Δέντρα Αναζήτησης

Δημήτρης Φωτάκης

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

Σχολή Θετικών Επιστημών

Πανεπιστήμιο Αιγαίου

Δεκέμβριος 2006

Πίνακας Περιεχομένων

0. Προκαταρκτικά

Οι δομές δεδομένων που θα μελετήσουμε λειτουργούν σε ένα σύνολο από διατεταγμένα ζεύγη $S = \{ (x, y) : x \in U, y\}$, όπου το U είναι ένα ολικά διατεταγμένο σύνολο (π.χ. το σύνολο των φυσικών αριθμών). Το x αποτελεί το *κλειδί* (key – μοναδική ταυτότητα) κάθε στοιχείου και το y τη *συσχετιζόμενη πληροφορία* (satellite data). Υπάρχει 1-1 αντιστοιχία μεταξύ των στοιχείων και των κλειδιών τους και μια σχέση *ολικής διάταξης* στο σύνολο των κλειδιών. Ανάλογα με την εφαρμογή, η ολική διάταξη των κλειδιών μπορεί να έχει ή να μην έχει φυσική σημασία.

Για παράδειγμα, το κλειδί x μπορεί να αντιπροσωπεύει τον Αριθμό Φορολογικού Μητρώου (ΑΦΜ), που είναι ένας μοναδικός φυσικός αριθμός για κάθε φορολογούμενο, και το y μπορεί να αντιπροσωπεύει το σύνολο της πληροφορίας που υπάρχει στο φάκελο του φορολογούμενου. Το κλειδί – ΑΦΜ χρησιμοποιείται για την οργάνωση του μητρώου των φορολογούμενων (π.χ. αναζήτηση, εισαγωγή, διαγραφή). Οι ΑΦΜ είναι ολικά διατεταγμένοι, αλλά η διάταξή τους δεν έχει φυσική σημασία (π.χ. το γεγονός ότι κάποιος έχει αριθμητικά μικρότερο ΑΦΜ από κάποιον άλλο δεν αντιπροσωπεύει κάτι). Στο μηχανογραφικό σύστημα ενός ληξιαρχείου, το x μπορεί να είναι ο Αριθμός Μητρώου (ΑΜ) των ατόμων που έχουν δηλωθεί εκεί και το y το σύνολο της πληροφορίας που διατηρεί το ληξιαρχείο για κάθε άτομο. Αν υποθέσουμε ότι οι ΑΜ δίνονται αποκλειστικά με βάση τη χρονική στιγμή γέννησης (στην πραγματικότητα δίνονται με βάση τη χρονική στιγμή δήλωσης), τότε η ολική διάταξη των ΑΜ έχει φυσική σημασία (μικρότερος ΑΜ υποδεικνύει μεγαλύτερη ηλικία).

Για την οργάνωση των δεδομένων χρησιμοποιούμε *αποκλειστικά* τα κλειδιά τους. Αυτός είναι ο λόγος που τόσο οι υλοποιήσεις όσο και η ανάλυση αγνοούν ουσιαστικά τη συσχετιζόμενη πληροφορία. Όμως πρέπει πάντα να έχουμε κατά νου ότι σκοπός μας είναι η αποτελεσματική αποθήκευση και διαχείριση της συσχετιζόμενης πληροφορίας.

Σε αυτή την ενότητα, εστιάζουμε σε *δυναμικά* μεταβαλλόμενα σύνολα (dynamic sets) S. Βασικό χαρακτηριστικό των δυναμικά μεταβαλλόμενων συνόλων (ή απλά δυναμικών συνόλων χάριν συντομίας) είναι ότι το σύνολο των στοιχείων δεν είναι γνωστό εκ των προτέρων και απαιτείται ενημέρωση με την προσθήκη νέων και την κατάργηση υπαρχόντων στοιχείων.

0.1. Πράξεις σε Δυναμικά Σύνολα

Οι πράξεις (operations) που εκτελούνται σε ένα δυναμικά μεταβαλλόμενο σύνολο χωρίζονται σε δύο κατηγορίες: πράξεις *ερώτησης* (queries), που επιστρέφουν πληροφορία αλλά δεν μεταβάλλουν το ίδιο το σύνολο, και πράξεις *ενημέρωσης* (modifying operations), που μεταβάλλουν το σύνολο.

Οι βασικές πράξεις ενημέρωσης είναι η *εισαγωγή* (insert) νέου στοιχείου και η *διαγραφή* (delete) υπάρχοντος στοιχείου. Άλλες πράξεις ενημέρωσης είναι η αλλαγή του κλειδιού ενός στοιχείου και η διαγραφή του μέγιστου (delete maximum) ή ελάχιστου (delete minimum) στοιχείου. Αυτές εφαρμόζονται όταν η διάταξη των κλειδιών έχει φυσική σημασία. Ειδικότερα η πρώτη ονομάζεται και *αλλαγή προτεραιότητας* (change priority) του στοιχείου.

Η πλέον βασική πράξη ερώτησης είναι αυτή της *αναζήτησης* (search) ενός στοιχείου με δεδομένο κλειδί. Άλλες πράξεις ερώτησης (ιδιαίτερα σημαντικές όταν η διάταξη των κλειδιών έχει φυσική σημασία) είναι η εύρεση του στοιχείου με το *ελάχιστο* (minimum) ή το *μέγιστο* (maximum) κλειδί, και δεδομένου ενός στοιχείου, η εύρεση του στοιχείου με το αμέσως *προηγούμενο* (predecessor) ή το αμέσως *επόμενο* (successor) κλειδί στην ολική διάταξη των κλειδιών.

0.1.1. Το Πρόβλημα του Λεξικού

Το πρόβλημα του *λεξικού* είναι η αποδοτική υλοποίηση των πράξεων εισαγωγής, διαγραφής, και αναζήτησης σε δυναμικά σύνολα. Μια δομή δεδομένων που υποστηρίζει αυτές τις πράξεις σε δυναμικά σύνολα ονομάζεται δομή δεδομένων για το πρόβλημα του λεξικού (dictionary data structure) ή απλά *λεξικό* (dictionary).

0.1.2. Ουρές Προτεραιότητας

Το πρόβλημα της διαχείρισης ουράς προτεραιότητας είναι να οργανώσουμε τα δεδομένα ενός δυναμικού συνόλου όταν η διάταξη των κλειδιών έχει φυσική σημασία και πρέπει να υλοποιήσουμε αποδοτικά τις πράξεις διαγραφής μέγιστου (ή ελάχιστου), εύρεσης μέγιστου (ή ελάχιστου), εισαγωγής νέου στοιχείου, και αλλαγής προτεραιότητας.

Μια δομή δεδομένων που υποστηρίζει τις παραπάνω πράξεις σε δυναμικά σύνολα ονομάζεται δομή δεδομένων για τη διαχείριση ουράς προτεραιότητας (priority queue data

structure) ή απλά *ουρά προτεραιότητας* (priority queue). Οι ουρές προτεραιότητας διακρίνονται σε αυτές που δίνουν προτεραιότητα στο μέγιστο στοιχείο (max-priority queues) και σε αυτές που δίνουν προτεραιότητα στο ελάχιστο στοιχείου (min-priority queues). Οι πρώτη κατηγορία υποστηρίζει αποδοτικά τις πράξεις της διαγραφής και εύρεσης του μέγιστου, ενώ η δεύτερη κατηγορία τις πράξεις της διαγραφής και εύρεσης του ελάχιστου.

1. Δυαδικά Δέντρα Αναζήτησης

Τα δυαδικά δέντρα αναζήτησης (binary search trees) είναι δομές δεδομένων που υποστηρίζουν (μεταξύ άλλων) τις πράξεις εισαγωγής, διαγραφής, αναζήτησης, και εύρεσης μέγιστου, ελάχιστου, προηγούμενου και επόμενου σε δυναμικά σύνολα. Επομένως, είναι δομές δεδομένων κατάλληλες τόσο για το πρόβλημα του λεξικού όσο και για τη διαχείριση ουρών προτεραιότητας (αν και υπάρχουν πιο αποδοτικές ουρές προτεραιότητας) σε δυναμικά σύνολα.

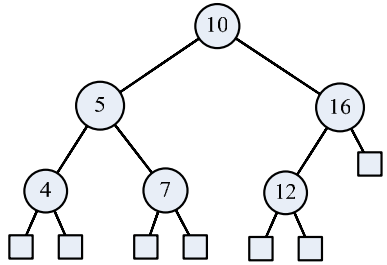
Όπως θα δούμε, η χρονική πολυπλοκότητα χειρότερης περίπτωσης για αυτές τις πράξεις σε ένα δυαδικό δέντρο αναζήτησης είναι γραμμική στο ύψους του δέντρου. Το ύψος ενός δυαδικού δέντρου αναζήτησης με n στοιχεία μπορεί να είναι από λογαριθμικό στον αριθμό των στοιχείων – $O(\log n)$ – μέχρι γραμμικό στον αριθμό των στοιχείων – $O(n)$.

Ένα Δυαδικό Δέντρο Αναζήτησης (ΔΔΑ) είναι ένα δέντρο με ρίζα (rooted tree) του οποίου κάθε εσωτερικός κόμβος έχει ακριβώς δύο παιδιά. Μόνο οι εσωτερικοί κόμβοι του δέντρου περιέχουν στοιχεία. Τα φύλλα του δέντρου, που δεν περιέχουν στοιχεία, αντιστοιχούν σε NULL δείκτες και ονομάζονται NULL-φύλλα[1]. Η ιδιότητα που χαρακτηρίζει τα ΔΔΑ είναι η εξής:

> Έστω x ένας εσωτερικός κόμβος ενός ΔΔΑ. Για κάθε εσωτερικό κόμβο y στο *αριστερό* υποδέντρο του x, το στοιχείο του y είναι *μικρότερο* ή ίσο από το στοιχείο του x, και για κάθε εσωτερικό κόμβο y στο *δεξιό* υποδέντρο του x, το στοιχείο του y είναι *μεγαλύτερο* ή ίσο από το στοιχείο του x.

[1] Διευκρίνιση σχετικά με την ορολογία: Με τον όρο *NULL-φύλλα* αναφερόμαστε στα φύλλα του δέντρου που δεν περιέχουν στοιχεία και αντιστοιχούν σε NULL δείκτες. Τα NULL-φύλλα δεν έχουν παιδιά και συμβολίζονται με τετράγωνα (ενώ όλοι οι υπόλοιποι κόμβοι του δέντρου συμβολίζονται με κύκλους). Με τον όρο *φύλλα* αναφερόμαστε στους κατώτατους εσωτερικούς κόμβους του δέντρου που περιέχουν στοιχεία. Τα φύλλα περιέχουν στοιχεία τα ίδια αλλά κανένα από τα παιδιά τους δεν περιέχει στοιχείο. Με άλλα λόγια, κάθε φύλλο έχει σαν παιδιά δύο NULL-φύλλα.

Η *ιδιότητα των δυαδικών δέντρων αναζήτησης* (binary search tree property) μπορεί να ελεγχθεί στο παράδειγμα του Σχήματος 1.1.



Σχήμα 1.1. Παράδειγμα δυαδικού δέντρου αναζήτησης.

Σαν αποτέλεσμα της ιδιότητας των ΔΔΑ, η ενδο-διατεταγμένη (inorder) διέλευση ενός ΔΔΑ τυπώνει τα στοιχεία του δέντρου σε αύξουσα σειρά. Για παράδειγμα, στο δέντρο του Σχήματος 1.1, η ενδο-διατεταγμένη διέλευση τυπώνει 4, 5, 7, 10, 12, 16. Στη συνέχεια παρουσιάζουμε μία εκδοχή της ενδο-διατεταγμένης διέλευσης σε ψευδοκώδικα και αποδεικνύουμε ότι όντως τυπώνει τα στοιχεία ενός ΔΔΑ σε αύξουσα σειρά σε γραμμικό χρόνο.

```
Inorder-Traversal(x)

    if x ≠ NULL then
        Inorder-Traversal(x.left);
        print(x.key);
        Inorder-Traveral(x.right);
```

Στον ψευδοκώδικα, θα χρησιμοποιούμε τα x, y, z για να δηλώσουμε τους κόμβους ενός ΔΔΑ, και το συμβολισμό *x.left*, *x.right*, *x.par*, και *x.key* για να δηλώσουμε το αριστερό παιδί, το δεξιό παιδί, τον πατέρα, και το κλειδί ενός κόμβου x.

Λήμμα 1.1. Η κλήση `Inorder-Traversal(root)`, όπου *root* η ρίζα ενός ΔΔΑ T με n στοιχεία, τυπώνει τα στοιχεία του T σε αύξουσα σειρά σε χρόνο $\Theta(n)$.

Απόδειξη. Θα δείξουμε το ζητούμενο χρησιμοποιώντας μαθηματική επαγωγή. Είναι φανερό ότι το λήμμα ισχύει όταν το δέντρο T έχει ένα μόνο στοιχείο. Έστω ότι το θεώρημα ισχύει για κάθε ΔΔΑ που έχει το πολύ n στοιχεία.

Θεωρούμε ένα ΔΔΑ T με n+1 στοιχεία. Αυτό αποτελείται από τη ρίζα *root*, το αριστερό υποδέντρο T_1 με $n - k$ στοιχεία, και το δεξιό υποδέντρο T_2 που έχει k στοιχεία

(για κάποιο ακέραιο $k \geq 0$). Η κλήση `Inorder-Traversal(root)` τυπώνει πρώτα τα στοιχεία του αριστερού υποδέντρου σε αύξουσα σειρά (από επαγωγική υπόθεση), μετά το στοιχείο της ρίζας (που είναι μεγαλύτερο ή ίσο όλων των προηγούμενων και μικρότερο ή ίσο όλων των επόμενων στοιχείων από την ιδιότητα των ΔΔΑ), και μετά τα στοιχεία του δεξιού υποδέντρου επίσης σε αύξουσα σειρά. Δηλαδή, τυπώνονται όλα τα στοιχεία του δέντρου σε αύξουσα σειρά.

Όσον αφορά στο χρόνο εκτέλεσης, είναι $\Theta(n–k) + \Theta(1) + \Theta(k) = \Theta(n+1)$, όπου ο πρώτος και ο τελευταίος όρος είναι οι χρόνοι εκτέλεσης των αναδρομικών κλήσεων για το αριστερό και το δεξιό υποδέντρο εξ' αιτίας της επαγωγικής υπόθεσης, και ο δεύτερος όρος είναι ο χρόνος για την κλήση `Inorder-Traversal(root)` και την εκτύπωση του στοιχείου της ρίζας. ■

Για παραδείγματα στα δυαδικά δέντρα αναζήτησης, μπορείτε να χρησιμοποιήσετε το Java applet στη διεύθυνση (απενεργοποιείστε όλες τις επιλογές στα δεξιά):

http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

1.1. Λειτουργίες Ερώτησης

1.1.1. Αναζήτηση

Σε ένα ΔΔΑ, η λειτουργία της αναζήτησης του στοιχείου με δεδομένο κλειδί k υλοποιείται από τον ακόλουθο ψευδοκώδικα:

```
Tree-Search(k)
    x ← root(T);
    while x ≠ NULL and x.key ≠ k do
        if key < x.key then x ← x.left;
        else x ← x.right;
    return(x);
```

Για παράδειγμα, στο ΔΔΑ του Σχήματος 1.1, η αναζήτηση του 7 θα ακολουθήσει το μονοπάτι <10, 5, 7> αφού το 7 είναι αρχικά μικρότερο του 10 και στη συνέχεια μεγαλύτερο του 5.

Η λειτουργία της αναζήτησης υλοποιείται σε χρόνο $O(h)$ στη χειρότερη περίπτωση, όπου h είναι το ύψος του δέντρου. Πράγματι, κάθε αναζήτηση ξεκινάει από τη ρίζα και

ακολουθεί ένα μονοπάτι προς τα φύλλα. Ο χρόνος εκτέλεσης για κάθε κόμβο σε αυτό το μονοπάτι είναι $O(1)$. Η αναζήτηση έχει ολοκληρωθεί όταν φτάσουμε σε NULL-φύλλο (όταν είναι ανεπιτυχής) ή νωρίτερα (όταν είναι επιτυχής).

