Τεχνική Έκθεση

Χρόνοι αναζήτησης για το αρχείο “gutenberg.txt” (10000 λέξεις) για τη κάθε δομή ξεχωριστά:

Αταξινόμητος Πίνακας: 12,5226 s

Ταξινομημένος Πίνακας: 7,8346 s

Πίνακας Κατακερματισμού: 6,6543 s

Δυαδικό δέντρο αναζήτησης: 6,9821 s

Δέντρο AVL: 6,8527

*ο χρόνος αφορά και την εμφάνιση της κάθε λέξης και του αντίστοιχου πλήθους της

!Στο κώδικα χρησιμοποιείται η βιβλιοθήκη algorithm αλλά αποκλειστικά και μόνο για την μετατροπή των πεζών χαρακτήρων σε κεφαλαία

!Όλες οι μέθοδοι επεξηγούνται πιο αναλυτικά με σχόλια στο κώδικα

main

Στη main:

- διαβάζονται μία μία οι λέξεις του αρχείου
- κατασκευάζονται οι δομές με τις λέξεις του αρχείου χρησιμοποιώντας τη μέθοδο εισαγωγής της κάθε δομής
- κατασκευάζεται ένας πίνακας Q δέκα χιλιάδων λέξεων παίρνοντας ομοιόμορφα λέξεις από το αρχείο
- αναζητούνται οι λέξεις του πίνακα Q σε κάθε μία από τις δομές χρησιμοποιώντας τη μέθοδο αναζήτησης της κάθε δομής και καταγράφεται ο συνολικός χρόνος αναζήτησης.

clean.h

Αυτό το αρχείο περιέχει τη συνάρτηση clean(string &). Η clean δέχεται με αναφορά μια συμβολοσειρά και μετατρέπει τους πεζούς χαρακτήρες σε κεφαλαίους και αφαιρεί τα σημεία στίξης. Επομένως ένας πεζός χαρακτήρας θεωρείται ίδιος με τον αντίστοιχο κεφαλαίο του. Για παράδειγμα οι λέξεις “Hello” και “hello” θεωρούνται

ίδιες. Ως λέξεις υπολογίζονται και οι αριθμοί. Οι αριθμοί έχουν μεγαλύτερη προτεραιότητα από τους χαρακτήρες. Η clean χρησιμοποιείται στη main για κάθε λέξη πριν από την εισαγωγή της στις δομές, αλλά χρησιμοποιείται και σε όλες τις μεθόδους εισαγωγής, αναζήτησης και διαγραφής ώστε οποιαδήποτε λέξη και αν δοθεί μέσω αυτών των μεθόδων να μετατραπεί στην κατάλληλη μορφή.

Αταξινόμητος πίνακας

Ο αταξινόμητος πίνακας υλοποιείται με την κλάση UnSorted και υποστηρίζει τις εξής λειτουργίες:

*Η λέξη και το πλήθος της περιέχονται σε ένα struct

- Εισαγωγή λέξης: Με τη μέθοδο void insert_(string)
  Αρχικά αναζητείται η λέξη με τη βοήθεια της μεθόδου search_ και εάν υπάρχει ήδη τότε αυξάνεται κατά 1 το πλήθος της λέξης αλλιώς εισάγεται η λέξη στην επόμενη κενή θέση του πίνακα (η λέξη που εισάγεται θα έχει αρχικά πλήθος 1). Μετά αυξάνεται ο pointer (δείκτης που δείχνει την επόμενη κενή θέση του πίνακα). Αν ο pointer είναι ίσος με το size_(ο αριθμός θέσεων μνήμης που καταλαμβάνει ο πίνακας) τότε χρησιμοποιείται η μέθοδος moreMemory() για την αύξηση της μνήμης.

- Αναζήτηση λέξης: Με τη μέθοδο bool search_(string)
  Σειριακή αναζήτηση. Για κάθε στοιχείο του πίνακα ελέγχεται εάν η λέξη του στοιχείου είναι η ίδια με αυτή που αναζητείται. Εάν είναι τότε κρατείται η θέση του στοιχείου αυτού σε μια μεταβλητή found και επιστρέφεται true. Εάν δεν βρεθεί η λέξη επιστρέφεται false.

