3η Εργασία στις Δομές Δεδομένων

Μέρος Γ – *Report*

Χρήστος Πατρινόπουλος

3200150

Αλβιόνα Μάντσο

3200098

Complexity

Big O

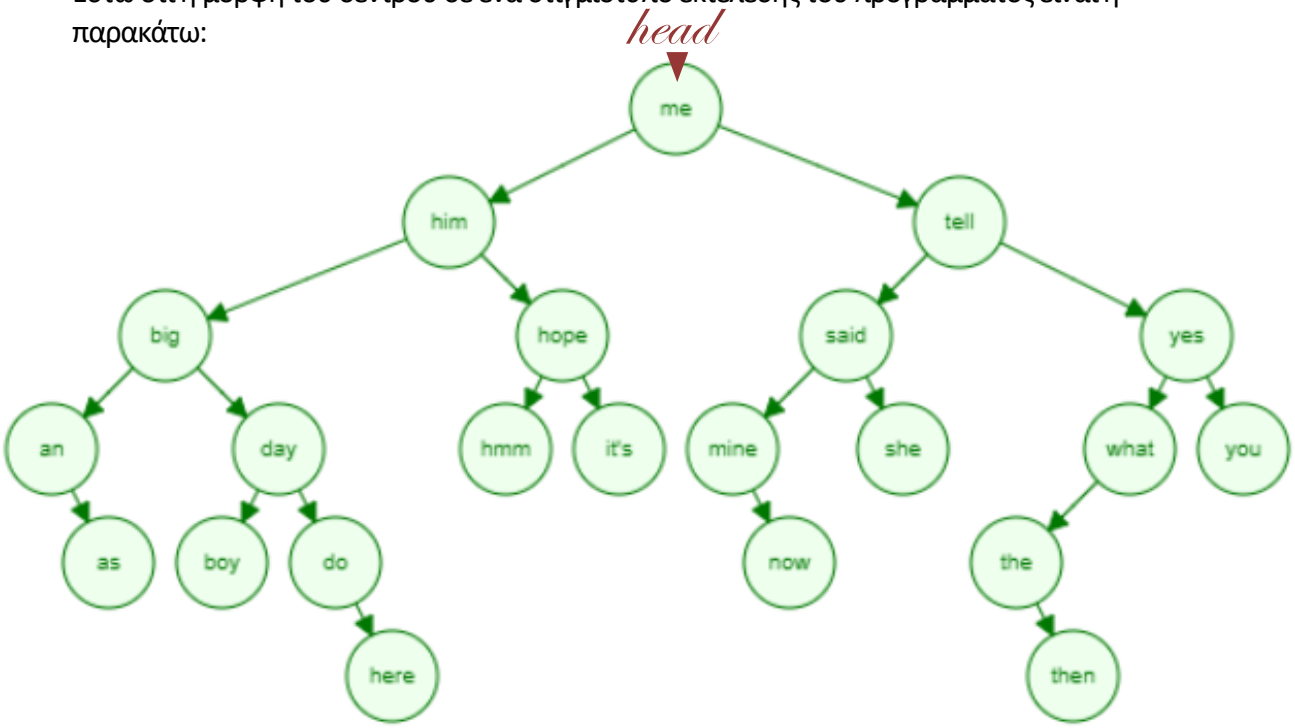
void **insert** (String w) Δέχεται ως όρισμα την λέξη w (String). Διατρέχει επαναληπτικά το δέντρο και εισάγει στην κατάλληλη θέση ως φύλλο έναν νέο κόμβο (TreeNode) με δεδομένο ένα αντικείμενο τύπου **WordFreq**, που έχει ως κλειδί το w. Παράλληλα με κάθε εισαγωγή λέξης ή εύρεσή της στο δέντρο (λειτουργίες που συμβαίνουν στη μέθοδο **load()**), αυξάνεται η μεταβλητή **totalWords**, η οποία συγκρατεί τον αριθμό των συνολικών λέξεων που έχουν διαβαστεί. Μετά από κάθε εισαγωγή καλείται η μέθοδος modifySubTreeSize()* που ενημερώνει κατάλληλα τα πεδία **subtreeSize** των σχετιζόμενων κόμβων. Τέλος, αν η λέξη (w) βρεθεί στο δέντρο (περίπτωση μη εισαγωγής), τότε η συχνότητά της αυξάνεται κατά 1 από τη μέθοδο **increaseFreq()**.	O(N) Η εισαγωγή σε ΔΔΑ τρέχει σε χρόνο O(N) στην χειρότερη περίπτωση. Η insert() εξαρτάται επίσης από τις μεθόδους increaseFreq() που τρέχει σε χρόνο O(1) και την modifySubTreeSize() που στην χειρότερη περίπτωση τρέχει σε χρόνο O(N). Σε κάθε κλήση της insert() εκτελείται μία εκ των από πάνω άρα το κόστος είναι O(N). **Άρα:** O(N)+O(N) = O(N)