1.1.2. Ελάχιστο και Μέγιστο

Το ελάχιστο στοιχείο ενός ΔΔΑ βρίσκεται πάντα στον πιο αριστερό (leftmost) κόμβο και το μέγιστο στοιχείο στον πιο δεξιό (rightmost) κόμβο. Πράγματι, η ενδο-διατεταγμένη διέλευση τυπώνει τον πιο αριστερό κόμβο πρώτο και τον πιο δεξιό κόμβο τελευταίο. Ο παρακάτω ψευδοκώδικας επιστρέφει τον κόμβο με το ελάχιστο στοιχείο σε ένα ΔΔΑ. Ο ψευδοκώδικας για το μέγιστο είναι παρόμοιος. Και οι δύο λειτουργίες έχουν χρόνο εκτέλεσης χειρότερης περίπτωσης $O(h)$.

```
Tree-Minimum
    x ← root(T);
    while x.left ≠ NULL do
        x ← x.left;
    return(x);
```

1.1.3. Επόμενος και Προηγούμενος

Σε τα περιπτώσεις ενδιαφερόμαστε για τον επόμενο (successor) ή τον προηγούμενο (predecessor) κόμβο τα δεδομένου κόμβου x στην ενδο-διατεταγμένη διέλευση. Αυτοί οι κόμβοι συμβολίζονται με suc(x) και pred(x) αντίστοιχα. Το στοιχείο του επόμενου κόμβου είναι το μικρότερο στοιχείο του δέντρου που δεν υπολείπεται του $x.key$, και το στοιχείου του προηγούμενου κόμβου είναι το μεγαλύτερο στοιχείου του δέντρου που δεν ξεπερνά το $x.key$. Στη συνέχεια θα επικεντρώσουμε στην εύρεση του επόμενου κόμβου. Η διαδικασία εύρεσης του προηγούμενου κόμβου είναι παρόμοια.

Αν το δεξί παιδί του x είναι εσωτερικός κόμβος (όχι NULL-φύλλο), τότε ο επόμενός του στην ενδο-διατεταγμένη διέλευση είναι ο πιο αριστερός κόμβος (ελάχιστο στοιχείο) του δεξιού υποδέντρου του x (π.χ. στο Σχήμα 1.1, ο επόμενος του 10 είναι το 12 και ο επόμενος του 5 είναι το 7).

Αν το δεξί παιδί του x είναι NULL-φύλλο, θεωρώ το ψηλότερο δυνατό υποδέντρο του οποίου ο x είναι ο πιο δεξιός κόμβος (μέγιστο στοιχείο). Από όλους τους κόμβους αυτού του υποδέντρου, η ενδο-διατεταγμένη διέλευση τυπώνει το στοιχείο του x

τελευταίο και αμέσως μετά τυπώνει τον πατέρα της ρίζας του υποδέντρου. Εξ' ορισμού, η ρίζα αυτού του υποδέντρου (η οποία είναι το αριστερό παιδί του succ(x)) είναι το *πρώτο αριστερό* παιδί που εμφανίζεται στο *μονοπάτι* από το x προς τη *ρίζα* (ολόκληρου) του δέντρου. Με άλλα λόγια, το succ(x) είναι ο *κοντινότερος πρόγονος* του x του οποίου το *αριστερό* παιδί είναι *πρόγονος* του x. Αν δεν υπάρχει τέτοιος κόμβος, ο x είναι ο πιο δεξιός κόμβος του δέντρου και δεν έχει επόμενο (είναι τελευταίος στην ενδο-διατεταγμένη διέλευση). Για παράδειγμα, στο Σχήμα 1.2, ο επόμενος του 8 είναι το 10 (η ρίζα του δέντρου), αφού είναι ο κοντινότερος πρόγονος του 8 του οποίου το αριστερό παιδί (δηλ. το 5) είναι πρόγονος του 8. Ομοίως, ο επόμενος του 12 είναι το 16.

Ο παρακάτω ψευδοκώδικας υλοποιεί τη λειτουργία εύρεσης του επόμενου. Ο χρόνος εκτέλεσης χειρότερης περίπτωσης είναι $O(h)$, όπου h το ύψος του δέντρου, αφού στη χειρότερη περίπτωση θα χρειαστεί να ακολουθήσουμε ολόκληρο το μονοπάτι από τον πιο δεξιό κόμβο του δέντρου προς τη ρίζα.

```
Tree-Successor(x)
    if x.right ≠ NULL then
    /* Ελάχιστο υποδέντρου με ρίζα x.right */
        x ← x.right;
        while x.left ≠ NULL do
            x ← x.left;
        return(x);
    y ← x.par;
    while y ≠ NULL and x = y.right do
        x ← y; y ← x.par;
    return(y);
```

1.1.4. Ασκήσεις

Άσκηση 1.1. Έχουμε ένα ΔΔΑ που περιέχει σαν στοιχεία αριθμούς από το 1 μέχρι το 1000 (όχι κατ' ανάγκη όλους) και ψάχνουμε το 363. Ποιες από τις παρακάτω ακολουθίες δεν μπορούν να είναι οι ακολουθίες των στοιχείων που εξετάστηκαν κατά την αναζήτηση του 363; Εξηγείστε γιατί.

1. 2, 252, 401, 398, 330, 344, 397, 363.

2. 924, 220, 911, 244, 898, 258, 362, 363.

3. 925, 202, 911, 240, 912, 245, 363.

4. 2, 399, 387, 219, 266, 282, 381, 278, 363.

5. 935, 278, 347, 621, 299, 392, 358, 363.

Υπόδειξη. Η ιδιότητα των ΔΔΑ εισάγει περιορισμούς. Για παράδειγμα, αν το 363 είναι μεγαλύτερο (μικρότερο) από το τρέχον στοιχείο, το επόμενο στοιχείο στην ακολουθία πρέπει να είναι μεγαλύτερο (μικρότερο) από το τρέχον στοιχείο. Ακόμη, κατεβαίνοντας από τη ρίζα προς τα φύλλα, το εύρος του διαστήματος αναζήτησης μικραίνει συνεχώς. Το επόμενο στοιχείο πρέπει να ανήκει στο διάστημα αναζήτησης όπως αυτό έχει διαμορφωθεί από τους κόμβους που έχουμε επισκεφθεί μέχρι τότε.

Άσκηση 1.2. Έστω κόμβος x με δύο παιδιά που δεν είναι NULL-φύλλα. Τι είναι το αριστερό παιδί του succ(x) και τι το δεξί παιδί του pred(x) ;

Άσκηση 1.3. Ποια λειτουργία υλοποιεί ο παρακάτω ψευδοκώδικας (να υποθέσετε ότι εφαρμόζεται σε ένα ΔΔΑ T); Ποιος είναι ο χρόνος εκτέλεσης αυτού του ψευδοκώδικα αν το ΔΔΑ T έχει n στοιχεία;

```
Tree-Unknown-Operation(T)
    x ← Tree-Minimum(root(T));
    while x ≠ NULL do
        print(x.key);
        x ← Tree-Successor(x);
```

Άσκηση 1.4. Έστω x φύλλο (που περιέχει στοιχείο) ενός ΔΔΑ T του οποίου όλα τα στοιχεία είναι διαφορετικά, και έστω y ο πατέρας του x. Να δείξετε ότι το $y.key$ είναι είτε το μικρότερο στοιχείο του T που ξεπερνά το $x.key$ είτε το μεγαλύτερο στοιχείο του T που υπολείπεται του $x.key$.

1.2. Λειτουργίες Ενημέρωσης

Οι λειτουργίες εισαγωγής και διαγραφής οδηγούν στη μεταβολή του συνόλου των στοιχείων. Το ΔΔΑ πρέπει να αλλάξει ώστε το περιεχόμενό του να ανταποκρίνεται στο ενημερωμένο σύνολο στοιχείων. Πρέπει όμως να συνεχίσει να διατηρεί την ιδιότητα των ΔΔΑ.

1.2.1. Εισαγωγή Στοιχείου

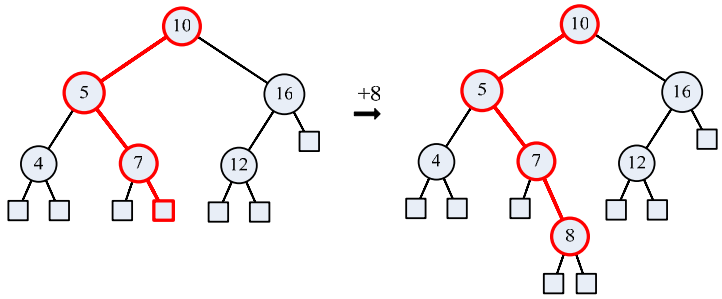
Κάθε νέο στοιχείο εισάγεται στη θέση του NULL-φύλλου που καταλήγει η αναζήτηση για αυτό το στοιχείο. Η εισαγωγή του νέου στοιχείου σαν φύλλο δεν επηρεάζει την ιδιότητα των ΔΔΑ. Στο Σχήμα 2.1 φαίνεται η εισαγωγή του στοιχείου 8 στο ΔΔΑ του Σχήματος 1.1. Η αναζήτηση του 8 διέρχεται από το μονοπάτι <10, 5, 7> και καταλήγει στο δεξιό NULL-παιδί του 7. Το νέο στοιχείο 8 εισάγεται σαν φύλλο στη θέση του δεξιού NULL-παιδιού του 7.

Ο παρακάτω ψευδοκώδικας υλοποιεί την εισαγωγής του στοιχείου k σε ένα ΔΔΑ T.

```
Tree-Insert(k)
    y ← NULL; x ← root(T);
    while x ≠ NULL do       /* Αναζήτηση του k */
        y ← x;
        if k < x.key then x ← x.left;
        else x ← x.right;
    z ← create_new_node();   /* Δημιουργία κόμβου */
    z.par ← y;      z.left ← NULL;
    z.right ← NULL; z.key ← k;
    if y = NULL then root(T) ← z;   /* Εισαγωγή */
    else if k < y.key then y.left ← z;
         else y.right ← z;
```

Η λειτουργία της εισαγωγής έχει χρόνο εκτέλεσης χειρότερης περίπτωσης $O(h)$, όπου h το ύψος του δέντρου, αφού η αναζήτηση ολοκληρώνεται σε χρόνο $O(h)$ και η εισαγωγή του νέου στοιχείου στην θέση που υποδεικνύει η αποτυχημένη αναζήτηση γίνεται σε χρόνο $O(1)$.

Αξίζει να σημειωθεί ότι η διαδοχική εισαγωγή μιας αύξουσας ή φθίνουσας ακολουθίας στοιχείων (π.χ., 1, 2, 3, 4, … ή 1000, 999, 998, 997, …) σε ένα αρχικά κενό δέντρο οδηγεί σε δέντρο με ύψος $\Theta(n)$ – γραμμικό στον αριθμό των στοιχείων (ουσιαστικά το δέντρο εκφυλλίζεται σε απλή διασυνδεδεμένη λίστα).



Σχήμα 1.2. Το στοιχείο 8 εισάγεται σαν φύλλο.

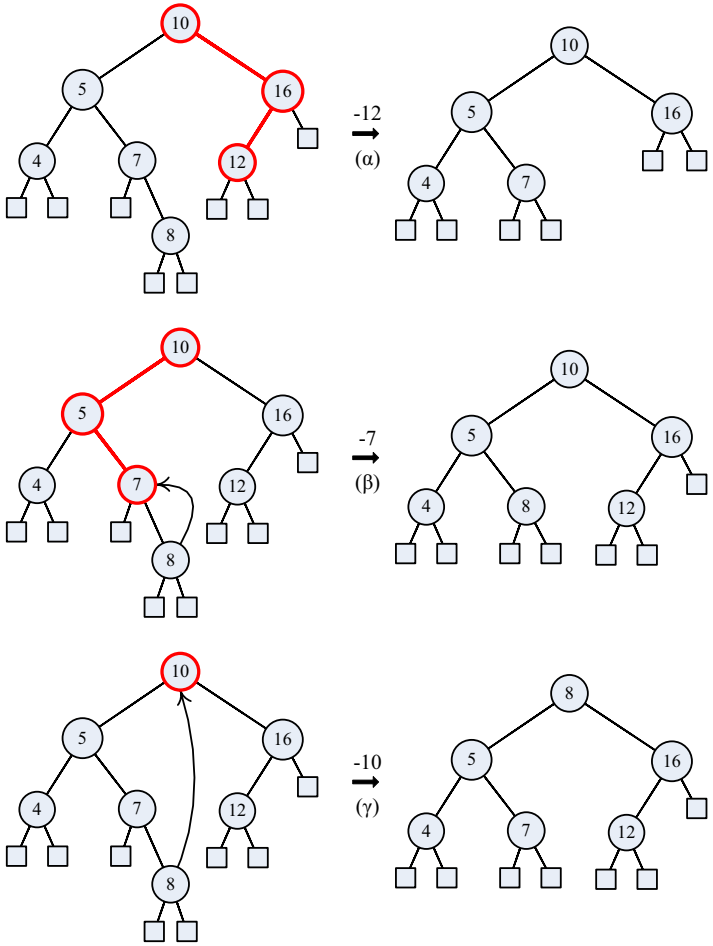
1.2.2. Διαγραφή Στοιχείου

Αρχικά χρησιμοποιούμε τη λειτουργία αναζήτησης για να εντοπίσουμε τον κόμβο που θα διαγραφεί. Αν και τα δύο παιδιά αυτού του κόμβου είναι NULL-φύλλα (Σχήμα 1.3.α), τον διαγράφουμε χωρίς άλλες ενέργειες. Αν ένα παιδί αυτού του κόμβου είναι NULL-φύλλο (Σχήμα 1.3.β), ο κόμβος απλώς «παραλείπεται» και το παιδί του που δεν είναι NULL παίρνει τη θέση του. Αν τέλος κανένα παιδί αυτού του κόμβου δεν είναι NULL–φύλλο (Σχήμα 1.3.γ), ο προηγούμενός του (ή ισοδύναμα, ο επόμενός του) στην ενδο-διατεταγμένη διέλευση αφήνει τη θέση του για να πάρει τη θέση του κόμβου που διαγράφεται. Είναι εύκολο να επιβεβαιώσουμε ότι σε όλες τις περιπτώσεις διατηρείται η ιδιότητα των ΔΔΑ.

Ο παρακάτω ψευδοκώδικας υλοποιεί τη λειτουργία διαγραφής ενός κόμβου z από ένα ΔΔΑ T. Θεωρούμε ότι ο κόμβος z έχει εντοπιστεί από τη λειτουργία αναζήτησης. Η κλήση `Tree-Predecessor(z)` επιστρέφει τον προηγούμενο κόμβο του z στην ενδο-διατεταγμένη διέλευση. Στη θέση της μπορεί να χρησιμοποιηθεί και η κλήση `Tree-Successor(z)`.

```
Tree-Delete(z)
    if z.left = NULL or z.right = NULL then y ← z;
    else y ← Tree-Predecessor(z);
    if y.left ≠ NULL then x ← y.left;
    else x ← y.right;
    if x ≠ NULL then x.par ← y.par;
```

```
if y.par = NULL then root(T) ← x;
else  if y = (y.par).left then (y.par).left ← x;
      else (y.par).right ← x;
if y ≠ z then z.key ← y.key;
```



Σχήμα 1.3. Παραδείγματα διαγραφής. Στο (α) διαγράφεται φύλλο (12), στο (β) διαγράφεται κόμβος με ένα παιδί που δεν είναι NULL (7) και το παιδί του (8) παίρνει τη θέση του, και στο (γ) διαγράφεται κόμβος με δύο παιδιά που δεν είναι NULL (10) και ο προηγούμενός του (8) παίρνει τη θέση του.

Η λειτουργία διαγραφής έχει χρόνο εκτέλεσης χειρότερης περίπτωσης $O(h)$ αφού οι λειτουργίες της αναζήτησης και της εύρεσης του προηγούμενου ή επόμενου κόμβου

ολοκληρώνονται σε $O(h)$ και η αντικατάσταση του κόμβου που διαγράφεται από τον κατάλληλο κόμβο γίνεται σε σταθερό χρόνο.

1.2.3. Ασκήσεις

Άσκηση 1.5. Να επιβεβαιώσετε ότι ο ψευδοκώδικας της `Tree-Delete` υλοποιεί τη λειτουργία της διαγραφής όπως αυτή περιγράφεται στην πρώτη παράγραφο της ενότητας 1.2.2.