- Διαγραφή λέξης: Με τη μέθοδο void delete_(string)
  Αρχικά αναζητείται η λέξη που είναι προς διαγραφή μέσω της μεθόδου search. Εάν υπάρχει η λέξη και το πλήθος της λέξης είναι μεγαλύτερο από 1 τότε απλά μειώνεται κατά 1 το πλήθος. Αν η λέξη υπάρχει και το πλήθος της είναι 1 τότε στη θέση της μπαίνει η τελευταία λέξη του πίνακα (μαζί με το αντίστοιχο πλήθος της) και διαγράφεται η τελευταία λέξη (μαζί με το αντίστοιχο πλήθος της) . Χρησιμοποιείται η μέθοδος lessMemory() για την μείωση της μνήμης κατά 1. Εάν η λέξη δεν υπάρχει τότε εμφανίζεται το μήνυμα “this word not exists”.

- Αναζήτηση λέξης στον πίνακα και εμφάνισης της λέξης αυτής και του πλήθους της: Με τη μέθοδο void searchHowTimes(string)

- Καλεί τη μέθοδο bool search_(string) και αν αυτή επιστρέψει true τότε εμφανίζει τη λέξη που βρίσκεται στη θέση found με την εξής μορφή μηνύματος: “the word (λέξη) exists (πλήθος λέξης) times.
  Εάν η bool search_(string) επιστρέψει false τότε εμφανίζεται το μήνυμα “this word not exists”.

- Μέγεθος (αριθμός στοιχείων δομής): Με τη μέθοδο int getSize()
- Εμφάνιση ενός στοιχείου: Με τη χρήση του τελεστή [ ]

Ταξινομημένος Πίνακας

Ο ταξινομημένος πίνακας υλοποιείται με την κλάση Sorted η οποία κληρονομεί την UnSorted και υποστηρίζει τις εξής λειτουργίες:

*Η λέξη και το πλήθος της περιέχονται σε ένα struct

- Εισαγωγή λέξης: Με τη μέθοδο void insert_(string)
  Αρχικά αναζητείται η λέξη μέσω της μεθόδου bool search_(string) και εάν υπάρχει ήδη τότε αυξάνεται κατά 1 το πλήθος της λέξης. Αλλιώς όλα τα στοιχεία από το found και μετά (το found είναι η θέση στην οποία πρέπει να εισαχθεί το νέο στοιχείο και βρίσκεται από την search_) μετακινούνται μία θέση προς τα δεξιά. Εισάγεται στη θέση found η λέξη και το πλήθος της και αυξάνεται ο pointer (δείκτης που δείχνει την επόμενη κενή θέση του πίνακα). Αν ο pointer είναι ίσος με το size τότε χρησιμοποιείται η μέθοδος moreMemory() για την αύξηση της μνήμης.

- Αναζήτηση λέξης: Με τη μέθοδο bool search_(string)
  Καλεί τη μέθοδο bool search_(int first, int last, string word). Αυτή η μέθοδος δέχεται ως όρισμα μια λέξη (word), τη θέση του πρώτου (first) και του τελευταίου στοιχείου (last) του πίνακα και πραγματοποιεί δυαδική αναζήτηση αναδρομικά. Μετά από την κλήση της η μεταβλητή found θα έχει την τιμή είτε της θέσης που βρέθηκε η λέξη είτε την τιμή της θέσης που θα έπρεπε να μπει στην περίπτωση που δεν υπάρχει (βοηθάει στην εισαγωγή). Η bool search(string) λοιπόν καλεί την search_(int first, int last, string word) για τη λέξη που της δίνεται σαν όρισμα και για το πρώτο και το τελευταίο στοιχείο του πίνακα. Επιστρέφει true αν η λέξη υπάρχει και false αν δεν υπάρχει.

- Διαγραφή λέξης: Με τη μέθοδο void delete_(string)
  Αρχικά αναζητείται η λέξη με τη βοήθεια της μεθόδου search_(string). Εάν υπάρχει τη λέξη τότε εάν το πλήθος της είναι μεγαλύτερο από 1 τότε απλά μειώνεται κατά 1. Εάν υπάρχει η λέξη και το πλήθος της είναι ίσο με 1 τότε όλα τα στοιχεία από το

found και μετά μετακινούνται μία θέση προς τα αριστερά και χρησιμοποιείται η μέθοδος lessMemory() για την μείωση της μνήμης. Εάν η λέξη δεν υπάρχει τότε εμφανίζεται το μήνυμα “this word not exists”

- Αναζήτηση λέξης στον πίνακα και εμφάνισης της λέξης αυτής και του πλήθους της: Με τη μέθοδο void searchHowTimes(string) που κληρονομείται από την UnSorted