WordFreq **search** (String w); Δέχεται ως όρισμα μια λέξη w (String) και διατρέχει επαναληπτικά το δέντρο, επισκεπτόμενη κατάλληλα το αριστερό ή δεξιό υποδέντρο του τρέχοντος κόμβου (χρησιμοποιώντας τις μεθόδους **equals()** και **less()**), για να ελέγξει την ύπαρξή της σε αυτό. Αν η λέξη βρεθεί, σε περίπτωση που η συχνότητά της είναι μεγαλύτερη από τη μέση συχνότητα (χρήση των **getFreq()** της WordFreq και **getMeanFrequency()** της BST), τότε ανεβάζει τον κόμβο WordFreq με κλειδί τη λέξη w στη ρίζα. Αυτό επιτυγχάνεται με διαγραφή της λέξης (μέθοδος remove()) και επανεισαγωγή στη ρίζα (μέθοδος **insertAtRoot()** που υλοποιεί κατάλληλα μεθόδους δεξιάς και αριστερής περιστροφής **rotR()** και **rotL()**).	O(N) Η αναζήτηση σε ΔΔΑ τρέχει σε χρόνο O(N) στην χειρότερη περίπτωση. Σε περίπτωση εύρεσης της λέξης και ικανοποίησης της συνθήκης καλείται η μέθοδος remove() που τρέχει σε χρόνο O(N) και η insertAtRoot() που τρέχει αναδρομικά το πολύ N φορές, άρα O(N). (Οι πράξεις των περιστροφών εντός της insertAtRoot() είναι O(1)). **Άρα:** O(N) + O(N) + O(N) = O(N)
void **remove** (String w); Δέχεται ως όρισμα τη λέξη w (String) και διατρέχει επαναληπτικά το δέντρο. Αν βρεθεί ο κόμβος με κλειδί τη λέξη w, τότε τον αφαιρεί και επαναφέρει την ιδιότητα του ΔΔΑ με κατάλληλες κινήσεις. Μετά από κάθε διαγραφή καλείται η μέθοδος modifySubTreeSize()* που ενημερώνει κατάλληλα τα πεδία **subtreeSize** των σχετιζόμενων	O(N) Η αφαίρεση κόμβου από ΔΔΑ τρέχει σε χρόνο O(N) στη χειρότερη περίπτωση. Η remove() εξαρτάται επίσης από τη μέθοδο modifySubTreeSize() η οποία τρέχει σε χρόνο O(N) στη χειρότερη περίπτωση.