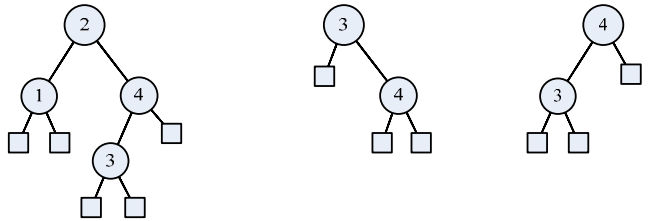
Άσκηση 1.6. Να αποδείξετε ότι η αντικατάσταση ενός κόμβου με δύο παιδιά που δεν είναι NULL από τον προηγούμενό του ή τον επόμενό του στην ενδο-διατεταγμένη διέλευση δεν επηρεάζει την ιδιότητα των ΔΔΑ. Να κάνετε το ίδιο για την περίπτωση που ο κόμβος που διαγράφεται έχει ένα παιδί που δεν είναι NULL, το οποίο και παίρνει τη θέση του.

Άσκηση 1.7. Μπορούμε να ταξινομήσουμε n στοιχεία σε αύξουσα σειρά εισάγοντας τα διαδοχικά σε ένα αρχικά κενό ΔΔΑ (με διαδοχικές κλήσεις της `Tree-Insert`) και εκτελώντας ενδο-διατεταγμένη διέλευση στο ΔΔΑ που παράγεται. Ποιος είναι ο χρόνος εκτέλεσης αυτής της διαδικασίας στη χειρότερη και στην καλύτερη περίπτωση;

Άσκηση 1.8. Είναι η λειτουργία της διαγραφής αντιμεταθετική; Δηλαδή ισχύει ότι για κάθε ΔΔΑ T και κάθε δύο κόμβους του x και y, όταν διαγράφουμε πρώτα το x και μετά το y από το T προκύπτει το ίδιο δέντρο όπως όταν διαγράφουμε πρώτα το y και μετά το x; Αν ναι, να αποδείξετε τον ισχυρισμό σας, αν όχι να δώσετε αντιπαράδειγμα.

Λύση. Η απάντηση είναι όχι. Η διαγραφή πρώτα του 2 και μετά του 1 από το ΔΔΑ του Σχήματος 1.4.α δίνει το ΔΔΑ του Σχήματος 1.4.β. Αντίθετα, η διαγραφή πρώτα του 1 και μετά του 2 από το ΔΔΑ του Σχήματος 1.4.α δίνει το ΔΔΑ του Σχήματος 1.4.γ, το οποίο είναι διαφορετικό από αυτό του Σχήματος 1.4.β.

Το συγκεκριμένο αντιπαράδειγμα είναι το ελάχιστο δυνατό, αφού κάθε αντιπαράδειγμα πρέπει να περιλαμβάνει τη διαγραφή ενός κόμβου με δύο παιδιά που δεν είναι NULL και δεν μπορεί να υπάρξει αντιπαράδειγμα με τρεις μόνο κόμβους.



Σχήμα 1.4. Παράδειγμα που αποδεικνύει ότι η λειτουργία της διαγραφής στα ΔΔΑ δεν είναι αντιμεταθετική.

2. Ζυγισμένα Δυαδικά Δέντρα Αναζήτησης

Είδαμε ότι στα Δυαδικά Δέντρα Αναζήτησης (ΔΔΑ) η χρονική πολυπλοκότητα για όλες τις πράξεις (αναζήτηση, εισαγωγή, διαγραφή, μέγιστο, ελάχιστο, προηγούμενο, επόμενο, κλπ.) είναι $O(h)$ – γραμμική στο *ύψος* του δέντρου. Με άλλα λόγια, τα ΔΔΑ αναζήτησης αποτελούν μία αποδοτική δομή δεδομένων ενόσω το ύψος τους παραμένει μικρό. Όμως στη χειρότερη περίπτωση ένα ΔΔΑ μπορεί να εκφυλιστεί σε γραμμική διασυνδεδεμένη λίστα. Τότε το ύψος του δέντρου είναι $\Theta(n)$ και ο μέσος χρόνος για την αναζήτηση ενός στοιχείου είναι γραμμικός στον αριθμό των αποθηκευμένων στοιχείων.

Για να εξασφαλίσουμε ότι το ύψος ενός ΔΔΑ παραμένει μικρό απαιτούμε το ύψος του αριστερού και του δεξιού υποδέντρου κάθε εσωτερικού κόμβου να είναι *περίπου* ίδια. Ένα τέτοιο ΔΔΑ ονομάζεται *ζυγισμένο* (*balanced*) και (είναι συνήθως εύκολο να αποδειχθεί ότι) έχει ύψος $O(\log n)$ [2].

Τα Ζυγισμένα Δυαδικά Δέντρα Αναζήτησης (ΖΔΔΑ – Balanced Binary Search Trees) αποτελούν ειδική περίπτωση των ΔΔΑ. Όλες οι πράξεις ερώτησης (δηλαδή, όλες οι πράξεις εκτός από την εισαγωγή και τη διαγραφή στοιχείου) πραγματοποιούνται όπως στα (μη-ζυγισμένα) ΔΔΑ. Οι πράξεις εισαγωγής και διαγραφής χρειάζονται κάποιες επιπλέον ενέργειες προκειμένου να διατηρηθεί η συνθήκη ζύγισης.

Από πλευράς απόδοσης, ο χρόνος χειρότερης περίπτωσης για τις πράξεις εισαγωγής, διαγραφής, αναζήτησης, εύρεσης του μέγιστου και του ελάχιστου, και εύρεσης του προηγούμενου και του επόμενου στα ΖΔΔΑ είναι $O(\log n)$ – λογαριθμικός στον αριθμό των αποθηκευμένων στοιχείων.

Ειδικότερα για τις πράξεις της εισαγωγής και της διαγραφής, η πρώτη φάση της γίνεται όπως στα (μη-ζυγισμένα) ΔΔΑ. Στην περίπτωση της εισαγωγής, κατεβαίνουμε το δέντρο από τη ρίζα προς τα φύλλα αναζητώντας το στοιχείο προς εισαγωγή. Αν το

[2] Το «*περίπου*» σημαίνει να διαφέρουν το πολύ κατά έναν σταθερό προσθετικό όρο ή έναν σταθερό πολλαπλασιαστικό παράγοντα. Ο ακριβής ορισμός εξαρτάται από τη *συνθήκη ζύγισης* με βάση την οποία ορίζεται κάθε ζυγισμένο δέντρο. Οι σταθερές που κρύβονται στο $O(\log n)$ εξαρτώνται από τον ακριβή ορισμό της συνθήκης ζύγισης.

στοιχείο δεν υπάρχει στο δέντρο, θα καταλήξουμε σε ένα NULL-φύλλο όπου και εισάγουμε το στοιχείο. Στην περίπτωση της διαγραφής, εντοπίζουμε (κατεβαίνοντας το δέντρο από τη ρίζα προς τα φύλλα) και διαγράφουμε τον κόμβο που περιέχει το στοιχείο προς διαγραφή. Αν ο κόμβος που διαγράψαμε έχει δύο παιδιά (όχι NULL), τοποθετούμε στη θέση του τον προηγούμενό του (ή τον επόμενό του) κόμβο στην ενδο-διατεταγμένη διέλευση (αυτός είναι ο κόμβος που περιέχει το προηγούμενο (αντίστοιχα επόμενο) στοιχείο του στοιχείου που διαγράφηκε), αν έχει ένα παιδί, τοποθετούμε στη θέση του το παιδί του, και αν δεν έχει κανένα παιδί (είναι φύλλο), τοποθετούμε στη θέση του ένα NULL-φύλλο.

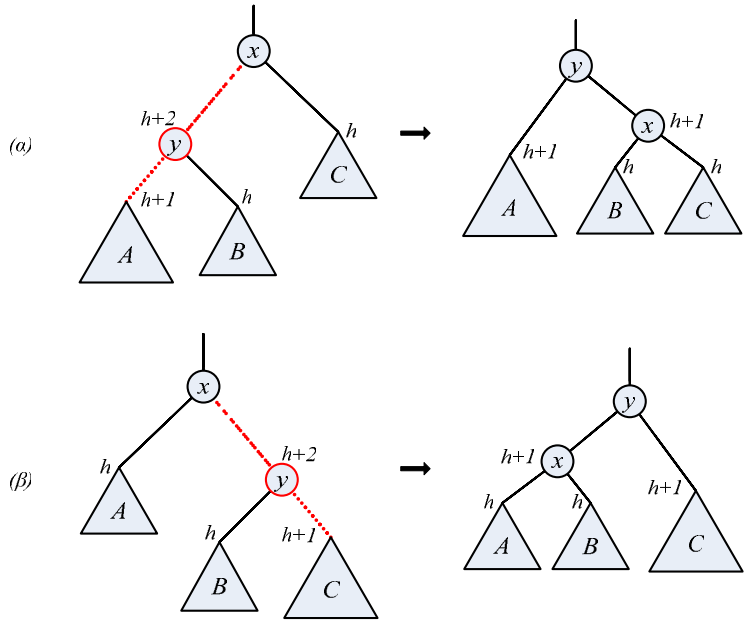
Όμως οι πράξεις της εισαγωγής και διαγραφής μπορούν να μεταβάλλουν το ύψος κάποιων υποδέντρων και συνεπώς να οδηγήσουν σε δέντρο που παραβιάζει τη συνθήκη ζύγισης. Στα ζυγισμένα δέντρα, μια εισαγωγή (διαγραφή) ακολουθείται πάντα από έλεγχο της συνθήκης ζύγισης στο μονοπάτι από το νέο (διαγραμμένο) κόμβο προς τη ρίζα. Με άλλα λόγια, ο έλεγχος και η αποκατάσταση της συνθήκης ζύγισης πραγματοποιούνται συνήθως ανεβαίνοντας το δέντρο από τα φύλλα προς τη ρίζα[3]. Όπου διαπιστωθεί παραβίαση, η συνθήκη ζύγισης αποκαθίσταται με τις λεγόμενες *επαναζυγιστικές πράξεις* (rebalancing operations). Οι πιο σημαντικές επαναζυγιστικές πράξεις είναι οι *περιστροφές* (rotations), που καλούνται και *δομικές* πράξεις επειδή μεταβάλλουν τη δομή του δέντρου. Υπάρχουν και μη-δομικές επαναζυγιστικές πράξεις (π.χ. ενημέρωση πληροφορίας σχετικής με τη συνθήκη ζύγισης που φυλάσσεται στους κόμβους) οι οποίες εξαρτώνται από το είδος του ΖΔΔΑ. Μια επαναζυγιστική πράξη ονομάζεται *τερματική* όταν αποκαθιστά οριστικά τη συνθήκη ζύγισης, και *μη-τερματική* διαφορετικά. Η εισαγωγή (διαγραφή) ολοκληρώνεται με μία τερματική επαναζυγιστική πράξη αφού μετά από αυτή δεν χρειάζεται να συνεχιστεί ο έλεγχος της συνθήκης ζύγισης προς τη ρίζα.

Μια περιστροφή συμβαίνει όταν εντοπίσουμε έναν κόμβο (έστω x) του οποίου τα δύο υποδέντρα παραβιάζουν τη συνθήκη ζύγισης. Δηλαδή, το ύψος του ενός υποδέντρου (αυτό που έχει ρίζα το y) είναι μεγαλύτερο από όσο επιτρέπει η συνθήκη ζύγισης σε

[3] Υπάρχουν περιπτώσεις ζυγισμένων δέντρων όπου για να βελτιστοποιηθεί η απόδοση, η αποκατάσταση της συνθήκης ζύγισης γίνεται ταυτόχρονα με την αναζήτηση, δηλαδή κατεβαίνοντας το δέντρο από τη ρίζα προς τα φύλλα. Σε αυτή την περίπτωση λέμε ότι η εισαγωγή (διαγραφή) και η επαναζύγιση γίνονται *σε ένα πέρασμα*.

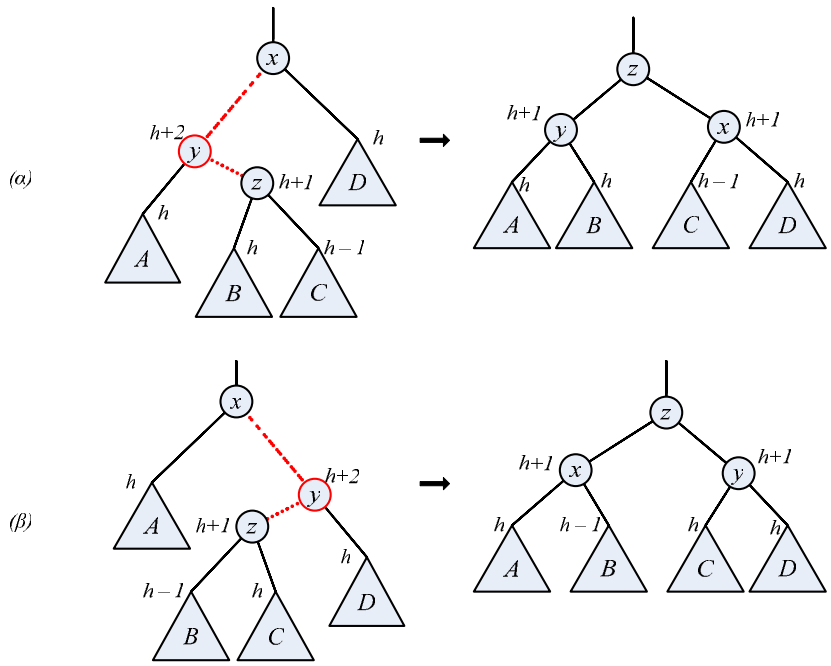
σχέση με το ύψος του άλλου υποδέντρου. Η κατάλληλη περιστροφή σε αυτό το σημείο αποκαθιστά τη συνθήκη ζύγισης.

Υπάρχουν δύο είδη περιστροφών, η *απλή* (single rotation) και η *διπλή* (double rotation). Κάθε είδος εμφανίζεται σε δύο εκδοχές, την *αριστερή* και τη *δεξιά*. Η διπλή περιστροφή μπορεί να υλοποιηθεί με δύο αντίρροπες απλές περιστροφές.



Σχήμα 2.1. Παραδείγματα δεξιάς και αριστερής απλής περιστροφής.

Η απλή περιστροφή (Σχήμα 2.1) εκτελείται όταν οι *δύο πρώτες* ακμές στο *μακρύτερο* μονοπάτι από το x προς τα φύλλα ακολουθούν την ίδια κατεύθυνση (σχηματίζουν ευθεία – το δέντρο «κρεμάει» προς τα άκρα). Όταν οι δύο πρώτες ακμές του μακρύτερου μονοπατιού είναι προς τα αριστερά (το δέντρο «κρεμάει» προς τα αριστερά), εκτελείται *δεξιά* απλή περιστροφή (Σχήμα 2.1.α). Όταν οι δύο πρώτες ακμές είναι προς τα δεξιά (το δέντρο «κρεμάει» προς τα δεξιά), εκτελείται *αριστερή* απλή περιστροφή (Σχήμα 2.1.β).



Σχήμα 2.2. Παραδείγματα δεξιάς και αριστερής διπλής περιστροφής.

Η διπλή περιστροφή (Σχήμα 2.2) εκτελείται όταν οι *δύο πρώτες* ακμές στο *μακρύτερο* μονοπάτι από το x προς τα φύλλα ακολουθούν διαφορετική κατεύθυνση (σχηματίζουν γωνία – το δέντρο «κρεμάει» προς τη μέση). Όταν η πρώτη ακμή στο μακρύτερο μονοπάτι είναι προς τα αριστερά και η δεύτερη προς τα δεξιά, εκτελείται *δεξιά* διπλή περιστροφή (Σχήμα 2.2.α). Όταν η πρώτη ακμή είναι προς τα δεξιά και η δεύτερη προς τα αριστερά, εκτελείται *αριστερή* διπλή περιστροφή (Σχήμα 2.2.β).

Εξετάζοντας τα Σχήματα 2.1 και 2.2, μπορούμε να επιβεβαιώσουμε ότι οι περιστροφές διατηρούν την ιδιότητα των ΔΔΑ[4]. Δηλαδή αν εκτελέσουμε μια οποιαδήποτε περιστροφή σε ένα ΔΔΑ, το αποτέλεσμα θα είναι επίσης ΔΔΑ.

