrder(Node *),** με αυτό τον τρόπο για κάθε κόμβο επισκεπτόμαστε πρώτα τον ίδιο τον κόμβο, έπειτα τους κόμβους του αριστερού υπόδενδρου και στη συνέχεια τους κόμβους του δεξί υπόδενδρου.
- Μέσω της **void printPostOrder**() καλείται η **void preOrder(Node *)** με αυτό τον τρόπο για κάθε κόμβο επισκεπτόμαστε πρώτα τους κόμβους του αριστερού υπόδενδρου, έπειτα τους κόμβους του δεξιού υπόδενδρου και στην συνέχεια τον ίδιο τον κόμβο.
- Μέσω της **void printInOrder**() καλείται η **void inOrder(Node ***) με αυτό τον τρόπο για κάθε κόμβο επισκεπτόμαστε πρώτα τους κόμβους του αριστερού υπόδενδρου, έπειτα τον ίδιο τον κόμβο και στην συνέχεια τους κόμβους του δεξιού του υπόδενδρου.

Αναζήτηση λέξης στο δυαδικό δένδρο (Search):

- Μέσω της **bool search_word(string,int &)** καλείται η **Node *search_word(const string&),** μέσω αυτής βλέπουμε αν το κλειδί που ψάχνουμε (στη συγκεκριμένη υλοποίηση είναι μία λέξη) είναι ίσο με αυτό της ρίζας. Σε περίπτωση που δεν ισχύει κάτι τέτοιο και το κλειδί είναι μικρότερο από αυτό της ρίζας κάνουμε αναδρομική κλήση στο αριστερό υπόδενδρο. Σε αντίθετη περίπτωση κάνουμε αναδρομική κλήση στο δεξί υπόδενδρο. Εν τέλει η μέθοδος bool search_word(string keyword, int &counter) επιστρέφει false αν δεν βρέθηκε το στοιχείο, αλλιώς επιστρέφει true και το πλήθος εμφανίσεων της λέξης αποθηκεύεται στη μεταβλητή counter

Εισαγωγή λέξης στο δυαδικό δένδρο(Insert):

- Μέσω της **bool insert_word(const string &)** καλείται η **bool insert_word(Node *,const string &).** Μέσω αυτής ελέγχουμε αν η λέξη είναι ίση με τη λέξη της ρίζας. Αν αυτό ισχύει τότε αυξάνουμε το πλήθος εμφανίσεων της λέξης κι επιστρέφουμε true. Αλλιώς αν η λέξη είναι μεγαλύτερη από τη λέξη της ρίζας τότε υπάρχουν 2 περιπτώσεις. Αν η ρίζα δεν έχει δεξί παιδί τότε δημιουργείται νέος κόμβος ως δεξί παιδί και προστίθεται σε αυτόν η λέξη κι επιστρέφεται true. Αν, όμως, υπάρχει δεξί παιδί τότε καλείται αναδρομικά η ίδια συνάρτηση **bool insert_word(Node *,const string &)** με παράμετρο Node το δεξί παιδί έτσι ώστε να γίνει εισαγωγή στο δεξί υποδένδρο. Σε περίπτωση που η λέξη είναι μικρότερη από τη λέξη της ρίζας εκτελείται η ίδια διαδικασία για το αριστερό υποδένδρο.

Διαγραφή λέξης από το δυαδικό δένδρο(Delete):

- Μέσω της **bool delete_word(string)** καλείται η **bool delete_word(Node *)** και γίνεται η διαγραφή μιας λέξης από το δένδρο. Αν ο κόμβος στον οποίο βρίσκεται η λέξη που θέλουμε να διαγράψουμε είναι φύλλο τότε απλά διαγράφουμε τον κόμβο. Αν ο κόμβος έχει ένα παιδί τότε ο κόμβος παιδί αντικαθιστά τον κόμβο που διαγράφουμε. Αν ο κόμβος έχει 2 παιδιά τότε ο κόμβος αντικαθιστάται από τον ελάχιστο κόμβο του δεξιού υποδένδρου.