- Μέγεθος (αριθμός στοιχείων δομής): Με τη μέθοδο int getSize() που κληρονομείται από την UnSorted

- Εμφάνιση ενός στοιχείου: Με τη χρήση του τελεστή [ ] που κληρονομείται από την UnSorted

Πίνακας Κατακερματισμού

Ο πίνακας κατακερματισμού υλοποιείται από την κλάση HashTable και υποστηρίζει τις εξής λειτουργίες:

*Η λέξη και το πλήθος της περιέχονται σε ένα struct

- Εισαγωγή λέξης: Με τη μέθοδο void insert_(string)
- Αρχικά αναζητείται η λέξη με τη βοήθεια της μεθόδου search_(string) και εάν υπάρχει ήδη τότε αυξάνεται κατά 1 το πλήθος του στοιχείου στο οποίο βρέθηκε η λέξη αλλιώς η λέξη κατακερματίζεται με τη μέθοδο void hashing(). Το πλήθος των λέξεων αυξάνεται κατά 1 και αν το πλήθος τους είναι ίσο με το μέγεθος του πίνακα τότε χρησιμοποιείται η μέθοδος moreMemory() για την αύξηση της μνήμης.

  void hashing(): Αρχικά υπολογίζεται το hashCode της δοσμένης λέξης. Θα χρειαστεί να ελεγχθούν το πολύ τόσα στοιχεία όσα έχει και ο πίνακας. Ξεκινώντας από το h αναζητείται μια κενή θέση. Αν το h ξεπεράσει το μέγεθος του πίνακα τότε το h γίνεται 0 για να συνεχιστεί η αναζήτηση κενής θέσης από εκεί. Εάν βρεθεί κενή θέση τότε η λέξη εισάγεται σ’ αυτή τη θέση και η μέθοδος τερματίζει.

- Αναζήτηση λέξης: Με τη μέθοδο bool search_(string)
  Αρχικά υπολογίζεται το hashCode της δοσμένης λέξης με τη μέθοδο long hashCode(string) και αποθηκεύεται στη μεταβλητή h. Ξεκινώντας από το στοιχείο h

ελέγχονται όλα τα στοιχεία του πίνακα μέχρι η λέξη να βρεθεί (οπότε επιστρέφεται true) ή να πέσουμε πάνω σε κενή θέση ή να ελεγχθούν όλα τα στοιχεία. Στις δύο τελευταίες περιπτώσεις σημαίνει πως η λέξη δεν υπάρχει και επιστρέφεται false. Σημειώνεται πως κατά τον έλεγχο αν το h ξεπεράσει το μέγεθος του πίνακα τότε το h γίνεται 0 για να συνεχιστεί η αναζήτηση από εκεί. Η θέση στην οποία θα βρεθεί η λέξη αποθηκεύεται σε μια μεταβλητή found.

- Αναζήτηση λέξης στον πίνακα και εμφάνισης της λέξης αυτής και του πλήθους της: Με τη μέθοδο void searchHowTimes(string)
  Καλεί τη μέθοδο bool search_(string) και αν αυτή επιστρέψει true τότε εμφανίζει τη λέξη που βρίσκεται στη θέση found με την εξής μορφή μηνύματος: “the word (λέξη) exists (πλήθος λέξης) times. Εάν η bool search_(string) επιστρέψει false τότε εμφανίζεται το μήνυμα “this word not exists”.

- Εμφάνιση όλων των στοιχείων του πίνακα: Με τη μέθοδο void print()
  Εμφανίζονται όλα τα στοιχεία τα οποία δεν περιέχουν κενή λέξη

- Μέγεθος (αριθμός στοιχείων δομής): Με τη μέθοδο int getSize()

Δυαδικό Δέντρο αναζήτησης

Το δυαδικό δέντρο αναζήτησης υλοποιείται από την κλάση BST

*Περιέχει την ρίζα root, ένα struct WordCount3 για την λέξη και το πλήθος της κα ένα struct Node το οποίο αναπαριστά ένα κόμβο και περιέχει μια μεταβλητή τύπου WordCount3, δύο παιδιά (αριστερό και δεξί) καθένα από τα οποία είναι ένας κόμβος και το ύψος του κόμβου.