[4] Υπενθυμίζεται η ιδιότητα που χαρακτηρίζει τα ΔΔΑ: όλα τα στοιχεία στο αριστερό υποδέντρο ενός κόμβου x είναι μικρότερα ή ίσα από το στοιχείο του x, και όλα τα στοιχεία στο δεξιό υποδέντρο ενός κόμβου x είναι μεγαλύτερα ή ίσα από το στοιχείο του x.

2.1. AVL Δέντρα

Τα AVL δέντρα οφείλουν το όνομά τους στα αρχικά των Adelson-Velskii και Landis, των επιστημόνων που τα πρότειναν το 1962.

Ένα ΔΔΑ είναι ένα AVL δέντρο αν τα ύψη των δύο υποδέντρων κάθε εσωτερικού κόμβου διαφέρουν το πολύ κατά 1. Με άλλα λόγια, η συνθήκη ζύγισης των AVL δέντρων επιτρέπει την ελάχιστη δυνατή διαφορά μεταξύ του ύψους των δύο υποδέντρων κάθε εσωτερικού κόμβου[5]. Τα AVL δέντρα είναι μία ειδική κατηγορία ΖΔΔΑ και συνεπώς ισχύει για αυτά ότι έχουμε αναφέρει για τα ΖΔΔΑ. Στη συνέχεια θα αποδείξουμε ότι το ύψος ενός AVL δέντρου είναι $\Theta(\log n)$.

Λήμμα 2.1. Έστω h το ύψος ενός AVL δέντρου με n στοιχεία. Είναι

$$\log(n+1) \le h \le 1.441 \log n$$

Απόδειξη. Το ελάχιστο ύψος για δεδομένο αριθμό στοιχείων συμβαίνει όταν έχουμε πλήρες δυαδικό δέντρο. Σε αυτή την περίπτωση είναι $n = 2^h - 1 \Rightarrow h = \log(n+1)$. Συνεπώς, πάντα είναι $h \ge \log(n+1)$.

Για να φράξουμε άνω το ύψος, έστω $T(h)$ ο ελάχιστος αριθμός στοιχείων σε ένα AVL δέντρο με ύψος h. Είναι $T(1) = 1$ και $T(2) \ge 2$. Για κάθε ύψος h, ισχύει η αναδρομική σχέση

$$T(h) \ge 1 + T(h-1) + T(h-2)$$

αφού η ρίζα πρέπει να έχει ένα υποδέντρο ύψους h–1 και ένα υποδέντρο ύψους τουλάχιστον h–2 (το πρώτο για να έχει η ρίζα ύψος h και το δεύτερο γιατί το επιβάλει η AVL συνθήκη ζύγισης). Συνεπώς πρέπει να είναι $T(h) \ge \text{Fib}(h+1)$, όπου $\text{Fib}(h+1)$ είναι ο $(h+1)$-οστός όρος της ακολουθίας Fibonacci[6].

[5] Παρατηρείστε ότι αν απαιτήσουμε τα ύψη των δύο υποδέντρων κάθε εσωτερικού κόμβου να είναι ίσα, καταλήγουμε στο πλήρες δυαδικό δέντρο. Όμως το πλήρες δυαδικό μπορεί να κατασκευαστεί μόνο όταν ο αριθμός των στοιχείων n είναι της μορφής $n = 2^h - 1$, όπου h το ύψος του δέντρου. Υπενθυμίζουμε επίσης ότι στο ύψος του δέντρου συμπεριλαμβάνουμε τα NULL φύλλα.

[6] Για την ακολουθία Fibonacci, βλέπε http://mathworld.wolfram.com/FibonacciNumber.html .

Ο αριθμός n των στοιχείων σε AVL δέντρο δεδομένου ύψους h δεν μπορεί να είναι μικρότερος από το $T(h)$ βάσει του ορισμού του $T(h)$. Συνεπώς,

$$n \geq T(h) \geq \text{Fib}(h+1)$$

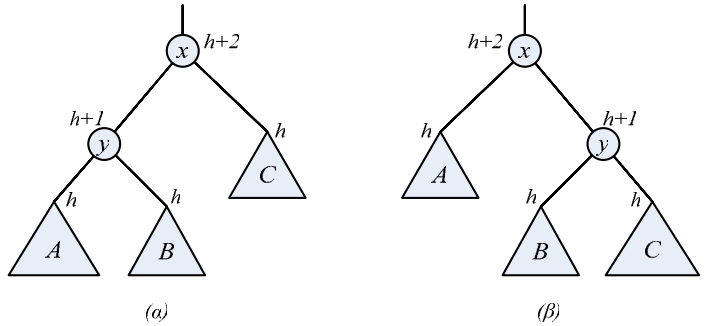
Το ζητούμενο άνω φράγμα προκύπτει από τον κλειστό τύπο της ακολουθίας Fibonacci. ■

Αφού το ύψος ενός AVL δέντρου είναι $\Theta(\log n)$, η χρονική πολυπλοκότητα των πράξεων αναζήτησης, μέγιστου, ελάχιστου, προηγούμενου, και επόμενου είναι $O(\log n)$ στη χειρότερη περίπτωση.

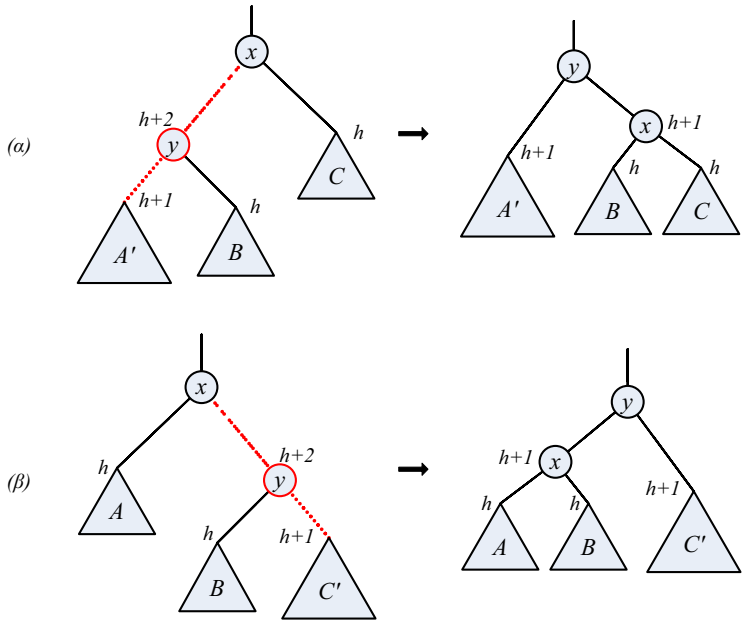
2.1.1. Εισαγωγή σε AVL Δέντρο

Αρχικά εισάγουμε το νέο στοιχείο σαν φύλλο, όπως ακριβώς στα μη-ζυγισμένα ΔΔΑ. Στη συνέχεια ανεβαίνουμε το μονοπάτι από το νέο στοιχείο προς τη ρίζα, ενημερώνοντας τα ύψη και ελέγχοντας αν συνεχίζει να ισχύει η AVL συνθήκη ζύγισης. Στον πρώτο κόμβο (πλησιέστερο προς νέο στοιχείο) που παραβιάζει την AVL συνθήκη ζύγισης (δηλαδή τα ύψη των δύο υποδέντρων του διαφέρουν κατά 2), εκτελούμε την κατάλληλη *περιστροφή*. Όπως θα δούμε, οι περιστροφές κατά την εισαγωγή σε AVL δέντρο είναι πάντα τερματικές. Επομένως, τερματίζουμε τη διαδικασία εισαγωγής μετά από την περιστροφή. Αν δεν βρούμε κόμβο που να παραβιάζει την AVL συνθήκη, η διαδικασία εισαγωγής τερματίζεται όταν φτάσουμε σε κόμβο του οποίου το ύψος παραμένει αμετάβλητο ή όταν φτάσουμε στη ρίζα.

Απομένει να εξηγήσουμε ποιο είδος περιστροφής εκτελείται σε κάθε περίπτωση. Η εισαγωγή ενός στοιχείου μπορεί να αυξήσει το ύψος ενός υποδέντρου το πολύ κατά 1. Για να παραβιάζει λοιπόν ένας κόμβος x (έστω σε ύψος h+2 πριν την εισαγωγή του νέου στοιχείου) την AVL συνθήκη πρέπει (πριν την εισαγωγή) να είχε ένα υποδέντρο ύψους h και ένα υποδέντρο ύψους h+1 (Σχήμα 2.3). Επιπλέον, η εισαγωγή του νέου στοιχείου πρέπει να γίνει στο υποδέντρο ύψους h+1 και να προκάλεσει αύξηση του ύψους του υποδέντρου σε h+2.



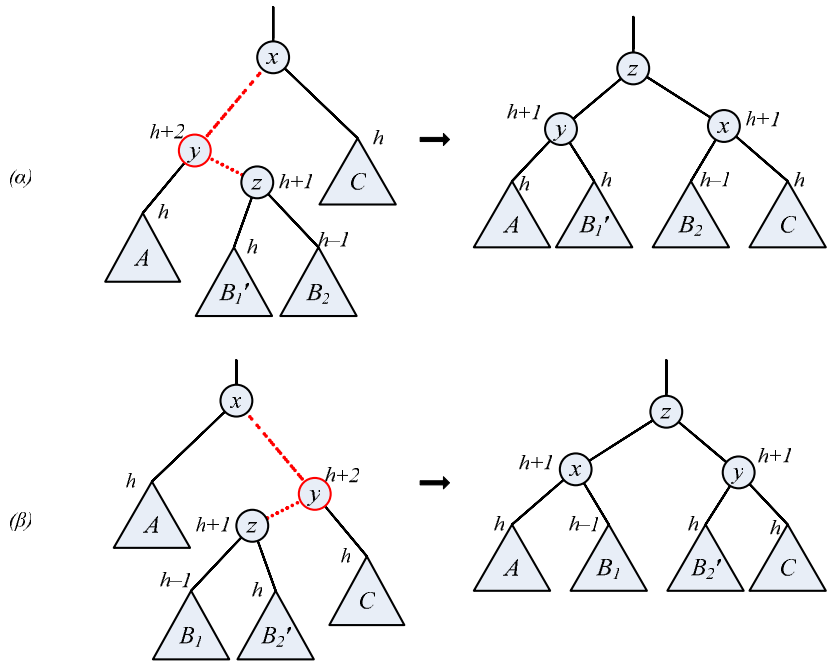
Σχήμα 2.3. Κάθε εισαγωγή στοιχείου στο υποδέντρο με ρίζα το y που επιφέρει αύξηση του ύψους του y σε h+2 οδηγεί σε παραβίαση της AVL συνθήκης ζύγισης στο x. Για να αποκατασταθεί η AVL συνθήκη πρέπει να καταφύγουμε σε περιστροφή.



Σχήμα 2.4. Μία δεξιά απλή περιστροφή αποκαθιστά την AVL συνθήκη όταν το ύψος του υποδέντρου A αυξάνεται σε h+1. Ομοίως, μία αριστερή απλή περιστροφή αποκαθιστά την AVL συνθήκη όταν το ύψος του υποδέντρου C αυξάνεται σε h+1.

Η εισαγωγή ενός στοιχείου στο υποδέντρο A του Σχήματος 2.3.α που προκαλεί την αύξηση του ύψους σε h+1 οδηγεί σε παραβίαση της AVL συνθήκης στο x. Η συνθήκη

αποκαθίσταται με μία απλή δεξιά περιστροφή (Σχήμα 2.4.α). Ομοίως, η εισαγωγή ενός στοιχείου στο υποδέντρο C του Σχήματος 2.3.β που προκαλεί την αύξηση του ύψους σε h+1 οδηγεί σε παραβίαση της AVL συνθήκης στο x. Η συνθήκη αποκαθίσταται με μία απλή αριστερή περιστροφή (Σχήμα 2.4.β). Και στις δύο περιπτώσεις η περιστροφή είναι *απλή* γιατί οι δύο πρώτες ακμές στο μακρύτερο μονοπάτι από το x προς τα φύλλα ακολουθούν την ίδια κατεύθυνση. Η περιστροφή είναι *τερματική* γιατί το y, που παίρνει τη θέση του x μετά την περιστροφή, έχει (μετά την περιστροφή) ύψος h+2, όσο ακριβώς είχε και το x πριν την περιστροφή (Σχήμα 2.4).



Σχήμα 2.5. Μία διπλή περιστροφή (δεξιά στο (α) και αριστερή στο (β)) αποκαθιστά την AVL συνθήκη όταν το ύψος του υποδέντρου B αυξάνεται σε h+1. Το υποδέντρο B του Σχήματος 2.3 έχει αναλυθεί στη ρίζα z και στα υποδέντρα B_1 και B_2.

Η εισαγωγή ενός στοιχείου στο υποδέντρο B του Σχήματος 2.3 που προκαλεί την αύξηση του ύψους σε h+1 οδηγεί σε παραβίαση της AVL συνθήκης στο x. Στην περίπτωση που το y είναι το αριστερό παιδί του x (Σχήμα 2.3.α), η συνθήκη αποκαθίσταται με μία διπλή δεξιά περιστροφή (Σχήμα 2.5.α). Όταν το y είναι το δεξιό παιδί του x (Σχήμα 2.3.β), η συνθήκη αποκαθίσταται με μία διπλή αριστερή περιστροφή

(Σχήμα 2.5.β). Και στις δύο περιπτώσεις η περιστροφή είναι *διπλή* γιατί οι δύο πρώτες ακμές στο μακρύτερο μονοπάτι από το x προς τα φύλλα ακολουθούν διαφορετική κατεύθυνση. Η περιστροφή είναι *τερματική* γιατί το z (η ρίζα του υποδέντρου B πριν την περιστροφή), που παίρνει τη θέση του x μετά την περιστροφή, έχει (μετά την περιστροφή) ύψος h+2, όσο ακριβώς είχε και το x πριν την περιστροφή (Σχήμα 2.5).

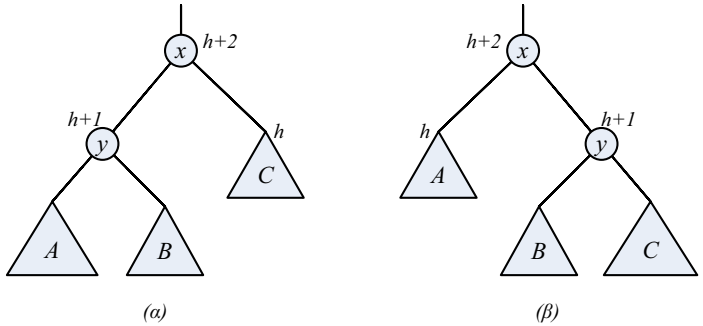
Συνολικά ο χρόνος χειρότερης περίπτωσης για την εισαγωγή στοιχείου σε ένα AVL δέντρο είναι $O(\log n)$: λογαριθμικός χρόνος για την εύρεσης της θέσης που εισάγεται το στοιχείο και το πολύ λογαριθμικός αριθμός από μη-δομικές πράξεις που ανανεώνουν τα ύψη των κόμβων στο μονοπάτι από τη θέση εισαγωγής προς τη ρίζα. Κάθε εισαγωγή μπορεί να προκαλέσει το πολύ 1 απλή ή διπλή περιστροφή (με άλλα λόγια, το πολύ 1 *δομική* επαναζυγιστική πράξη).