κόμβων. Παράλληλα με κάθε διαγραφή μειώνεται η μεταβλητή **totalWords** κατά τη συχνότητα της λέξης που διαγράφτηκε (με τη μέθοδο **getFreq()**).	**Άρα:** O(N) + O(N) = O(N)
void **load** (String filename) Δέχεται ως όρισμα το path ενός αρχείου filename (String) και κάνει parse το αρχείο. Για κάθε συμβολοσειρά αφού τη χειριστεί κατάλληλα και κρατήσει τη λέξη, ελέγχει αν αυτή εμφανίζεται στη λίστα stopWords (με τη μέθοδο **search()** της LinkedList) και αν όχι, γίνεται εισαγωγή στο δέντρο καλώντας τη μέθοδο **insert()** του BST. Αν υπάρξει πρόβλημα κατά το άνοιγμα-διάβασμα του αρχείου ενημερώνεται κατάλληλα ο χρήστης.	$O(N^2)$ Έστω N: λέξεις του κειμένου, M: αριθμός υπαρχόντων κόμβων, E: αριθμός stopWords Για κάθε λέξη του κειμένου εκτελείται η μέθοδος search() που τρέχει σε χρόνο O(E) και η insert() που τρέχει σε χρόνο O(M) στη χειρότερη περίπτωση. **Άρα:** N * (O(M) + O(E)) = O(N*M) + O(N*E) = O(N*M) Όμως M<=N (αφού οι κόμβοι είναι <= από τις λέξεις του κειμένου) και E αμελητέο, συνεπώς η πολυπλοκότητα είναι $O(N^2)$.
int **getTotalWords**(); Επιστρέφει το συνολικό αριθμό των λέξεων που διαβάστηκαν από το κείμενο κατά την κλήση της μεθόδου **load()**, δηλαδή το πεδίο **totalWords** το οποίο έχει ενημερωθεί κατάλληλα μέσω των μεθόδων **insert()** και **remove()**.	O(1) Δε χρησιμοποιείται διάσχιση, επιστρέφεται κατάλληλα ενημερωμένο πεδίο.
int **getDistinctWords**(); Επιστρέφει τον διακεκριμένο αριθμό λέξεων που διαβάστηκαν από το κείμενο κατά την κλήση της μεθόδου **load()**, δηλαδή το πεδίο **subtreeSize** του κόμβου της ρίζας αυξημένο κατά 1, αφού το πεδίο **subtreeSize** αναφέρεται στον αριθμό των υποκόμβων του κόμβου αναφοράς. Η επιστρεφόμενη τιμή αντιστοιχεί και στον πληθικό αριθμό των κόμβων του δέντρου.	O(1) Δε χρησιμοποιείται διάσχιση, επιστρέφεται κατάλληλα ενημερωμένο πεδίο.
int **getFrequency**(String w); Δέχεται ως όρισμα μία λέξη w (String) και επιστρέφει τη συχνότητα εμφάνισής της μέσα στο κείμενο. Για το σκοπό αυτό χρησιμοποιεί την μέθοδο search(), για να βρεθεί ο κόμβος που έχει ως κλειδί τη λέξη αυτή και καλείται η getFreq() του κόμβου που επιστρέφεται. Αν η λέξη δε βρεθεί επιστρέφεται 0.	O(N) Εξαρτάται από τη search() και τη getFreq(). Η πρώτη τρέχει σε χρόνο O(N) και η δεύτερη σε χρόνο O(1).
WordFreq **getMaximumFrequency**(); Επιστρέφει τον κόμβο του οποίου το κλειδί key έχει τη μεγαλύτερη συχνότητα εμφάνισης στο κείμενο. Χρησιμοποιείται η μέθοδος **preorderMaxFreq()** η οποία δέχεται ως όρισμα το **head** (ρίζα) και το **max** (έστω αρχικά max = κόμβος της ρίζας) και διασχίζει	O(N) Εξαρτάται από τη μέθοδο preorderMaxFreq() η οποία εκτελεί μια διάσχιση του δέντρου σε χρόνο O(N) στη χειρότερη περίπτωση και σε κάθε επίσκεψη εκτελεί σύγκριση σε χρόνο O(1).