Υποστηρίζει τις εξής λειτουργίες:

- Εισαγωγή λέξης: Με τη μέθοδο void insertNode(string)
  Αν η τιμή της ρίζας είναι nullptr (αυτό θα συμβεί όταν ακόμα δεν θα έχουμε δέντρο) τότε με τη βοήθεια της μεθόδου newNode() δημιουργείται ένας νέος

κόμβος με τη λέξη (αυτή η εντολή θα εκτελεστεί για την πρώτη λέξη που θα διαβαστεί). Με ένα loop διασχίζουμε ένα μονοπάτι από τη ρίζα μέχρι ένα φύλλο ή μέχρι να βρεθεί η λέξη με ένα δείκτη curr να δείχνει στο τρέχοντα κόμβο και ένα δείκτη parent το γονέα αυτού. Εάν η λέξη προς εισαγωγή είναι μεγαλύτερη από αυτή του curr τότε το curr θα είναι ίσο με το δεξί παιδί του ενώ αν η λέξη είναι μικρότερη από αυτή του curr τότε το curr θα είναι ίσο με το αριστερό παιδί του, ενώ αν είναι ίσα τότε σημαίνει πως η λέξη υπάρχει ήδη και αυξάνεται το πλήθος της κατά 1. Όταν το curr γίνει nullptr σημαίνει πως το μονοπάτι έφτασε στο τέλος. Αν η λέξη είναι μικρότερη από τη λέξη του parent τότε με τη βοήθεια της μεθόδου newNode η λέξη εισάγεται στο αριστερό παιδί του parent αλλιώς αν η λέξη είναι μεγαλύτερη από τη λέξη του parent τότε με τη βοήθεια της μεθόδου newNode η λέξη εισάγεται στο δεξί παιδί του parent

- Διαγραφή λέξης: Με τη μέθοδο void deleteNode(string)
  Με τη βοήθεια της μεθόδου void searchKey(Node*,string, Node* &) βρίσκεται ο κόμβος(curr) προς διαγραφή και κόμβος-γονέας (parent) του κόμβου προς διαγραφή. Αν το curr είναι null, δηλαδή αν η λέξη δεν υπάρχει τότε εμφανίζεται κατάλληλο μήνυμα και η μέθοδος τερματίζει. Αν το πλήθος της λέξης προς διαγραφή δεν είναι 1 απλά μειώνεται το πλήθος κατά 1 και η μέθοδος τερματίζει. Αν το πλήθος της λέξης του curr είναι 1 διακρίνουμε τρεις περιπτώσεις: α)το curr δεν έχει παιδιά τότε: Αν το curr είναι το αριστερό παιδί του parent τότε διαγράφεται το αριστερό παιδί του parent αλλιώς αν το curr είναι το δεξί παιδί του parent τότε διαγράφεται το δεξί παιδί του parent και απελευθερώνεται η μνήμη που καταλαμβάνει το curr. β) Αν το curr έχει 2 παιδιά τότε το temp χρησιμοποιείται για την αποθήκευση του κόμβου με τον οποίο θα αντικαταστήσουμε το διαγραμμένο κόμβο (βρίσκεται με τη μέθοδο Node *maxValue(Node*)). Αποθηκεύεται σε μια μεταβλητή val η λέξη και το πλήθος που περιέχονται στο temp και το πλήθος του temp γίνεται 1 έτσι ώστε κατά τη διαγραφή του temp να μη μειωθεί το πλήθος αλλά να διαγραφεί εντελώς και διαγράφεται αναδρομικά ο κόμβος με τη λέξη του temp.Τέλος αντικαθιστούμε στο κόμβο που θέλουμε να διαγράψουμε με το με το val. γ)Αν το curr έχει μόνο ένα παιδί τότε αποθηκεύεται στη μεταβλητή child αυτό το παιδί (δεξί ή αριστερό) και ν το curr είναι η ρίζα τότε αντικαθιστούμε την ρίζα αλλιώς αν το curr είναι το αριστερό παιδί του parent τότε αντικαθιστούμε το αριστερό παιδί του parent με το child αλλιώς αν το curr είναι το δεξί παιδί του parent τότε αντικαθιστούμε το δεξί παιδί του parent με το child και απελευθερώνεται η μνήμη που καταλαμβάνει το curr.

- Αναζήτηση λέξης: Με τη μέθοδο bool searchNode(string)
  Θέτει τη public μεταβλητή b με τη τιμή true και καλεί την μέθοδο bool search_(Node* root, string key). Αυτή η μέθοδος είναι αναδρομική. Αρχικά ελέγχει αν το root είναι null και αν δεν είναι τότε αν η λέξη είναι ίδια με αυτή του root τίθεται το b ίσο με true και αποθηκεύεται σε μια μεταβλητή found το root. Αν η λέξη είναι μικρότερη από αυτή του root τότε η λέξη αναζητείται αναδρομικά στο υποδέντρο με ρίζα το αριστερό παιδί του root, αλλιώς η λέξη αναζητείται αναδρομικά στο υποδέντρο με ρίζα το δεξί παιδί του root και επιστρέφεται το b.