2.1.2. *Διαγραφή από ένα AVL Δέντρο*

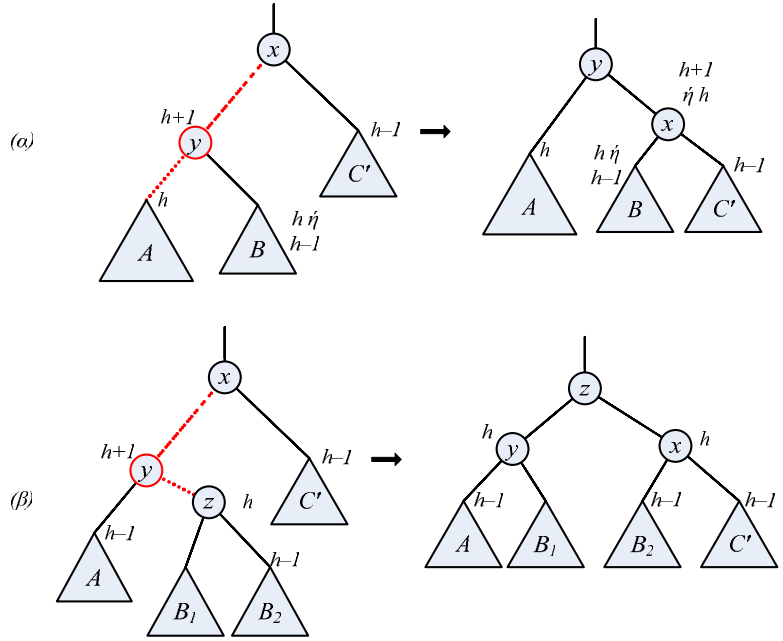
Αρχικά διαγράφουμε το στοιχείο όπως ακριβώς στα μη-ζυγισμένα ΔΔΑ. Στη συνέχεια ανεβαίνουμε το μονοπάτι από το σημείο της διαγραφής προς τη ρίζα[7] ενημερώνοντας τα ύψη και ελέγχοντας αν συνεχίζει να ισχύει η AVL συνθήκη ζύγισης. Όταν φτάσουμε σε κόμβο που παραβιάζει την AVL συνθήκη ζύγισης, εκτελούμε την κατάλληλη *περιστροφή*. Στην περίπτωση της διαγραφής, μια περιστροφή μπορεί να μην είναι τερματική. Επομένως συνεχίζουμε την ίδια διαδικασία μέχρι να βρούμε κόμβο του οποίου το ύψος δεν επηρεάζεται από τη διαγραφή ή να φτάσουμε στη ρίζα.

Η διαγραφή ενός στοιχείου μπορεί να μειώσει το ύψος ενός υποδέντρου το πολύ κατά 1. Για να παραβιάζει λοιπόν ένας κόμβος x (έστω σε ύψος h+2 πριν τη διαγραφή) την AVL συνθήκη πρέπει να είχε (πριν τη διαγραφή) ένα υποδέντρο ύψους h και ένα υποδέντρο ύψους h+1 (Σχήμα 2.6). Επιπλέον, η διαγραφή πρέπει να γίνει στο υποδέντρο ύψους h και να προκαλέσει τη μείωση του ύψους του υποδέντρου σε h–1.

[7] Όταν ο κόμβος που διαγράφεται έχει δύο παιδιά, το πραγματικό σημείο της διαγραφής είναι η αρχική θέση του (προηγούμενου ή επόμενου στην ενδο-διατεταγμένη διέλευση) κόμβου που πήρε τη θέση του κόμβου που διαγράφηκε.



Σχήμα 2.6. Κάθε διαγραφή στοιχείου από το υποδέντρο C στο (α) και A στο (β) που επιφέρει μείωση του ύψους σε h–1 οδηγεί σε παραβίαση της AVL συνθήκης ζύγισης στο x. Για να αποκατασταθεί η AVL συνθήκη πρέπει να καταφύγουμε σε περιστροφή.

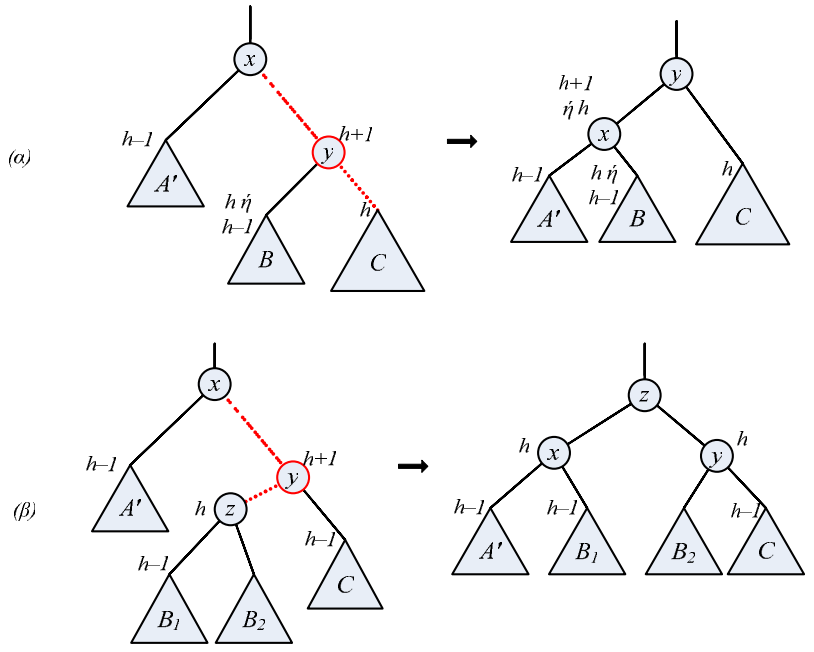


Σχήμα 2.7. Μία δεξιά απλή (α) ή διπλή (β) περιστροφή αποκαθιστά την AVL συνθήκη όταν το ύψος του υποδέντρου C μειώνεται σε h–1 εξαιτίας μιας διαγραφής.

Η διαγραφή ενός στοιχείου από το υποδέντρο C του Σχήματος 2.6.α που προκαλεί τη μείωση του ύψους σε h–1 οδηγεί σε παραβίαση της AVL συνθήκης στο x. Όταν το

ύψος του υποδέντρου A είναι h, η συνθήκη αποκαθίσταται με μία απλή δεξιά περιστροφή (Σχήμα 2.7.α). Η περιστροφή είναι *απλή* γιατί οι δύο πρώτες ακμές στο μακρύτερο μονοπάτι από το x προς τα φύλλα ακολουθούν την ίδια κατεύθυνση. Όταν το υποδέντρο B έχει ύψος h, η περιστροφή είναι *τερματική* γιατί το y, που παίρνει τη θέση του x μετά την περιστροφή, έχει (μετά την περιστροφή) ύψος h+2, όσο ακριβώς είχε και το x πριν την περιστροφή. Όταν το υποδέντρο B έχει ύψος h–1, η περιστροφή είναι *μη-τερματική* γιατί το y έχει (μετά την περιστροφή) ύψος h+1. Συνεπώς πρέπει να συνεχίσουμε προς τη ρίζα ενημερώνοντας τα ύψη των προγόνων του y.

Όταν το ύψος του υποδέντρου A είναι h–1, το υποδέντρο B πρέπει να έχει ύψος h. Η συνθήκη αποκαθίσταται με μία διπλή δεξιά περιστροφή (Σχήμα 2.7.β). Η περιστροφή είναι *διπλή* γιατί οι δύο πρώτες ακμές στο μακρύτερο μονοπάτι από το x προς τα φύλλα ακολουθούν διαφορετική κατεύθυνση. Η περιστροφή είναι *μη-τερματική* γιατί το z (η ρίζα του υποδέντρου B πριν την περιστροφή), που παίρνει τη θέση του x μετά την περιστροφή, έχει (μετά την περιστροφή) ύψος h+1.



Σχήμα 2.8. Μία αριστερή απλή (α) ή διπλή (β) περιστροφή αποκαθιστά την AVL συνθήκη όταν το ύψος του υποδέντρου A μειώνεται σε h–1 εξαιτίας μιας διαγραφής.

Ομοίως, η διαγραφή ενός στοιχείου από το υποδέντρο A του Σχήματος 2.6.β που προκαλεί τη μείωση του ύψους σε h–1 οδηγεί σε παραβίαση της AVL συνθήκης στο x. Όταν το ύψος του υποδέντρου C είναι h, η συνθήκη αποκαθίσταται με μία απλή αριστερή περιστροφή (Σχήμα 2.8.α). Όταν το υποδέντρο B έχει ύψος h, η περιστροφή είναι *τερματική*, ενώ όταν το υποδέντρο B έχει ύψος h–1, η περιστροφή είναι *μη-τερματική*. Όταν το ύψος του υποδέντρου C (Σχήμα 2.6.β) είναι h–1, η συνθήκη αποκαθίσταται με μία διπλή αριστερή περιστροφή (Σχήμα 2.8.β). Η περιστροφή είναι *μη-τερματική* γιατί το z έχει (μετά την περιστροφή) ύψος h+1.

Συνολικά ο χρόνος χειρότερης περίπτωσης για τη διαγραφή ενός στοιχείου από ένα AVL δέντρο είναι $O(\log n)$: λογαριθμικός χρόνος για την εύρεση του κόμβου που θα διαγραφή και του κόμβου που θα πάρει τη θέση του και το πολύ λογαριθμικός αριθμός από επαναζυγιστικές πράξεις στο μονοπάτι από το σημείο διαγραφής προς τη ρίζα. Ειδικότερα σε σχέση με τις δομικές επαναζυγιστικές πράξεις, κάθε φορά που εκτελείται μία μη-τερματική περιστροφή, η συνθήκη ζύγισης αποκαθίσταται στο συγκεκριμένο υποδέντρο. Επομένως, αν υπάρξει επόμενη περιστροφή, αυτή θα συμβεί σε υψηλότερο επίπεδο. Αυτό σημαίνει ότι ο αριθμός των δομικών επαναζυγιστικών πράξεων είναι $O(\log n)$ στη χειρότερη περίπτωση.

2.1.3. Παραδείγματα

Για συγκεκριμένα παραδείγματα εισαγωγής και διαγραφής στοιχείων σε AVL δέντρα χρησιμοποιείστε το Java applet στη διεύθυνση (ενεργοποιείστε την επιλογή AVL στα δεξιά):

http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

Για παράδειγμα, δοκιμάστε να εισάγετε τα 6, 9, 14, 17, 5, 7, 16, 20, 18, 19, 4, και 11 σε ένα αρχικά κενό δέντρο. Παρατηρείστε τις περιπτώσεις απλών και διπλών περιστροφών. Στη συνέχεια δοκιμάστε να διαγράψετε τα 7, 11, 14, 4, 5, 20, 19.

Επίσης δοκιμάστε να εισάγετε τα 63, 30, 36, 31, 12, 50, 35, 5, 27, 59, 43, και 17 σε ένα αρχικά κενό δέντρο. Στη συνέχεια να σβήσετε τα 36 και 43.

2.2. Κόκκινα-Μαύρα Δέντρα

Τα Κόκκινα-Μαύρα δέντρα (Red-Black trees) είναι από τα πλέον δημοφιλή ΖΔΔΑ. Προτάθηκαν από τον Bayer το 1972, ο οποίος αρχικά τους έδωσε το όνομα Συμμετρικά Δυαδικά B-Δέντρα (Symmetric Binary B-trees). Οι Guibas και Sedgewick μελέτησαν τις ιδιότητές τους σε βάθος και εισήγαγαν την ιδέα του διαχωρισμού των κόμβων σε κόκκινους και μαύρους.

Ένα ΔΔΑ ονομάζεται Κόκκινο-Μαύρο δέντρο (ΚΜ-δέντρο) όταν ικανοποιεί τις ακόλουθες ιδιότητες:

(1) Κάθε κόμβος είναι είτε *κόκκινος* είτε *μαύρος*.

(2) Η *ρίζα* του δέντρου είναι *μαύρη*.

(3) Όλα τα NULL-φύλλα είναι *μαύρα*.

(4) Κάθε κόκκινος κόμβος έχει μαύρα παιδιά (ή ισοδύναμα, σε κανένα μονοπάτι από τη ρίζα προς τα φύλλα δεν υπάρχουν δύο διαδοχικοί κόκκινοι κόμβοι).

(5) Για κάθε εσωτερικό κόμβο, όλα τα μονοπάτια από τον κόμβο προς τα υποκείμενα φύλλα περιέχουν τον ίδιο αριθμό μαύρων κόμβων.

Στα ΚΜ-δέντρα, η συνθήκη ζύγισης είναι κωδικοποιημένη στις ιδιότητες (4) και (5). Για να αντιληφθούμε τη συνθήκη ζύγισης που προκύπτει από τις (4) και (5), ορίζουμε το *μαύρο ύψος* ενός εσωτερικού κόμβου x, που συμβολίζεται με $bh(x)$, σαν τον αριθμό των μαύρων κόμβων σε ένα μονοπάτι από το x προς υποκείμενο φύλλο μειωμένο κατά ένα. Εξ΄ ορισμού, το ύψος του x δεν μπορεί να είναι μικρότερο από το μαύρο ύψος του. Από την άλλη πλευρά, το ύψος του x δεν μπορεί να ξεπερνά το διπλάσιο του μαύρου ύψους του γιατί κάθε κόκκινος κόμβος παρεμβάλλεται ανάμεσα σε δύο μαύρους. Συνεπώς, σε κάθε ΚΜ-δέντρο, τα ύψη των δύο υποδέντρων κάθε εσωτερικού κόμβου x διαφέρουν το πολύ κατά ένα πολλαπλασιαστικό παράγοντα 2 (ή ισοδύναμα το ύψος του ψηλότερου υποδέντρου είναι το πολύ διπλάσιο του ύψους του χαμηλότερου υποδέντρου).

Στη συνέχεια αποδεικνύουμε ότι αυτή η συνθήκη συνεπάγεται ότι το ύψος ενός ΚΜ-δέντρου με n στοιχεία είναι $\Theta(\log n)$.

Λήμμα 2.2. Έστω h το ύψος ενός ΚΜ-δέντρου με n στοιχεία. Είναι

$$\log(n+1) \le h \le 2\log(n+1)$$

Απόδειξη. Το κάτω φράγμα προκύπτει επειδή το ελάχιστο ύψος για δεδομένο αριθμό στοιχείων συμβαίνει όταν έχουμε το πλήρες δυαδικό δέντρο (βλ. Λήμμα 2.1). Για το άνω φράγμα, έστω h το ύψος και bh το μαύρο ύψος του δέντρου (τα ύψη του δέντρου είναι αυτά της ρίζας). Οι ιδιότητες (4) και (5) συνεπάγονται ότι $h \le 2bh$.

Αφού σε κάθε μονοπάτι από τη ρίζα προς υποκείμενο φύλλο, υπάρχουν ακριβώς bh (μαύροι) εσωτερικοί κόμβοι (και ακόμη ένας που είναι NULL-φύλλο), ο αριθμός n των στοιχείων σε ΚΜ-δέντρο με μαύρο ύψος bh είναι τουλάχιστον

$$n \ge 2^{bh} - 1 \Rightarrow bh \le \log(n+1)$$

Το άνω φράγμα στο ύψος του δέντρου προκύπτει από την ανισότητα ότι $h \le 2bh$. ■

Τα AVL έχουν αυστηρότερη συνθήκη ζύγισης από τα ΚΜ-δέντρα. Αυτός είναι ο λόγος που το μέγιστο ύψος ενός AVL δέντρου με n στοιχεία είναι λίγο μικρότερο από το μέγιστο ύψος ενός ΚΜ-δέντρου με τον ίδιο αριθμό στοιχείων. Από την άλλη πλευρά, περιμένει κανείς η χαλαρότερη συνθήκη ζύγισης των ΚΜ-δέντρων να απλοποιεί τις επαναζυγιστικές πράξεις και τις κάνει σπανιότερες.

Οι πράξεις αναζήτησης, μέγιστου, ελάχιστου, προηγούμενου, και επόμενου υλοποιούνται όπως ακριβώς στα (μη-ζυγισμένα) ΔΔΑ και έχουν χρόνο εκτέλεσης χειρότερης περίπτωσης $O(\log n)$, αφού το μέγιστο ύψος ενός ΚΜ-δέντρου είναι λογαριθμικό.

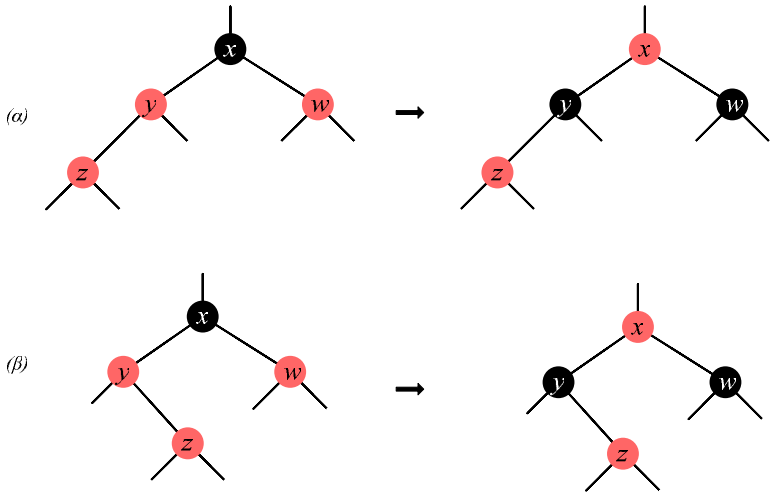
2.2.1. Εισαγωγή σε ΚΜ-Δέντρο

Εισάγουμε το νέο στοιχείο σαν φύλλο (όχι NULL) όπως στα (μη-ζυγισμένα) ΔΔΑ. Ο νέος κόμβος χρωματίζεται πάντα κόκκινος και έχει ύψος 1 και μαύρο ύψος επίσης 1.