προδιατεταγμένα και επαναληπτικά το δέντρο. Σε κάθε κόμβο που επισκέπτεται συγκρίνει με το max και το ενημερώνει κατάλληλα. Για τη διάσχιση γίνεται χρήση της στοίβας **s** (αντικείμενο της κλάσης Stack<T>).	**Άρα:** O(N) * O(1) = O(N)
double **getMeanFrequency**(); Επιστρέφει τη μέση συχνότητα εμφάνισης των λέξεων του κειμένου, καλώντας τις μεθόδους getTotalWords() και getDistinctWords(). (μέση συχνότητα = σύνολο λέξεων / πλήθος διαφορετικών λέξεων)	O(1) Εξαρτάται από τη getTotalWords() και τη getDistinctWords(). Και οι δύο τρέχουν σε χρόνο O(1).
void **addStopWord**(String w); Δέχεται ως όρισμα μία λέξη w (String) και την εισάγει στη λίστα **stopWords**, καλώντας τη μέθοδο **insertAtBack()** της κλάσης LinkedList.	O(1) Εξαρτάται από την insertAtBack(), η οποία τρέχει σε χρόνο O(1). Δε χρησιμοποιείται διάσχιση.
void **removeStopWord**(String w); Δέχεται ως όρισμα μία λέξη w (String) και την αφαιρεί από τη λίστα **stopWords**, καλώντας τη μέθοδο **remove()** της κλάσης LinkedList.	O(M) Εξαρτάται από τη remove() που τρέχει σε χρόνο O(M) και M ο αριθμός των κόμβων της LinkedList. Ουσιαστικά στη χειρότερη περίπτωση διατρέχει όλη τη λίστα.
void **printTreeAlphabetically**(PrintStream stream); Δέχεται ως όρισμα ένα stream (PrintStream) και εκτυπώνει τους κόμβους του δέντρου, καλώντας τη μέθοδο inorderPrintByKey(), η οποία κάνει ενδοδιατεταγμένη αναδρομική διάσχιση αυτού. Σε κάθε κόμβο που επισκέπτεται εκτυπώνει το key().	O(N) Εξαρτάται από την inorderPrintByKey(), η οποία επισκέπτεται κάθε κόμβο του δέντρου μία φορά ακριβώς και άρα τρέχει σε χρόνο O(N), όπου N ο αριθμός των κόμβων.
void **printTreeByFrequency**(PrintStream stream); Δέχεται ως όρισμα ένα stream (PrintStream) και εκτυπώνει τους κόμβους του δέντρου με βάση τη συχνότητα εμφάνισης των κλειδιών (λέξεων) μέσα στο κείμενο σε αύξουσα σειρά, καλώντας τη μέθοδο preorderPrintByFreq(). Η μέθοδος αυτή κάνει προδιατεταγμένη αναδρομική διάσχιση και για κάθε κόμβο που επισκέπτεται τον εισάγει σε έναν πίνακα N θέσεων, όπου N ο αριθμός των συνολικών κόμβων. Στη συνέχεια καλείται ο αλγόριθμος ταξινόμησης (HeapSort) και στο τέλος διατρέχεται ο πίνακας και εκτυπώνονται τα στοιχεία.	O(NlogN) Εξαρτάται από τις getDistinctWords(), getMeanFreq() που τρέχουν σε O(1), από την preorderPrintByFreq() που τρέχει σε χρόνο O(N) (περιλαμβάνει N εισαγωγές στον πίνακα σε O(1) η κάθε μία), την μέθοδο ταξινόμησης σε O(NlogN) και τη διάσχιση του πίνακα για την εκτύπωση σε O(N). **Άρα:** O(1)+O(1)+O(N)+O(NlogN)+O(N)=O(NlogN)