3.5 Δυαδικό Δέντρο Αναζήτησης Τύπου AVL

Στο private μέρος της κλάσης AVL είναι δηλώνεται:

- Μία ακέραια μεταβλητή **int numofwords:** αποθηκεύει το πλήθος των διαφορετικών λέξεων που υπάρχουν στο δένδρο
- μία μέθοδος **NodeAVL *min(NodeAVL *):** για την εύρεση του ελαχίστου στοιχείου του δένδρου

Αντίστοιχα στο public μέρος της κλάσης αυτής δηλώνεται:

- Ένας κενός κατασκευαστής **AVL()**
- Μία μέθοδος **void destroyTree(NodeAVL *r):** για την καταστροφή του δένδρου
- Μία μέθοδος **int max(int,int):** επιστρέφει το μέγιστο μεταξύ των 2 ακεραίων που εισάγονται ως παράμετροι
- Μία μέθοδος **int height(NodeAVL *):** που επιστρέφει το ύψος του υπόδενδρου
- Μία μέθοδος **int getBalance(NodeAVL *):** που επιστρέφει τη διαφορά του ύψους ανάμεσα στο αριστερό και το δεξί υπόδενδρο
- Μία μέθοδος **NodeAVL* newNode(string, int):** για την δημιουργία ενός κόμβου
- Μία μέθοδος **NodeAVL* insert_word(NodeAVL*,const string):** για την εισαγωγή μιας λέξης στο δυαδικό δένδρο
- Μία μέθοδος **NodeAVL *rightRotate(NodeAVL *):** απλή δεξιά περιστροφή
- Μία μέθοδος **NodeAVL *leftRotate(NodeAVL *):** απλή αριστερή περιστροφή
- Μία μέθοδος **NodeAVL* deleteNodeAVL(NodeAVL *,const string):** για τη διαγραφή μιας λέξης από το δυαδικό δένδρο
- Μια μέθοδος **bool search_word(NodeAVL *,const string,string &,int &):** για την αναζήτηση μιας λέξης στο δυαδικό δένδρο
- μία μέθοδος **void inOrder(NodeAVL *):** για την εμφάνιση των στοιχείων του δένδρου σε ενδοδιατεταγμένη σειρά
- μία μέθοδος **void preOrder(NodeAVL *):** για την εμφάνιση των στοιχείων του δένδρου σε προδιατεταγμένη σειρά
- μία μέθοδος **void postOrder(NodeAVL *):** για την εμφάνιση των στοιχείων του δένδρου σε μεταδιατεταγμένη σειρά
- μία ακέραια μέθοδο **int getNumofwords():** που επιστρέφει το πλήθος των διαφορετικών λέξεων που υπάρχουν στο δυαδικό δένδρο

Μέσα στο header της κλάσης αυτής δημιουργούμε και μία κλάση NodeAVL, η οποία έχει μόνο public μεταβλητές. Με αυτόν τον τρόπο μπορούμε να διαχειριστούμε τη ρίζα του δένδρου τόσο στην main όσο και στην κλάση AVL.

<u>ΑΝΑΛΥΤΙΚΟΤΕΡΑ</u>

- Μέσω της **void preOrder(NodeAVL *)** για κάθε κόμβο επισκεπτόμαστε πρώτα τον ίδιο τον κόμβο, έπειτα τους κόμβους του αριστερού υπόδενδρου και στη συνέχεια τους κόμβους του δεξί υπόδενδρου.
- Μέσω της **void preOrder(NodeAVL *)** για κάθε κόμβο επισκεπτόμαστε πρώτα τους κόμβους του αριστερού υπόδενδρου, έπειτα τους κόμβους του δεξιού υπόδενδρου και στην συνέχεια τον ίδιο τον κόμβο.

- Μέσω της **void inOrder(NodeAVL *)** για κάθε κόμβο επισκεπτόμαστε πρώτα τους κόμβους του αριστερού υπόδενδρου, έπειτα τον ίδιο τον κόμβο και στην συνέχεια τους κόμβους του δεξιού του υπόδενδρου.