Η εισαγωγή ενός κόκκινου κόμβου δεν παραβιάζει τις ιδιότητες (1), (3), και (5). Ειδικότερα για την ιδιότητα (5), η εισαγωγή ενός κόκκινου κόμβου δεν μεταβάλει τον αριθμό των μαύρων κόμβων σε κανένα μονοπάτι από τη ρίζα προς τα φύλλα. Η ιδιότητα (2) παραβιάζεται όταν το δέντρο είναι κενό (πριν την εισαγωγή) ή σαν αποτέλεσμα μιας ακολουθίας μη-δομικών επαναζυγιστικών πράξεων. Σε κάθε περίπτωση, η ιδιότητα (2) αποκαθίσταται εύκολα χρωματίζοντας τη ρίζα μαύρη (βλ. Άσκηση 2.1).

Το σημαντικότερο πρόβλημα προκύπτει από την παραβίαση της ιδιότητας (4), αφού η εισαγωγή ενός νέου κόκκινου κόμβου μπορεί να οδηγήσει στην εμφάνιση δυο διαδοχικών κόκκινων κόμβων στο μονοπάτι από τη θέση εισαγωγής προς τη ρίζα. Η αποκατάσταση της ιδιότητας (4) γίνεται με κατάλληλες επαναζυγιστικές πράξεις.

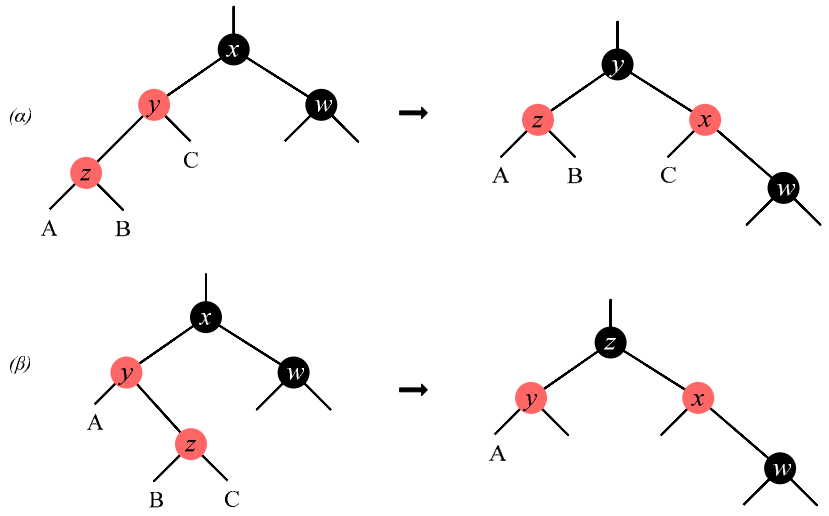
Έστω z ο χαμηλότερος και y ο ψηλότερος από τους δύο διαδοχικούς κόκκινους κόμβους. Μπορούμε να υποθέσουμε ότι το y δεν είναι η ρίζα του δέντρου, αφού αν ήταν θα μπορούσαμε εύκολα να διορθώσουμε την ιδιότητα (4) χρωματίζοντας τη ρίζα (δηλαδή το y) μαύρη. Έστω λοιπόν x ο πατέρας και w ο αδελφός του y (βλ. Σχήματα 2.9 και 2.10). Ο x είναι μαύρος γιατί πριν την εισαγωγή του νέου στοιχείου το δέντρο είχε όλες τις ιδιότητες των ΚΜ-δέντρων. Διακρίνουμε δύο περιπτώσεις ανάλογα με το χρώμα του w.



Σχήμα 2.9. Αν ο αδελφός του y είναι επίσης κόκκινος, η ιδιότητα (4) αποκαθίσταται τοπικά αντιστρέφοντας τα χρώματα των x, y και w. Η αντιστροφή χρωμάτων δεν επηρεάζει την ιδιότητα (5) και είναι μη-δομική επαναζυγιστική πράξη. Όμως, είναι μη-τερματική αφού ο πατέρας του x μπορεί να είναι επίσης κόκκινος.

Όταν ο αδερφός w του y είναι επίσης *κόκκινος* (Βλ. Σχήμα 2.9 για την περίπτωση που το y είναι αριστερό παιδί του x. Η περίπτωση που το y είναι δεξί παιδί του x είναι όμοια), η ιδιότητα (4) αποκαθίσταται αντιστρέφοντας τα χρώματα των x, y, και w. Έτσι ο κόμβος x χρωματίζεται κόκκινος, εκτός αν είναι η ρίζα του δέντρου, οπότε παραμένει μαύρος (για την περίπτωση που το x είναι ρίζα, βλ. Άσκηση 2.1). Οι κόμβοι y και w

χρωματίζονται μαύροι. Η ιδιότητα (5) δεν επηρεάζεται αφού ο αριθμός των μαύρων κόμβων σε όλα τα μονοπάτια από το x προς τα υποκείμενα φύλλα παραμένει ίδιος (πριν και μετά την αλλαγή χρωμάτων). Η πράξη αντιστροφής των χρωμάτων είναι *μη-τερματική* αφού ενδεχομένως μεταφέρει την παραβίαση της ιδιότητας (4) στον κόμβο x και στον πατέρα του. Η αντιστροφή χρωμάτων είναι *μη-δομική* επαναζυγιστική πράξη, αφού επηρεάζει μόνο τα χρώματα των κόμβων και όχι τη δομή του δέντρου.



Σχήμα 2.10. Αν ο αδελφός του y είναι μαύρος, η ιδιότητα (4) αποκαθίσταται με απλή περιστροφή (όταν οι ακμές (x, y) και (y, z) ακολουθούν την ίδια κατεύθυνση, (α)) ή διπλή περιστροφή (όταν οι ακμές (x, y) και (y, z) ακολουθούν αντίθετη κατεύθυνση, (β)). Οι περιστροφές δεν επηρεάζουν την ιδιότητα (5) και είναι τερματικές.

Όταν ο αδερφός w του y είναι *μαύρος* (Βλ. Σχήμα 2.10 για την περίπτωση που το y είναι αριστερό παιδί του x. Η περίπτωση που το y είναι δεξί παιδί του x είναι όμοια), η ιδιότητα (4) αποκαθίσταται με απλή ή διπλή περιστροφή στο υποδέντρο με ρίζα το x (δομικές επαναζυγιστικές πράξεις). Όταν οι ακμές (x, y) και (y, z) ακολουθούν την ίδια κατεύθυνση, εκτελείται *απλή* περιστροφή (Σχήμα 2.10.α). Όταν οι ακμές (x, y) και (y, z) ακολουθούν αντίθετη κατεύθυνση, εκτελείται *διπλή* περιστροφή (Σχήμα 2.10.β). Οι περιστροφές αποκαθιστούν την ιδιότητα (4). Επίσης, δεν μεταβάλλουν τον αριθμό των μαύρων κόμβων από τη ρίζα του υποδέντρου στα υποκείμενα φύλλα και συνεπώς δεν επηρεάζουν την ιδιότητα (5). Τέλος, οι περιστροφές είναι *τερματικές* αφού η ρίζα του

υποδέντρου παραμένει μαύρη και η παραβίαση της ιδιότητας (4) δεν διαδίδεται προς τη ρίζα.

Συνολικά ο χρόνος χειρότερης περίπτωσης για την εισαγωγή στοιχείου σε ένα ΚΜ-δέντρο είναι $O(\log n)$: λογαριθμικός χρόνος για την εύρεσης της θέσης που εισάγεται το στοιχείο και το πολύ λογαριθμικός αριθμός από αντιστροφές χρωμάτων (μη-δομικές επαναζυγιστικές πράξεις). Κάθε εισαγωγή μπορεί να προκαλέσει το πολύ 1 δομική επαναζυγιστική πράξη (απλή ή διπλή περιστροφή).

Σημείωση. Κατά την εισαγωγή ενός στοιχείου, θα μπορούσαμε να χρωματίσουμε *μαύρο* τον νέο κόμβο. Αυτό θα διατηρούσε όλες τις ιδιότητες εκτός από την (5). Η αποκατάσταση της ιδιότητας (5) θα ήταν εφικτή αλλά δυσκολότερη αφού η ιδιότητα αφορά το δέντρο συνολικά. Αντίθετα, η παραβίαση της ιδιότητας (4) αφορά «τοπικά» το σημείο στο οποίο εμφανίζονται δύο συνεχόμενοι κόκκινοι κόμβοι. Έτσι η αποκατάστασή της είναι ευκολότερη.

2.2.2. Διαγραφή σε ΚΜ-Δέντρο

Αρχικά διαγράφουμε τον αντίστοιχο κόμβο όπως στα (μη-ζυγισμένα) ΔΔΑ. Αυτό μπορεί να οδηγήσει σε παραβίαση της ιδιότητας (4) (όταν διαγράφεται ένας μαύρος κόμβος ανάμεσα σε δύο κόκκινους), της ιδιότητας (5) (όταν διαγράφεται ένας μαύρος κόμβος), και της ιδιότητας (2) (όταν διαγράφεται η ρίζα και ένας κόκκινος κόμβος παίρνει τη θέση της). Η ιδιότητα (2) αποκαθίσταται εύκολα χρωματίζοντας τη ρίζα μαύρη. Για τις ιδιότητες (4) και (5) πρέπει να εκτελέσουμε τις κατάλληλες επαναζυγιστικές πράξεις (μεταβολές χρωμάτων και περιστροφές).

Δεν θα εξηγήσουμε αναλυτικά τις επαναζυγιστικές πράξεις που μπορούν να λάβουν χώρα σε μία διαγραφή. Ο συνολικός χρόνος για τη διαγραφή ενός στοιχείου από ένα ΚΜ-δέντρο είναι $O(\log n)$ στη χειρότερη περίπτωση. Ο αριθμός των μη-δομικών επαναζυγιστικών πράξεων (αντιστροφές χρώματος) είναι $O(\log n)$, ενώ ο αριθμός των δομικών επαναζυγιστικών πράξεων (περιστροφές) είναι το πολύ 2, εκ των οποίων η μία μπορεί να είναι διπλή περιστροφή.

2.2.3. Παραδείγματα

Για συγκεκριμένα παραδείγματα εισαγωγής και διαγραφής στοιχείων σε Κόκκινα-Μαύρα δέντρα χρησιμοποιείστε το Java applet στη διεύθυνση (ενεργοποιείστε την επιλογή R-B στα δεξιά):

http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

Δοκιμάστε αρχικά τις ίδιες ακολουθίες με αυτές που δοκιμάσατε στα AVL δέντρα και συγκρίνετε τις δύο περιπτώσεις.

2.2.4. Ασκήσεις

Άσκηση 2.1. Relaxed KM-δέντρο είναι ένα δέντρο που έχει όλες τις ιδιότητες των Κόκκινων-Μαύρων δέντρων εκτός από την ιδιότητα (2). Δηλαδή, η ρίζα ενός relaxed KM-δέντρου μπορεί να είναι είτε μαύρη είτε κόκκινη. Έστω ένα relaxed KM-δέντρο με κόκκινη ρίζα. Χρωματίζουμε τη ρίζα του μαύρη. Το δέντρο που σχηματίζεται είναι KM-δέντρο ή όχι; Αν ναι να αποδείξετε τον ισχυρισμό σας, αλλιώς να δώσετε αντιπαράδειγμα.

Άσκηση 2.2. Θεωρείστε ένα KM-δέντρο που σχηματίζεται με διαδοχική εισαγωγή n διαφορετικών στοιχείων σε ένα αρχικά κενό δέντρο. Να αποδείξετε ότι για κάθε $n \geq 2$, το δέντρο θα περιέχει τουλάχιστον ένα κόκκινο κόμβο.

Άσκηση 2.3. Θεωρείστε το λόγο του αριθμού των κόκκινων στοιχείων προς τον αριθμό των μαύρων στοιχείων (δεν συμπεριλαμβάνονται τα NULL-φύλλα) σε ένα KM-δέντρο. Ποια είναι η μέγιστη και ποια η ελάχιστη τιμή του λόγου αυτού; Να δώσετε KM-δέντρα που να έχουν τις τιμές αυτές.

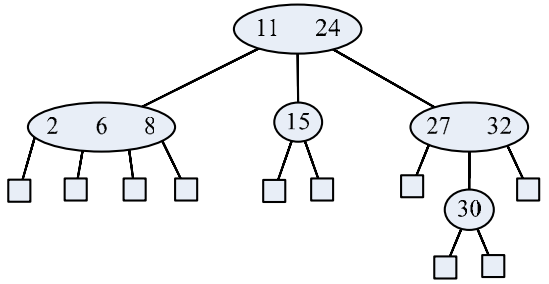
3. Δέντρα Αναζήτησης Πολλαπλής Διακλάδωσης

Τα *δέντρα αναζήτησης πολλαπλής διακλάδωσης* (multi-way search trees) αποτελούν τη φυσιολογική γενίκευση των ΔΔΑ. Ένα Δέντρο Αναζήτησης Πολλαπλής Διακλάδωσης (ΔΑΠΔ) είναι ένα δέντρο με ρίζα (rooted tree) του οποίου κάθε εσωτερικός κόμβος έχει *τουλάχιστον* δύο παιδιά. Μόνο οι εσωτερικοί κόμβοι περιέχουν στοιχεία. Όπως και στα ΔΔΑ, τα NULL-φύλλα αντιστοιχούν σε NULL δείκτες και δεν περιέχουν στοιχεία.

Κάθε εσωτερικός κόμβος x με k παιδιά, $x.c[1], \ldots, x.c[k]$, $k \geq 2$, έχει αποθηκευμένα $k-1$ στοιχεία με κλειδιά διατεταγμένα σε αύξουσα σειρά, $x.key[1] < x.key[2] < \ldots < x.key[k-1]$. Η χαρακτηριστική ιδιότητα των ΔΑΠΔ είναι:

- Τα κλειδιά των στοιχείων που είναι αποθηκευμένα στο υποδέντρο με ρίζα το $x.c[1]$ είναι μικρότερα ή ίσα $x.key[1]$.
- Για κάθε $i = 2, \ldots, k-1$, τα κλειδιά των στοιχείων που είναι αποθηκευμένα στο υποδέντρο με ρίζα το $x.c[i]$ είναι μεγαλύτερα ή ίσα του $x.key[i-1]$ και μικρότερα ή ίσα του $x.key[i]$.
- Τα κλειδιά των στοιχείων που είναι αποθηκευμένα στο υποδέντρο με ρίζα το $x.c[k]$ είναι μεγαλύτερα ή ίσα $x.key[k-1]$.

Με άλλα λόγια τα κλειδιά δύο διαδοχικών στοιχείων $x.key[i]$ και $x.key[i+1]$ ορίζουν το διάστημα στο οποίο πρέπει να ανήκουν όλα τα κλειδιά του υποδέντρου που έχει ρίζα το $x.c[i]$. Η ιδιότητα που χαρακτηρίζει τα ΔΑΠΔ αποτελεί γενίκευση της ιδιότητας των ΔΔΑ. Ένα απλό παράδειγμα ΔΑΠΔ φαίνεται στο Σχήμα 3.1.



Σχήμα 3.1. Παράδειγμα δέντρου αναζήτησης πολλαπλής διακλάδωσης.