Update subreeSize field

Έστω ότι η μορφή του δέντρου σε ένα στιγμιότυπο εκτέλεσης του προγράμματος είναι η παρακάτω:



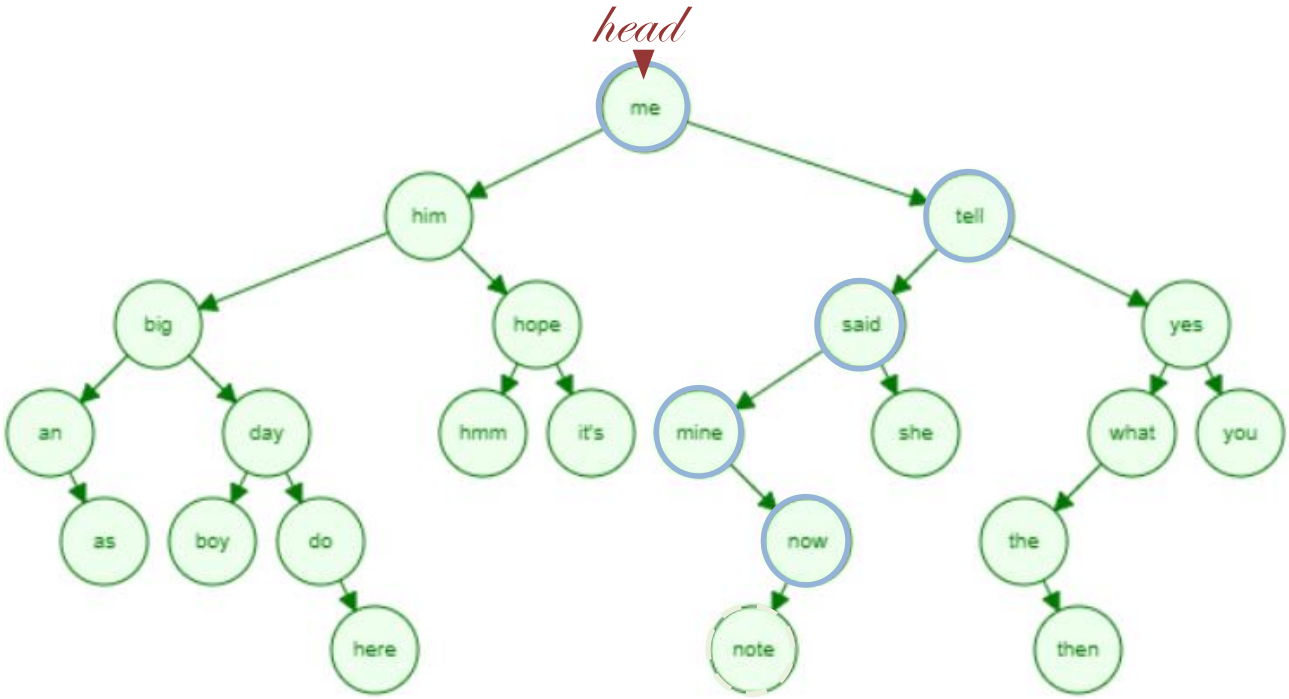
Insert 

Έστω ότι διαβάζεται η λέξη **note** ως επόμενη λέξη του κειμένου. Το μονοπάτι που διασχίζει η λέξη-κόμβος "note" για να εισαχθεί ως φύλλο στο δέντρο:

me – tell – said – mine – now

δηλαδή οι κόμβοι που θα είναι «πρόγονοι» του "note" μόλις αυτό εισαχθεί, θα έχουν έναν επιπλέον υποκόμβο.

Συνεπώς για κάθε κόμβο του μονοπατιού έχουμε αύξηση **subtreeSize += 1**.



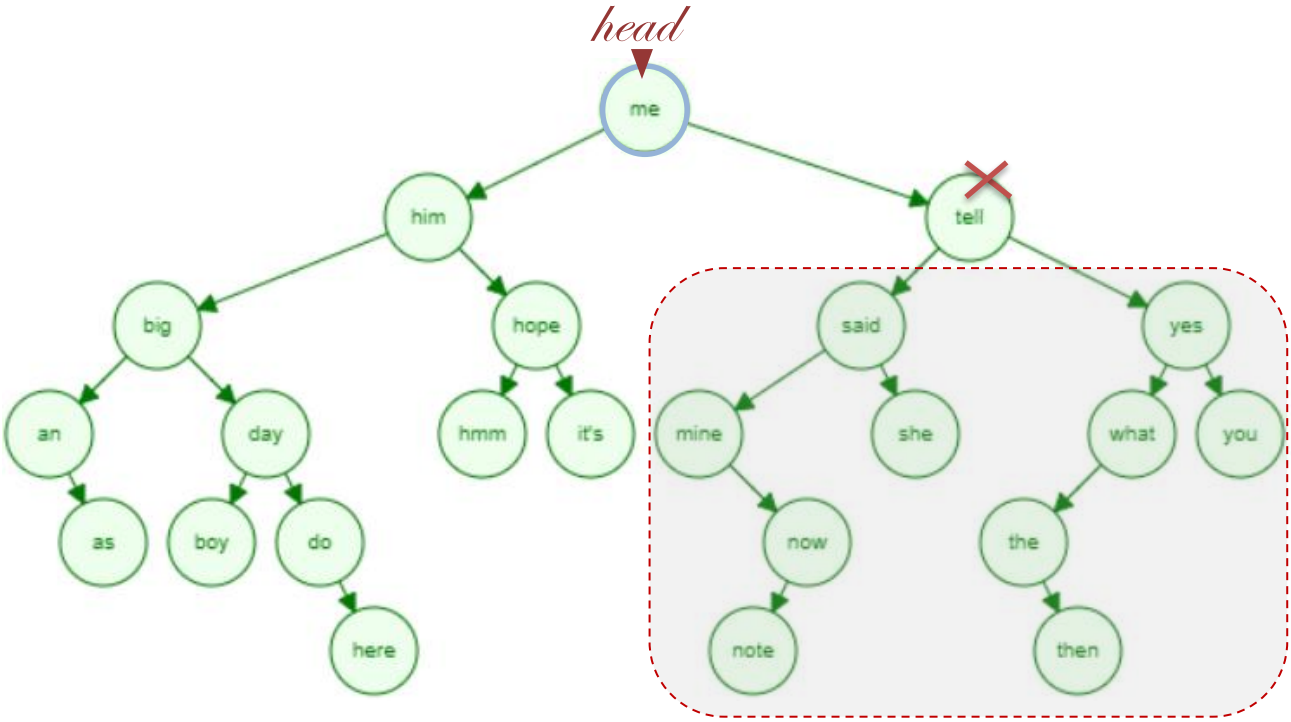
Remove 

Έστω ότι διαγράφεται η λέξη-κόμβος **tell**.