Αναζήτηση λέξης στο δυαδικό δένδρο AVL (Search):

- Μέσω της **bool search_word(NodeAVL *,const string,string &,int &)** βλέπουμε αν το κλειδί που ψαχνουμε (στη συγκεκριμένη υλοποίηση είναι μία λέξη) είναι ίσο με αυτό της ρίζας. Σε περίπτωση που δεν ισχύει κάτι τέτοιο και το κλειδί είναι μικρότερο από αυτό της ρίζας κάνουμε αναδρομική κλήση στο αριστερό υπόδενδρο. Σε αντίθετη περίπτωση κάνουμε αναδρομική κλήση στο δεξί υπόδενδρο.

Εισαγωγή λέξης στο δυαδικό δένδρο AVL (Insert):

- Μέσω της **NodeAVL* insert_word(NodeAVL*, string)** γίνεται η εισαγωγή των λέξεων στο δέντρο. Συγκεκριμένα, αν ο κόμβος NodeAVL είναι nullptr τότε αυξάνεται το πλήθος των λέξεων που υπάρχουν στο δένδρο και προστίθεται σε αυτόν η νέα λέξη. Αν ο κόμβος δεν είναι nullptr και αν η λέξη για εισαγωγή είναι μικρότερη από την λέξη του κόμβου τότε γίνεται αναδρομική κλήση για την εισαγωγή της λέξης στο αριστερό υποδένδρο. Αλλιώς αν η λέξη για εισαγωγή είναι μεγαλύτερη από την λέξη του κόμβου τότε εκτελείται η αντίστοιχη διαδικασία για το αριστερό υποδένδρο. Αν η λέξη υπάρχει ήδη στο δένδρο τότε αυξάνεται το πλήθος εμφανίσεών της. Στη συνέχεια γίνεται ενημέρωση του ύψους του δένδρου και στην περίπτωση που το δένδρο δεν είναι ισοζυγισμένο εκτελούνται οι κατάλληλες περιστροφές ώστε να είναι ισοζυγισμένο.
- Μέσω της **NodeAVL *leftRotate(NodeAVL *)** και της **NodeAVL *rightRotate(NodeAVL *)** γίνονται οι κατάλληλες περιστροφές προκειμένου το δένδρο να είναι ισορροπημένο. Συγκεκριμένα, αν με την εισαγωγή μιας λέξης το δένδρο δεν ισορροπεί και το ύψος του δεξιού υπόδενδρου είναι μεγαλύτερο από αυτό του αριστερού τότε γίνεται αριστερή περιστροφή. Το αντίθετο συμβαίνει όταν το αριστερό υπόδενδρο είναι μεγαλύτερο. Δηλαδή σε αυτή την περίπτωση καλείται η μέθοδος **NodeAVL *rightRotate(NodeAVL *).**

Διαγραφή λέξης από το δυαδικό δένδρο AVL(Delete)

- Μέσω της **NodeAVL* deleteNodeAVL(NodeAVL *,const string)** γίνεται η διαγραφή των λέξεων από το δέντρο. Αν ο κόμβος στον οποίο βρίσκεται η λέξη που θέλουμε να διαγράψουμε είναι φύλλο τότε απλά διαγράφουμε τον κόμβο. Αν ο κόμβος έχει ένα παιδί τότε ο κόμβος παιδί αντικαθιστά τον κόμβο που διαγράφουμε. Αν ο κόμβος έχει 2 παιδιά τότε ο κόμβος αντικαθιστάται από τον ελάχιστο κόμβο του δεξιού υποδένδρου. Στη συνέχεια γίνεται ενημέρωση του ύψους του δένδρου και στην περίπτωση που το δένδρο δεν είναι ισοζυγισμένο εκτελούνται οι κατάλληλες περιστροφές ώστε να είναι ισοζυγισμένο.