Όσο μεγαλύτερος είναι ο αριθμός των παιδιών που μπορούν να έχουν οι εσωτερικοί κόμβοι ενός ΔΑΠΔ, τόσο μικρότερο είναι το ύψος του. Χρησιμοποιώντας δέντρα αναζήτησης με αρκετά μεγάλο βαθμό διακλάδωσης μπορούμε να αποθηκεύσουμε μεγάλο αριθμό στοιχείων σε δέντρα με πολύ μικρό ύψος. Για παράδειγμα, χρησιμοποιώντας βαθμό διακλάδωσης 10^3 μπορούμε να αποθηκεύσουμε 10^9 (ένα δισεκατομμύριο) στοιχεία σε ένα δέντρο ύψους μόλις 3 (χωρίς να λογίζονται τα NULL-φύλλα). Αυτό το πλεονέκτημα κάνει τα ΔΑΠΔ κατάλληλα για εφαρμογές όπου ο αριθμός των κόμβων που επισκεπτόμαστε κατά την εκτέλεση μιας λειτουργίας (εισαγωγής, διαγραφής, αναζήτησης, κλπ.) πρέπει να είναι πολύ μικρός. Αντιπροσωπευτικό παράδειγμα είναι οι εφαρμογές όπου μεγάλο μέρος των στοιχείων είναι αποθηκευμένο στο σκληρό δίσκο (π.χ. Συστήματα Διαχείρισης Βάσεων Δεδομένων, DBMSs).

Στις παραγράφους που ακολουθούν, δεδομένου ενός εσωτερικού κόμβου x, θα χρησιμοποιούμε τα $x.nc$, $x.key[i]$, και $x.c[i]$ για να συμβολίζουμε τον αριθμό των στοιχείων του x, το i-οστό στοιχείο του x ($1 \le i \le x.nc$), και το i-οστό παιδί του x ($1 \le i \le x.nc+1$).

3.1. Λειτουργίες Ερώτησης

3.1.1. Ενδο-Διατεταγμένη Διέλευση

Οι ορισμός της ενδο-διατεταγμένης (inorder) διέλευσης μπορεί εύκολα να γενικευθεί στα ΔΑΠΔ. Στην ενδο-διατεταγμένη διέλευση ενός υποδέντρου με ρίζα τον κόμβο x, ξεκινάμε από τα στοιχεία του υποδέντρου με ρίζα το πρώτο παιδί ($x.c[1]$), και συνεχίζουμε με το πρώτο στοιχείο ($x.key[1]$), τα στοιχεία του υποδέντρου με ρίζα το δεύτερο παιδί ($x.c[2]$), το δεύτερο στοιχείο ($x.key[2]$), κοκ. Η διέλευση ολοκληρώνεται με το τελευταίο στοιχείο ($x.key[x.nc]$) ακολουθούμενο από τα στοιχεία του υποδέντρου που έχει ρίζα το τελευταίο παιδί ($x.c[x.nc+1]$).

Ένα πιο συνοπτικός αλλά ισοδύναμος ορισμός είναι ότι η ενδο-διατεταγμένη διέλευση επισκέπτεται το στοιχείο $x.key[i]$ μεταξύ των στοιχείων του υποδέντρου με ρίζα το $x.c[i]$ και των στοιχείων του υποδέντρου με ρίζα το $x.c[i+1]$, i = 1, …, $x.nc$.

Όπως και στα ΔΔΑ, είναι εύκολο να αποδειχθεί ότι η ενδο-διατεταγμένη διέλευση τυπώνει τα στοιχεία ενός ΔΑΠΔ σε αύξουσα σειρά. Ο παρακάτω ψευδοκώδικας αποτελεί

μία αναδρομική υλοποίηση της ενδο-διατεταγμένη διέλευση ενός ΔΑΠΔ. Σαν παράδειγμα, δοκιμάστε να τον εφαρμόσετε στο δέντρο του Σχήματος 3.1.

```
Inorder-Traversal(x)
    if x = NULL then return;
    for i ← 1 to x.nc do
        Inorder-Traversal(x.c[i]);
        print(x.key[i]);
    Inorder-Traversal(x.c[x.nc+1]);
```

3.1.2. Αναζήτηση

Η αναζήτηση σε ένα ΔΑΠΔ αποτελεί επίσης γενίκευση της αναζήτηση σε ένα ΔΔΑ. Έστω ότι αναζητούμε το στοιχείο με κλειδί k και ότι η αναζήτησή μας έχει φτάσει στον κόμβο x. Σε αυτό το σημείο πρέπει να καθορίσουμε σε ποιο από τα $x.nc$+1 υποδέντρα του x θα προχωρήσει η αναζήτηση. Πρέπει δηλαδή να προχωρήσουμε σε μία απόφαση διακλάδωσης με $x.nc$+1 σκέλη (στα ΔΔΑ αναζήτησης είχαμε μόνο δύο σκέλη: αριστερά και δεξιά ανάλογα με το αν το στοιχείο είναι μικρότερο ή μεγαλύτερο από το $x.key$). Η απόφαση αυτή λαμβάνεται με βάση την ιδιότητα των ΔΑΠΔ.

Κατ' αρχήν, αν κάποιο από τα στοιχεία του x έχει κλειδί k, επιστρέφουμε έναν δείκτη στο αντίστοιχο στοιχείου του x. Διαφορετικά, αν το k είναι μικρότερο από το $x.key[1]$, συνεχίζουμε την αναζήτηση στο υποδέντρο με ρίζα το $x.c[1]$ (αφού γνωρίζουμε ότι σε αυτό το υποδέντρο βρίσκονται όλα τα στοιχεία με κλειδί μικρότερο του $x.key[1]$). Ομοίως, αν το k είναι μεγαλύτερο από το $x.key[x.nc]$, συνεχίζουμε την αναζήτηση στο υποδέντρο με ρίζα το $x.c[x.nc+1]$. Διαφορετικά, βρίσκουμε τον πρώτο (μικρότερο) δείκτη i, $2 \le i \le x.nc$, τέτοιο ώστε $x.key[i-1] < k < x.key[i]$, και συνεχίζουμε στο υποδέντρο με ρίζα το $x.c[i]$. Μπορούμε εύκολα να διακρίνουμε τις παραπάνω περιπτώσεις εκτελώντας είτε *γραμμική* είτε *δυαδική* αναζήτηση στον πίνακα $x.key$.

Για παράδειγμα, η (επιτυχημένη) αναζήτηση του 30 στο δέντρο του Σχήματος 3.1, ξεκινάει από τρίτο παιδί της ρίζας (αφού 30 > 24), συνεχίζει στο δεύτερο παιδί του κόμβου (27, 32) (αφού 27 < 30 < 32), και καταλήγει στο 30. Η (αποτυχημένη) αναζήτηση του 20, ξεκινάει από το δεύτερο παιδί της ρίζας (11 < 20 < 24), και συνεχίζει στο δεύτερο παιδί του 15 (15 < 20), το οποίο όμως είναι NULL-φύλλο.

Ο παρακάτω ψευδοκώδικας αποτελεί μία αναδρομική υλοποίηση της αναζήτησης στα ΔΑΠΔ. Η συγκεκριμένη εκδοχή χρησιμοποιεί *γραμμική* αναζήτηση για να καθορίσει

το υποδέντρο στο οποίο πρέπει να συνεχίσει. Σαν άσκηση, να γράψετε ψευδοκώδικα για την εκδοχή που χρησιμοποιεί *δυαδική* αντί για γραμμική αναζήτηση.

```
M-Tree-Search(x, k)

i ← 1;
while i ≤ x.nc and k > x.key[i] do
    i ← i + 1;
if i ≤ x.nc and k = x.key[i] then
    return(x, i);
if x.nc = 0 then return(NULL); /* Ο x είναι NULL-φύλλο */
else return(M-Tree-Search(x.c[i], k));
```

Άσκηση 3.1. Πως υπολογίζουμε το μέγιστο και το ελάχιστο στοιχείο ενός ΔΑΠΔ. Πώς υπολογίζουμε το προηγούμενο και το επόμενο (στην ενδο-διατεταγμένη διάταξη) ενός δεδομένου στοιχείου;

Υπόδειξη. Προσπαθήστε να γενικεύσετε τις αντίστοιχες λειτουργίες στα ΔΔΑ.

3.2. (α, β)-Δέντρα

Τα (α, β)-δέντρα αποτελούν τη ζυγισμένη εκδοχή των δέντρων αναζήτησης με πολλαπλή διακλάδωση. Έστω α, β ακέραιοι τέτοιοι ώστε α ≥ 2 και β ≥ 2α. Ένα ΔΑΠΔ καλείται *(α, β)-δέντρο* ((a,b)-tree) όταν:

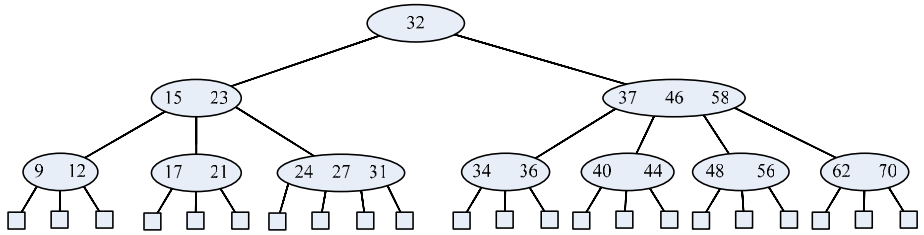
(1) Όλα τα φύλλα έχουν το ίδιο βάθος (δηλαδή το δέντρο είναι *πλήρως* ζυγισμένο).

(2) Η ρίζα του δέντρου έχει τουλάχιστον 2 και το πολύ β παιδιά.

(3) Κάθε εσωτερικός κόμβος εκτός από τη ρίζα έχει τουλάχιστον α και το πολύ β παιδιά.

Στην ειδική περίπτωση που α = 2 και β = 4, έχουμε τα (2, 4)-δέντρα. Ένα παράδειγμα (2, 4) δέντρου φαίνεται στο Σχήμα 3.2. Στην ειδική περίπτωση που $a=\lceil \beta/2 \rceil$ (κάθε κόμβος εκτός από τη ρίζα είναι μισό-γεμάτος – έχει πάντα τουλάχιστον τα μισά από το μέγιστο αριθμό παιδιών που μπορεί να έχει), το δέντρο ονομάζεται *B-δέντρο* (B-tree). Το δέντρο του Σχήματος 3.2 είναι B-δέντρο για β = 4 (όλοι οι εσωτερικοί κόμβοι εκτός από τη ρίζα έχουν τουλάχιστον 3 και το πολύ 4 παιδιά). Στην ειδική περίπτωση που $a=\lceil 2\beta/3 \rceil$ (κάθε κόμβος εκτός από τη ρίζα είναι γεμάτος σε ποσοστό τουλάχιστον 2/3), το δέντρο ονομάζεται B^**-δέντρο* (B*-tree). Το δέντρο του Σχήματος 3.2. είναι B^*-δέντρο

για β = 4. Οι διάφορες εκδόσεις των B-δέντρων βρίσκουν εφαρμογές σε όλα τα σύγχρονα Συστήματα Διαχείρισης Βάσεων Δεδομένων.



Σχήμα 3.2. Ένα παράδειγμα (2, 4)-δέντρου.

Άσκηση 3.2. Περιγράψτε τη δομή δεδομένων που προκύπτει αν κάθε μαύρος κόμβος σε ένα ΚΜ-δέντρο απορροφούσε τα κόκκινα παιδιά του (εφόσον είχε) και έκανε τα παιδιά τους δικά του παιδιά.

Υπόδειξη. Ποιος θα ήταν ο μέγιστος και ποιος ο ελάχιστος αριθμός παιδιών κάθε μαύρου κόμβου μετά από την απορρόφηση των κόκκινων παιδιών του; Ποιο θα ήταν το νέο βάθος των NULL-φύλλων;

3.2.1. Ύψος (α, β)-Δέντρων

Στη συνέχεια θα αποδείξουμε ότι το ύψος ενός (α, β)-δέντρου είναι λογαριθμικό στον αριθμό των στοιχείων του.

Λήμμα 3.1. Για κάθε (α, β)-δέντρο με ύψος h και n στοιχεία, ισχύει ότι

$$2\,\alpha^{h-1} - 1 \le n \le \beta^h - 1$$

Απόδειξη. Ο *ελάχιστος* αριθμός στοιχείων για δεδομένο ύψος h προκύπτει όταν η ρίζα έχει ένα στοιχείο (δύο παιδιά) και κάθε εσωτερικός κόμβος α – 1 στοιχεία (α παιδιά). Σε αυτή την περίπτωση, ο αριθμός των στοιχείων είναι:

$$1 + 2(\alpha - 1) + 2\alpha(\alpha - 1) + \ldots + 2\alpha^{h-2}(\alpha - 1) = 1 + 2(\alpha - 1)(\alpha^{h-1} - 1)/(\alpha - 1) = 2\,\alpha^{h-1} - 1$$

αφού η ρίζα έχει ένα στοιχείο, υπάρχουν 2 κόμβοι σε βάθος 1, 2α κόμβοι σε βάθος 2, κοκ., $2\alpha^{h-2}$ κόμβοι σε βάθος h–1, και ο κάθε εσωτερικός κόμβος (εκτός της ρίζας) έχει (α – 1) στοιχεία.

Ο *μέγιστος* αριθμός στοιχείων για δεδομένο ύψος h προκύπτει όταν όλοι οι εσωτερικοί κόμβοι έχουν β – 1 στοιχεία (β παιδιά). Σε αυτή την περίπτωση, ο αριθμός των στοιχείων είναι:

$$(\beta-1)+\beta(\beta-1)+\beta^2(\beta-1)+\ldots+\beta^{h-1}(\beta-1)=(\beta-1)(\beta^h-1)/(\beta-1)=\beta^h-1$$

αφού υπάρχει ένας κόμβος σε βάθος 0 (η ρίζα), β κόμβοι σε βάθος 1, β^2 κόμβοι σε βάθος 2, κοκ, β^{h-1} κόμβοι σε βάθος h, και κάθε εσωτερικός κόμβος έχει (β – 1) στοιχεία. ■

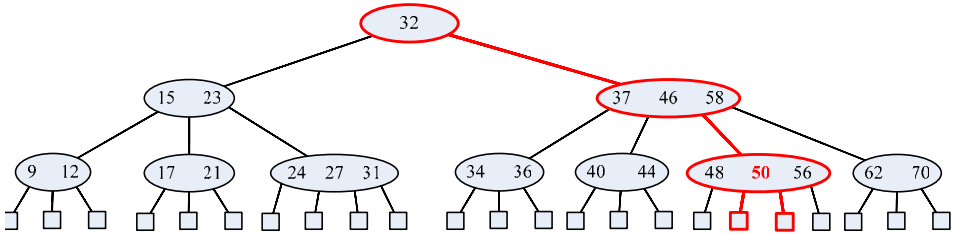
Από το Λήμμα 3.1, το ύψος h ενός (α, β)-δέντρου με n στοιχεία είναι Θ(log n) και συγκεκριμένα ικανοποιεί τις παρακάτω ανισότητες:

$$\log_\beta(n+1)\le h\le\log_\alpha\left(\frac{n+1}{2}\right)+1$$

Όπως και στα ΔΔΑ, ο χρόνος εκτέλεσης χειρότερης περίπτωσης της αναζήτησης στα (α, β)-δέντρα είναι γραμμικός στο ύψος του δέντρου (βλ. και παράγραφο 3.1.2). Στα (α, β)-δέντρα ο χρόνος αναζήτησης εξαρτάται και από τις τιμές των α και β εξαιτίας της αναζήτησης του κατάλληλου παιδιού σε κάθε κόμβο και της επίδρασης που έχουν αυτές στο ύψος του δέντρου.

Στην εκδοχή που χρησιμοποιεί γραμμική αναζήτηση, ο χρόνος εκτέλεσης χειρότερης περίπτωσης είναι O((β / log α)log n). Χρησιμοποιώντας δυαδική αναζήτηση μπορούμε να πετύχουμε χρόνο αναζήτησης χειρότερης περίπτωσης O((logβ/logα)log$n$).

3.2.2. Εισαγωγή Στοιχείου

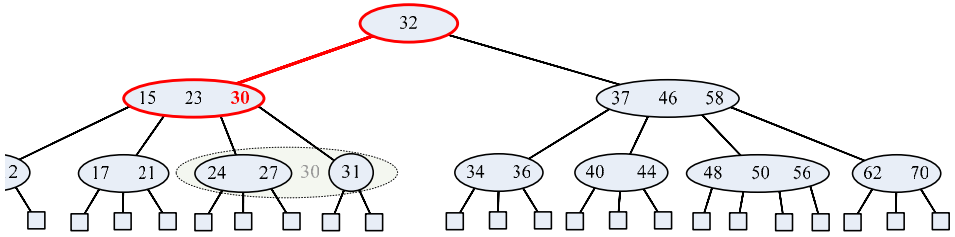


Σχήμα 3.3. Το (2, 4)-δέντρο του σχήματος 3.2 μετά την εισαγωγή του 50.