Η διαγραφή χρειάζεται 3 χειρισμούς:

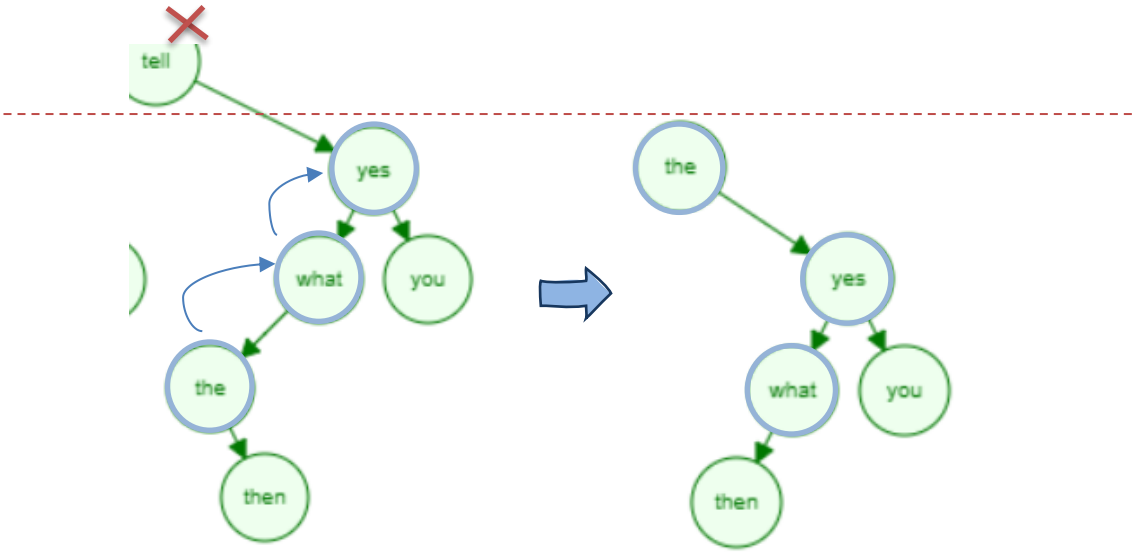
a) Το μονοπάτι των προγόνων του κόμβου που διαγράφεται
b) Το μονοπάτι των προγόνων του μικρότερου στοιχείου (εδώ: **the**) του δεξιού υποδέντρου που με δεξιές περιστροφές ανεβαίνει στη ρίζα
c) Ο ίδιος ο κόμβος (εδώ: **the**) που ανέβηκε στη ρίζα του υποδέντρου

a) Κάθε πρόγονος του κόμβου που διαγράφεται (εδώ: -**me**-) θα έχει έναν λιγότερο υποκόμβο. Συνεπώς για κάθε κόμβο του μονοπατιού προς τη ρίζα έχουμε μείωση **subtreeSize - = 1**.



Προς το παρόν: **Black Box**

b) Το μονοπάτι των προγόνων του μικρότερου στοιχείου του δεξιού υποδέντρου (εδώ: **the**) που ανεβαίνει στη ρίζα, δηλαδή Κάθε κόμβος του μονοπατιού **what – yes** έχει έναν λιγότερο υποκόμβο αφού ο κόμβος the ανέβηκε προς τα πάνω στη ρίζα. Συνεπώς για κάθε κόμβο του μονοπατιού έχουμε μείωση **subtreeSize -= 1**



c) Ο ίδιος ο κόμβος (εδώ: **the**) που ανέβηκε στη ρίζα του υποδέντρου θα έχει πλέον: αριστερό υπόδεντρο το πρώην αριστερό του κόμβου που διαγράφηκε, δεξί υπόδεντρο το πρώην δεξί του κόμβου που διαγράφηκε, εκτός βέβαια από τον εαυτό του.