3.6 Πίνακας Κατακερματισμού (HashTable)

Στο private μέρος της κλάσης HashTable δηλώνεται:

- Μία ακέραια μεταβλητή (int) **size:** για το μέγεθος του πίνακα
- Μία ακέραια μεταβλητή (int) **elements:** το πλήθος των στοιχείων του πίνακα
- Ένας πίνακας WordCNT ****A**
- Ένας δείκτης (void) ***deleted**
- Μία θετική ακέραια συνάρτηση **unsigned int hash(const string &):** συνάρτηση κατακερματισμού
- Μία ακόμη θετική ακέραια συνάρτηση **unsigned int hash2(unsigned int):** δεύτερη συνάρτηση κατακερματισμού
- Μία Wordfr συνάρτηση ***&search(const string &)**

Αντίστοιχα στο public μέρος της κλάσης δηλώνεται:

- Ένας κενός κατασκευαστής **HashTable()**
- Μία μέθοδος **void clearHashTable():** για την καταστροφή του πίνακα
- Μία λογική συνάρτηση **bool insert(const string &):** για την εισαγωγή μιας λέξης στον πίνακα
- Μία λογική συναρτηση **bool search(const string &, int &):** για την αναζήτηση μιας λέξης στον πίνακα
- Μία ακέραια συνάρτηση **int getElements():** επιστρέφει το πλήθος των στοιχείων του πίνακα
- Μία συνάρτηση **void show():** για την εμφάνιση των στοιχείων του πίνακα

Στο header της κλάσης αυτής δημιουργούμε και μία άλλη κλάση **WordCNT**, στο public μέρος της οποίας δηλώνεται ένας κατασκευαστής και οι μεταβλητές cnt(int), word(string). Η δημιουργία αυτής της κλάσης γίνεται προκειμένου να μπορούμε να αποθηκεύουμε σε κάθε κελί του πίνακα μία λέξη και το πλήθος εμφανίσεων της λέξης.

ΑΝΑΛΥΤΙΚΟΤΕΡΑ

- Η μέθοδος **unsigned int hash(const string &)** είναι η συνάρτηση κατακερματισμού. Δέχεται μια λέξη κι επιστρέφει τη θέση του πίνακα στην οποία πρέπει να μπει. Λεπτομέρειες: http://www.cse.yorku.ca/~oz/hash.html
- Η μέθοδος **unsigned int hash2(unsigned int)** είναι η δεύτερη συνάρτηση κατακερματισμού. Δέχεται έναν μη αρνητικό ακέραιο κι επιστρέφει τη θέση του πίνακα στην οποία πρέπει να μπει η λέξη σε περίπτωση που έχει υπάρξει collision.

Αναζήτηση λέξης στον πίνακα κατακερματισμού (Search):

- Μέσω της **bool search(const string &, int &)** καλείται η ***&search(const string &),** η οποία επιστρέφει το δείκτη θέσης της λέξης που αναζητείται (επιστρέφει nullptr αν η λέξη δεν βρεθεί). Συγκεκριμένα, μέσω της ***&search(const string &)**, καλείται η συνάρτηση **hash** κι αν η λέξη που βρίσκεται στη θέση που επέστρεψε η hash δεν είναι αυτή που ψάχνουμε τότε καλείται η hash2 μέχρις ότου βρεθεί η λέξη ή φτάσουμε σε κενό δείκτη. Αν η λέξη βρεθεί, η συνάρτηση **bool search(const string &, int &)**

επιστρέφει true και μέσω την μεταβλητής c (αναφορικά) το πλήθος εμφανίσεων της λέξης. Αν η λέξη δεν βρεθεί επιστρέφεται η τιμή false.

Εισαγωγή λέξης στον πίνακα κατακερματισμού (Insert):

- Μέσω της **bool insert(const string &)** γίνεται η εισαγωγή μιας λέξης στον πίνακα κατακερματισμού. Αρχικά, γίνεται αναζήτηση της λέξης μέσω της συνάρτησης **bool search(const string &, int &)**. Αν η λέξη υπάρχει ήδη στον πίνακα, τότε αυξάνεται το πλήθος εμφανίσεών της κι επιστρέφεται true. Διαφορετικά, αν ο πίνακας έχει γεμίσει πάνω από το μισό, τότε διπλασιάζεται το μέγεθός του και τα στοιχεία του μεταφέρονται σε νέες θέσεις με βάση τη συνάρτηση κατακερματισμού. Τέλος, η νέα λέξη προστίθεται στον πίνακα. Συγκεκριμένα, καλείται η **unsigned int hash(const string &),** που επιστρέφει την θέση στην οποία πρέπει να εισαχθεί το στοιχείο. Σε περίπτωση που η θέση δεν είναι κενή και υπάρξει collision τότε καλείται η **unsigned int hash2(unsigned int),** η οποία στέλνει νέα θέση μέχρις ότου να βρεθεί κενή θέση. Όταν τελικά η νέα θέση εισαχθεί στον πίνακα, το πλήθος των στοιχείων του πίνακα αυξάνεται κατά μία μονάδα κι επιστρέφεται true.