- Αναζήτηση λέξης στον πίνακα και εμφάνισης της λέξης αυτής και του πλήθους της: Με τη μέθοδο void searchHowTimes(string)
  Λειτουργεί όπως η bool searchNode(string) με τη διαφορά ότι δεν επιστρέφει τιμή bool αλλά εμφανίζει τη λέξη που βρίσκεται στο κόμβο found με την εξής μορφή μηνύματος: “the word (λέξη) exists (πλήθος λέξης) times, εάν η λέξη υπάρχει ενώ αν δεν υπάρχει εμφανίζει το μήνυμα “this word not exists”.

- Διάσχιση inorder: Με τη μέθοδο void inorder()
  Καλεί την μέθοδο inorder(Node* root) με όρισμα τη ρίζα του δέντρου. Η inorder(Node* root) διατρέχει τους κόμβους με τη σειρά: αριστερά, ρίζα, δεξιά αναδρομικά

- Διάσχιση postorder: Με τη μέθοδο void postorder()
  Καλεί την μέθοδο postorder(Node* root) με όρισμα τη ρίζα του δέντρου. Η postorder(Node* root) διατρέχει τους κόμβους με τη σειρά: Αριστερά, δεξιά, ρίζα αναδρομικά.

- Διάσχιση preorder: Με τη μέθοδο void preorder()
  Καλεί την μέθοδο preorder(Node* root) με όρισμα τη ρίζα του δέντρου. Η preorder(Node* root) διατρέχει τους κόμβους με τη σειρά: ρίζα, αριστερά, δεξιά αναδρομικά.

Ισοζυγισμένο Δυαδικό Δέντρο Αναζήτησης

Το ισοζυγισμένο δυαδικό δέντρο αναζήτησης υλοποιείται από την κλάση AVL και κληρονομεί την BST

Υποστηρίζει τις εξής λειτουργίες:

- Εισαγωγή λέξης: Με τη μέθοδο void insertNode(string)
  Καλεί τη μέθοδο Node* insertNode(Node*, string) δίνοντας ως πρώτο όρισμα τη ρίζα του δέντρου. Αυτή η μέθοδος είναι αναδρομική. Το root είναι κάθε φορά ο τρέχον κόμβος. Αν η τιμή του root είναι nullptr (δηλαδή δεν έχουμε ακόμα δέντρο τότε με τη βοήθεια της μεθόδου newNode δημιουργείται ένας νέος κόμβος και επιστρέφεται αυτός ο κόμβος. Αν η λέξη είναι μικρότερη από τη λέξη του root τότε εισάγεται η λέξη αναδρομικά στο υποδέντρο με ρίζα το δεξί παιδί του root, αλλιώς αν η λέξη είναι μικρότερη από τη λέξη του root τότε εισάγεται η λέξη αναδρομικά στο υποδέντρο με ρίζα το αριστερό παιδί του root, αλλιώς αν η λέξη είναι ίση με τη λέξη του root (που σημαίνει ότι η λέξη υπάρχει ήδη) τότε αυξάνεται το πλήθος της κατά 1. Επιστρέφεται το root και ενημερώνεται το ύψος του. Με τη βοήθεια της getBalance() βρίσκεται η διαφορά του ύψους των δύο υποδέντρων του root και αποθηκεύεται στη μεταβλητή balance. Αν το balance είναι θετικό τότε σημαίνει πως το αριστερό υποδέντρο του root είναι το μεγαλύτερο ενώ αν το balance είναι αρνητικό τότε σημαίνει πως το δεξί υποδέντρο του root είναι το μεγαλύτερο. Αν το balance είναι μεγαλύτερο του 1 και η διαφορά των υψών των υποδέντρων του αριστερού παιδιού του root είναι μεγαλύτερο του 0 (που σημαίνει ότι το αριστερό υποδέντρο του αριστερού παιδιού του root είναι το μεγαλύτερο) τότε γίνεται μια δεξιά περιστροφή και επιστρέφεται η νέα ρίζα του υποδέντρου που προέκυψε μετά την περιστροφή. Αν το balance είναι μεγαλύτερο του 1 και η διαφορά των υψών των υποδέντρων του αριστερού παιδιού του root είναι μικρότερο του 0 (που σημαίνει ότι το δεξί υποδέντρο του αριστερού παιδιού του root είναι το μεγαλύτερο) τότε γίνεται μια αριστερή περιστροφή και μια δεξιά περιστροφή και επιστρέφεται η νέα ρίζα του υποδέντρου που προέκυψε μετά τις περιστροφές. Αν το balance είναι μικρότερο του -1 και η διαφορά των υψών των υποδέντρων του δεξιού παιδιού του root είναι μικρότερο του 0 (που σημαίνει ότι το δεξί υποδέντρο του δεξιού παιδιού του root είναι το μεγαλύτερο) τότε γίνεται μια αριστερή περιστροφή και επιστρέφεται η νέα ρίζα του υποδέντρου που προέκυψε μετά την περιστροφή. Αν το balance είναι μικρότερο του -1 και η διαφορά των υψών των υποδέντρων του δεξιού παιδιού του root είναι μεγαλύτερο του 0 (που σημαίνει ότι το αριστερό υποδέντρο του δεξιού παιδιού του root είναι το μεγαλύτερο) τότε