Εισάγουμε κάθε νέο στοιχείο στον πατέρα του NULL-φύλλου όπου κατέληξε η (αποτυχημένη) διαδικασία αναζήτησης του στοιχείου. Το νέο στοιχείο παίρνει (στον

εσωτερικό κόμβο-πατέρα) τη θέση που αντιστοιχεί στο συγκεκριμένο NULL-φύλλο (Σχήμα 3.3). Παρατηρείστε ότι συγκεκριμένη διαδικασία δεν μεταβάλει το ύψος του δέντρου (δηλ. η ιδιότητα (1) συνεχίζει να ισχύει και μετά την εισαγωγή).

Μια εισαγωγή στοιχείου μπορεί όμως να οδηγήσει στην παραβίαση των ιδιοτήτων (2) και (3). Συγκεκριμένα, κάποιος κόμβος μπορεί να αποκτήσει β+1 παιδιά (β στοιχεία), να παρουσιάσει όπως λέμε *υπερχείλιση* (overflow). Σε αυτή την περίπτωση, ο ανώτατος αριθμός των παιδιών αποκαθίσταται με τη *δομική επαναζυγιστική πράξη* της *διάσπασης* (split) του κόμβου στον οποίο εμφανίστηκε το πρόβλημα. Ο κόμβος που παρουσίασε υπερχείλιση διασπάται σε δύο και το στοιχείο $\lfloor(\beta+1)/2\rfloor$ («μεσαίο» στοιχείο) «ανεβαίνει» στον πατέρα ως διαχωριστικό στοιχείο των νέων κόμβων. Τα παιδιά από το πρώτο μέχρι το $\lfloor(\beta+1)/2\rfloor$ μαζί με τα στοιχεία στα αριστερά του $\lfloor(\beta+1)/2\rfloor$ στοιχείου συγκροτούν τον «αριστερό» νέο κόμβο. Τα παιδιά από το $\lfloor(\beta+1)/2\rfloor+1$ μέχρι το β+1 μαζί με τα στοιχεία στα δεξιά του $\lfloor(\beta+1)/2\rfloor$ στοιχείου συγκροτούν τον «δεξιό» νέο κόμβο (Σχήμα 3.4).



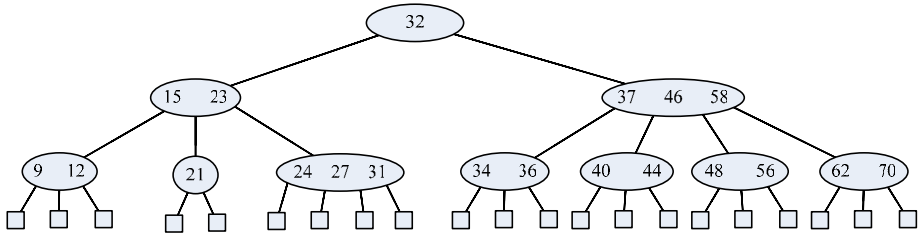
Σχήμα 3.4. Το (2, 4)-δέντρο του σχήματος 3.3 μετά την εισαγωγή του 30. Ο κόμβος (24, 27, 30, 31) παρουσιάζει υπερχείλιση και διασπάται. Ο «αριστερός» νέος κόμβος είναι ο (24, 27) και ο «δεξιός» νέος κόμβος είναι ο (31). Το 30 ανεβαίνει στον (15, 23) για να διαχωρίσει τους δύο νέους κόμβους. Έτσι δημιουργείται ο κόμβος (15, 23, 30).

Μια διάσπαση δεν είναι πάντα τερματική αφού μπορεί να προκαλέσει υπερχείλιση στον πατέρα του κόμβου που διασπάστηκε. Σε αυτή την περίπτωση προχωρούμε σε διάσπαση του πατέρα και η διαταραχή μπορεί να φτάσει μέχρι τη ρίζα. Σε περίπτωση υπερχείλισης της ρίζας, δημιουργείται μία νέα ρίζα με παιδιά τους δύο κόμβους που προέκυψαν από τη διάσπαση. Το ύψους του δέντρου αυξάνεται κατά 1. Παρατηρείστε ότι η επαναζυγιστική πράξη της διάσπασης δεν επηρεάζει σε καμία περίπτωση την ιδιότητα (1).

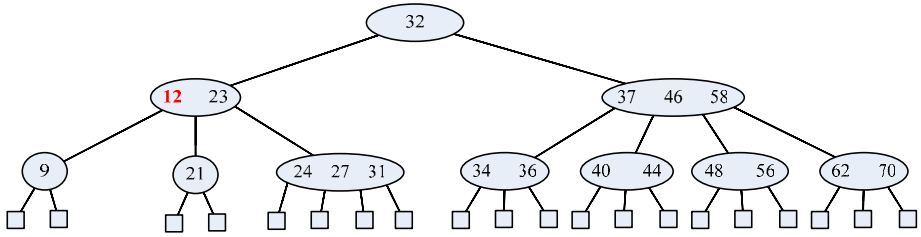
Ο χρόνος εκτέλεσης χειρότερης περίπτωσης για τη λειτουργία της εισαγωγής στα (α, β)-δέντρα είναι O(log *n*) και μπορεί να χρειαστούν O(log *n*) διασπάσεις (μία το πολύ σε κάθε επίπεδο του μονοπατιού από το σημείο εισαγωγής προς τη ρίζα).

3.2.3. Διαγραφή Στοιχείου

Αρχικά βρίσκουμε και διαγράφουμε το στοιχείο από τον αντίστοιχο κόμβο. Αν ο κόμβος που περιέχει το στοιχείο έχει NULL-φύλλα σαν παιδιά, τότε σβήνουμε και το αντίστοιχο NULL-φύλλο (Σχήμα 3.5). Διαφορετικά, το προηγούμενο (ή ισοδύναμα το επόμενο) στοιχείο στην ενδο-διατεταγμένη διέλευση παίρνει τη θέση του στοιχείου που διαγράφηκε (Σχήμα 3.6).



Σχήμα 3.5. Διαγραφή του στοιχείου 17 από το (2, 4)-δέντρο του Σχήματος 3.2. Ένα NULL-φύλλο διαγράφεται από τον κόμβο όπου βρισκόταν το 56.



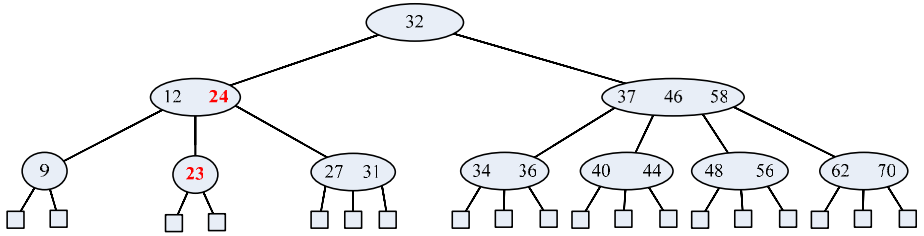
Σχήμα 3.6. Διαγραφή του στοιχείου 15 από το (2, 4)-δέντρο του Σχήματος 3.5. Το 12 (το προηγούμενο του 15 στην ενδο-διατεταγμένη διέλευση) παίρνει τη θέση του 15 και ένα NULL-φύλλο διαγράφεται από τον κόμβο όπου βρισκόταν το 12.

Στο εξής λοιπόν μπορούμε να υποθέτουμε ότι η διαγραφή συμβαίνει πάντα σε κόμβο του τελευταίου επιπέδου (του οποίου τα παιδιά είναι NULL-φύλλα), αφού ακόμη και όταν ένα στοιχείο διαγράφεται από κόμβο υψηλότερου επιπέδου, ένα στοιχείο από

κόμβο του τελευταίου επιπέδου παίρνει τη θέση του. Με αυτό τον τρόπο, είναι πάντα ένας κόμβος του τελευταίου επιπέδου αυτός του οποίου τα στοιχεία μειώνονται κατά 1. Επιπλέον, η δομή και το ύψος του δέντρου (ιδιότητα (1)) δεν επηρεάζονται κατ' αρχήν από τη διαγραφή ενός στοιχείου.

Όμως η διαγραφή ενός στοιχείου μπορεί να προκαλέσει *έλλειμμα* (underflow) στοιχείων σε έναν κόμβο του τελευταίου επιπέδου μειώνοντας τα στοιχεία του σε α – 2 (αντίστοιχα, μειώνοντας τα παιδιά του σε α – 1). Το έλλειμμα αντιμετωπίζεται με τις δομικές επαναζυγιστικές πράξεις του *δανεισμού* (sharing ή *μεταφοράς* – tranfer) και της συνένωσης ή *συγχώνευσης* (fusion ή merging).

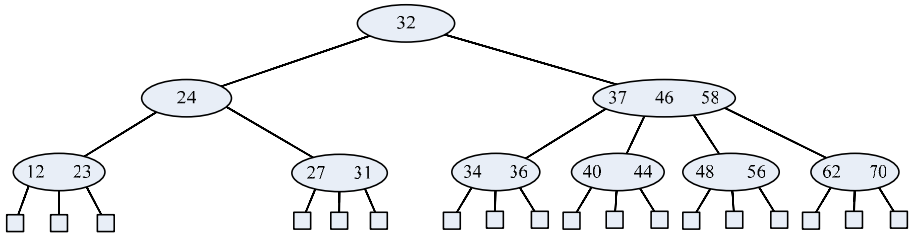
Αν είτε ο δεξιός είτε ο αριστερός γειτονικός κόμβος-αδελφός του ελλειμματικού κόμβου έχει τουλάχιστον α στοιχεία (α+1 παιδιά), τότε το ακραίο στοιχείο προς την πλευρά του ελλειμματικού κόμβου γίνεται το διαχωριστικό τους στοιχείο στον κοινό κόμβο-πατέρα τους και το προηγούμενο διαχωριστικό στοιχείο παίρνει τη θέση του ακραίου στοιχείου στον ελλειμματικό κόμβο (Σχήμα 3.7). Ο αριθμός των στοιχείων αποκαθίσταται στον ελλειμματικό κόμβο χωρίς να δημιουργηθεί πρόβλημα σε κάποιον άλλο κόμβο. Ο δανεισμός είναι λοιπόν μία *τερματική* επαναζυγιστική πράξη.



Σχήμα 3.7. Η διαγραφή του στοιχείου 21 από το (2, 4)-δέντρο του Σχήματος 3.6 καθιστά τον κόμβο που περιείχε το 21 ελλειμματικό (κανένα στοιχείο). Ο γειτονικός του αδελφός στα δεξιά (κόμβος (24, 27, 31) στο Σχήμα 3.6) έχει 3 στοιχεία και έχουμε δανεισμό. Το 23 παίρνει τη θέση του 21 και το 24 παίρνει τη θέση του 23 στον κόμβο-πατέρα τους.

Αν όλοι οι γειτονικοί κόμβοι-αδελφοί του ελλειμματικού κόμβου έχουν α – 1 στοιχεία (α παιδιά), τότε ο ελλειμματικός κόμβος συγχωνεύεται με ένα από τους γειτονικούς κόμβους-αδελφούς του. Το διαχωριστικό τους στοιχείο παύει να χρειάζεται και «κατεβαίνει» από τον κοινό κόμβο-πατέρα τους στον κόμβο που προκύπτει από τη συγχώνευση (Σχήμα 3.8). Έτσι, ο αριθμός των στοιχείων του κόμβου που προκύπτει από

τη συγχώνευση είναι (α – 2) + 1 + (α – 1) = 2(α – 1) ≤ β – 2 (επειδή β ≥ 2α). Η συγχώνευση μειώνει τον αριθμό των στοιχείων του κόμβου-πατέρα κατά 1 και ενδέχεται να μην είναι *τερματική* (μπορεί να δημιουργήσει έλλειμμα στον κόμβο-πατέρα).



Σχήμα 3.8. Η διαγραφή του στοιχείου 9 από το (2, 4)-δέντρο του Σχήματος 3.7 καθιστά τον κόμβο που περιείχε το 9 ελλειμματικό (κανένα στοιχείο). Ο μοναδικός γειτονικός του αδελφός (κόμβος δεξιός αδελφός (23) στο Σχήμα 3.7) έχει 1 στοιχείο οπότε έχουμε συγχώνευση των δύο κόμβων. Το διαχωριστικό τους στοιχείο (12) «κατεβαίνει» στον νέο κόμβο που είναι ο (12, 23). Η συγχώνευση μειώνει τον αριθμό των στοιχείων του κόμβου-πατέρα κατά 1 και ενδέχεται να μην είναι τερματική.

Μια ακολουθία διαδοχικών συγχωνεύσεων που λαμβάνει χώρα στο μονοπάτι από το σημείο διαγραφής στο τελευταίο επίπεδο προς τη ρίζα ενδέχεται να καταλήξει καθιστώντας τη ρίζα ελλειμματική. Υπενθυμίζουμε ότι η ρίζα καθίσταται ελλειμματική όταν απομείνει χωρίς κανένα στοιχείο (ή με ένα μόνο παιδί). Τότε το μοναδικό της παιδί παίρνει τη θέση της ρίζας και το ύψος του δέντρου μειώνεται κατά 1.

Ο χρόνος εκτέλεσης χειρότερης περίπτωσης για τη λειτουργία της διαγραφής στα (α, β)-δέντρα είναι O(log *n*) και μπορεί να χρειαστούν O(log *n*) συγχωνεύσεις (μία το πολύ σε κάθε επίπεδο του μονοπατιού από το σημείο εισαγωγής προς τη ρίζα) και το πολύ 1 δανεισμός.

Σημείωση. Τόσο η εισαγωγή όσο και η διαγραφή σε ένα (α, β)-δέντρο μπορούν να υλοποιηθούν σε *ένα πέρασμα*. Σε αυτή την περίπτωση, οι επαναζυγιστικές πράξεις λαμβάνουν χώρα ταυτόχρονα με την αναζήτηση του σημείου όπου πρέπει να εισαχθεί το νέο στοιχείο (κάθε κόμβος στο μονοπάτι από τη ρίζα προς το σημείο εισαγωγής με β – 1 παιδιά διασπάται «προληπτικά») ή την αναζήτηση του κόμβου που θα διαγραφεί (κάθε κόμβος στο μονοπάτι από τη ρίζα προς το σημείο εισαγωγής με α παιδιά συγχωνεύεται ή δανείζεται παιδί από τον κόμβο-αδελφό του «προληπτικά»). Έτσι εξασφαλίζεται ότι η

εισαγωγή ή η διαγραφή δεν θα προκαλέσουν άλλες επαναζυγιστικές πράξεις. Αυτό το χαρακτηριστικό είναι ιδιαίτερα σημαντικό όταν οι κόμβοι του δέντρου είναι αποθηκευμένοι στο σκληρό δίσκο.

4. Βιβλιογραφία

Τα δέντρα αναζήτησης είναι μια ευρύτατα χρησιμοποιούμενη δομή δεδομένων. Υπάρχουν πολλά βιβλία στη διεθνή βιβλιογραφία και αρκετά στην ελληνική βιβλιογραφία που καλύπτουν σε επαρκή έκταση και βάθος όλα τα θέματα που θίχτηκαν σε αυτές τις εκπαιδευτικές σημειώσεις. Ενδεικτικά αναφέρουμε:

[1] T.H. Cormen. C.E. Leiserson, R.L. Rivest, and C. Stein. *An Introduction to Algorithms, 2nd Edition*. The MIT Press, 2003.

[2] M.T. Goodrich, R. Tamassia, and D.M. Mount. *Data Structures and Algorithms in C++*. Wiley.

[3] S. Sahni. *Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++*. Μετάφραση Ι. Μανωλόπουλος και Ι. Θεοδωρίδης, Εκδόσεις Τζιόλα, 2004.

[4] Γ.Φ. Γεωργακόπουλος. *Δομές δεδομένων. Έννοιες, τεχνικές και αλγόριθμοι*. Πανεπιστημιακές Εκδόσεις Κρήτης.

[5] Π. Μποζάνης. *Δομές Δεδομένων: Ταξινόμηση και Αναζήτηση με Java*. Εκδόσεις Τζιόλα, 2003.