4. Main

Το πρόγραμμα αρχικά κατασκευάζει τις δομές δεδομένων δημιουργώντας αντικείμενα από την κάθε κλάση και θέτει την ρίζα του AVL σε nullptr. Έπειτα, ορίζει στην μεταβλητή filename το όνομα του αρχείου κειμένου που θα διαβαστεί και διαγράφει τα περιεχόμενα του αρχείου output.txt ώστε να γίνει η αποθήκευση των νέων μετρήσεων.

Στη συνέχεια, δημιουργεί έναν πίνακα random καλώντας την συνάρτηση **create_random(const string &fn),** στον οποίο αποθηκεύονται 1000 τυχαίες λέξεις από το αρχείο.

Για κάθε δομή καλεί την αντίστοιχη συνάρτηση insert (πχ. insertHashTable), η οποία ανοίγει το αρχείο με όνομα filename και εισάγει κάθε λέξη του αρχείου σε κάθε δομή καλώντας την αντίστοιχη μέθοδο insert. Πριν την εισαγωγή κάθε λέξης γίνεται επεξεργασία της λέξης μέσω της συνάρτησης **edit_word(string word_input),** η οποία αφαιρεί από τη λέξη τα σημεία στίξης και μετατρέπει τα κεφαλαία γράμματα σε πεζά.

Μετά την εισαγωγή των λέξεων σε κάθε δομή, εμφανίζονται στην οθόνη οι συνολικοί χρόνοι εισαγωγής μέσω της συνάρτησης showTime που δέχεται ως ορίσματα 2 χρονικά σημεία t1 και t2 (t1 χρονικό σημείο πριν την έναρξη της εισαγωγής και t2 χρονικό σημείο αμέσως μετά το τέλος της εισαγωγής).

Έπειτα, για κάθε δομή καλείται μια συνάρτηση που μετρά τον συνολικό χρόνο αναζήτησης των 1000 τυχαίων λέξεων σε κάθε δομή και αποθηκεύει τον χρόνο στο αρχείο κειμένου output.txt.

Αφού ολοκληρωθούν οι παραπάνω εργασίες (εισαγωγή, αναζήτηση), η δομή καταστρέφεται (εκτός από τον ταξινομημένο και τον αταξινόμητο πίνακα) ώστε να υπάρχει χώρος στον buffer για την επόμενη δομή. Επίσης, εκτελώντας διαδοχικά τις εργασίες για κάθε δομή

μπορούμε να έχουμε γρήγορα αποτελέσματα για τις γρήγορες δομές (HashTable, AVL, BST) και δεν χρειάζεται να περιμένουμε ώρες για την ολοκλήρωση της εισαγωγής στον ταξινομημένο και τον αταξινόμητο πίνακα, αφού για το μεγάλο αρχείο για την εισαγωγή των λέξεων σε αυτές τις 2 δομές απαιτούνται πολλές ώρες.

5. Τρέξιμο προγράμματος - Μετρήσεις χρόνων

Οι χρόνοι μετρήθηκαν για το μικρό αρχείο (small-file) και για το μεγάλο αρχείο (gutenberg).

Στο μικρό αρχείο υπάρχουν συνολικά ~250.000 λέξεις (*n1*=18.874 διαφορετικές λέξεις).

Στο μεγάλο αρχείο υπάρχουν συνολικά ~390.000.000 λέξεις (*n2*=1.724.393 διαφορετικές λέξεις).

Άρα το *n2* (δηλαδή το μέγεθος των δομών) για το μεγάλο αρχείο είναι περίπου 100 φορές σε σχέση με το *n1* για το μικρό αρχείο (*n2* =100**n1*).