γίνεται μια δεξιά περιστροφή και μια αριστερή περιστροφή και επιστρέφεται η νέα ρίζα του υποδέντρου που προέκυψε μετά τις περιστροφές.

- Διαγραφή λέξης: Με τη μέθοδο void deleteNode(string)
  Αυτή η μέθοδος είναι αναδρομική. Αν το root (η ρίζα του δέντρου) είναι nullptr τότε εμφανίζεται κατάλληλο μήνυμα και επιστρέφεται το root. Αν η λέξη είναι μικρότερη από αυτή του root τότε καλείται αναδρομικά η deleteNode για το υποδέντρο με ρίζα το αριστερό παιδί του root, αλλιώς αν η λέξη είναι μεγαλύτερη από αυτή του root τότε καλείται αναδρομικά η deleteNode για το υποδέντρο με ρίζα το δεξί παιδί του root, αλλιώς αν η λέξη είναι ίση με αυτή του root τότε αν το πλήθος του root είναι μεγαλύτερο του 1 μειώνεται το πλήθος του κατά 1 και επιστρέφεται το root. Αν το root έχει ένα παιδί ή κανένα τότε αποθηκεύεται στο temp αυτό το παιδί (ή nullptr αν δεν έχει παιδιά). Αν το temp είναι nullptr (θα είναι στη περίπτωση που το root δεν έχει παιδιά) τότε το temp θα πάρει τη τιμή του root και το root θα διαγραφεί. Αν το temp δεν είναι nullptr (η περίπτωση που το root έχει ένα παιδί) το root θα πάρει τη τιμή του temp και θα απελευθερωθεί η μνήμη που καταλαμβάνει. Αλλιώς αν το root έχει δύο παιδιά αντικαθίσταται η λέξη του root με τη λέξη του temp (ο κόμβος με τον οποίο θα αντικαταστήσουμε το διαγραμμένο κόμβο, βρίσκεται με τη μέθοδο Node *maxValue(Node*). Αντικαθίσταται η λέξη του root με τη λέξη του temp και το πλήθος του root με το πλήθος του temp. Το πλήθος του temp γίνεται 1 έτσι ώστε κατά τη διαγραφή του temp να μη μειωθεί το πλήθος αλλά να διαγραφεί εντελώς. Έπειτα καλείτε αναδρομικά η deleteNode για το υποδέντρο με ρίζα το δεξί παιδί του root για την διαγραφή του temp. Τέλος ενημερώνεται το ύψος του root και γίνεται διόρθωση του δέντρου όπως και στην insert.

- Αναζήτηση λέξης: Με τη μέθοδο bool searchNode(string)
  Κληρονομείται από την BST.
- Αναζήτηση λέξης στον πίνακα και εμφάνισης της λέξης αυτής και του πλήθους της: Με τη μέθοδο void searchHowTimes(string)
  Κληρονομείται από την BST.
- Διάσχιση inorder: Με τη μέθοδο void inorder()
  Κληρονομείται από την BST.

- Διάσχιση postorder: Με τη μέθοδο void postorder()
  Κληρονομείται από την BST.

- Διάσχιση preorder: Με τη μέθοδο void preorder()
  Κληρονομείται από την BST.

Τσιριγώτη Ναυσικά ΑΕΜ: 3604

Τσιριγώτης Παντελής ΑΕΜ: 3858