Συγκεκριμένα μετρήθηκε για κάθε αρχείο, ο χρόνος που χρειάστηκε ώστε να εισαχθούν όλες οι λέξεις του αρχείου σε κάθε δομή και στη συνέχεια ο χρόνος που χρειάστηκε συνολικά για την αναζήτηση 1000 τυχαίων λέξεων σε κάθε δομή.

Οι μετρήσεις έγιναν μέσω του IDE CLion, ενώ ο κώδικας δοκιμάστηκε και στα IDE Code::Blocks και Replit.

Data Structures	**Insertion (small-file)**	**Insertion (gutenberg)**
Hash Table	200 ms	230 sec
AVL	320 ms	531 sec
BST	380 ms	326 sec
Sorted Array	2500 ms	38989 sec (11h)
Unsorted Array	6000 ms	104400 sec (29h)

Πίνακας 1: Χρόνοι εισαγωγής όλων των λέξεων των 2 αρχείων σε κάθε δομή

Data Structures	Search (small-file)	Search (gutenberg)
Hash Table	0.2 ms	0.2 ms
AVL	0.4 ms	0.7 ms
BST	0.7 ms	1.6 ms
Sorted Array	0.5 ms	0.9 ms
Unsorted Array	2.5 ms	3.4 ms

Πίνακας 2: Χρόνοι αναζήτησης 1000 τυχαίων λέξεων σε κάθε δομή

Σχολιασμός Χρόνων

Παρατηρούμε ότι ο χρόνος αναζήτησης των 1000 λέξεων (*Πίνακας 2*) χρησιμοποιώντας τη δομή **Hash Table** είναι ο γρηγορότερος, όπως ήταν αναμενόμενο, αφού γνωρίζουμε ότι η μέση πολυπλοκότητα χρόνου αναζήτησης είναι O(1) και η χειρότερη O(*n*), στην περίπτωση που όλες οι *n* τιμές κλειδιού έχουν τον ίδιο κάδο αναφοράς (collisions). Επίσης, παρατηρούμε πως ο χρόνος αναζήτησης είναι περίπου ο ίδιος τόσο στο μεγάλο όσο και στο μικρό αρχείο, κάτι που επιβεβαιώνει το O(1). Δηλαδή, ο χρόνος αναζήτησης είναι ανεξάρτητος από το μέγεθος του Hash Table (το *n*).

Όσον αφορά τις δομές **AVL** και **Sorted Array,** ο χρόνος αναζήτησης είναι περίπου ίδιος. Αυτό είναι λογικό, καθώς η αναζήτηση σε δένδρο AVL και η δυαδική αναζήτηση είναι O(*logn*).

Φαίνεται, ακόμα, πως ο χρόνος αναζήτησης χρησιμοποιώντας τη δομή **BST** είναι ελάχιστα πιο αργός σε σχέση με τους προηγούμενους. Αυτό, προκύπτει από το γεγονός ότι το BST έχει μέση πολυπλοκότητα χρόνου αναζήτησης O(*logn*), όμως στη χειρότερη περίπτωση O(*n*).

Τέλος, όπως είναι λογικό ο χρόνος αναζήτησης χρησιμοποιώντας τη δομή **Unsorted Array** είναι ο πιο αργός, αφού έχει μέση πολυπλοκότητα χρόνου αναζήτησης O(*n/2*). Θα περιμέναμε ότι ο χρόνος αναζήτησης των 1000 λέξεων στο μεγάλο αρχείο χρησιμοποιώντας τη δομή Unsorted Array να ήταν σημαντικά πιο αργός (μέχρι και 100 φορές περισσότερο αφού *n2*=100**n1*) σε σχέση με το μικρό αρχείο. Ο γρήγορος χρόνος στο μεγάλο αρχείο δικαιολογείται από το γεγονός ότι οι 1000 τυχαίες λέξεις που χρησιμοποιούνται για την αναζήτηση, είναι δείγμα από τις πρώτες λέξεις του αρχείου (Η συνάρτηση create_random που δημιουργήσαμε επιστρέφει δείγμα περίπου λέξη παρα λέξη).