ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΑΛΑΜΑΤΑΣ ΠΑΡΑΡΤΗΜΑ ΣΠΑΡΤΗΣ
ΤΜΗΜΑ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Τα δυαδικά δένδρα αναζήτησης και οι εφαρμογές τους»

ΔΕΛΗ ΜΑΡΙΑ
2005011

-2012-

Επιβλέπων Καθηγητής: κ. Γρηγόρης Καραγιώργος

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου, κ. Γρηγόρη Καραγιώργο, για την βοήθεια που μου προσέφερε σε όλες τις φάσεις της συγγραφής αυτής της πτυχιακής εργασίας και για τις πολύτιμες συμβουλές του πάνω σε ερωτήματα που προέκυπταν κατά τη διάρκεια της εκπόνησης της.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τους γονείς μου, Πάρη και Νίκη, για τη στήριξη, που μου έδειξαν όλα αυτά τα χρόνια. Χωρίς την βοήθεια τους, την καθοδήγησή τους, αλλά κυρίως την αγάπη τους δε θα κατάφερνα να ολοκληρώσω τις σπουδές μου και να πραγματοποιήσω τα όνειρά μου. Τους οφείλω πολλά και τους υπερευχαριστώ.

Δελή Μαρία

Περίληψη

Στην παρούσα πτυχιακή εργασία έγινε μια μελέτη δομής δεδομένων, των *δυαδικών δένδρων αναζήτησης*. Αφού παρουσιάστηκε μια θεωρητική προσέγγιση των δένδρων, έγινε μελέτη διαφόρων πράξεών τους (αναζήτηση, εισαγωγή, διαγραφή κόμβου κα.), πραγματοποιήθηκε η υλοποίηση τους σε γλώσσα προγραμματισμού C, έγινε μελέτη τις πολυπλοκότητας τους και στη συνέχεια αναφορά στα διάφορα είδη των δομών αυτών που υπάρχουν. Τέλος, δόθηκε έμφαση στις διάφορες εφαρμογές των δυαδικών δένδρων αναζήτησης.

Η διάρθρωση της ύλης έγινε ως εξής:

Το *Κεφάλαιο 1* αποτελεί μια εισαγωγή στα δέντρα, τα δυαδικά δένδρα και τα δυαδικά δένδρα αναζήτησης. Οι γενικοί κανόνες τους είναι χρήσιμοι για τα επόμενα κεφάλαια. Τέλος στο κεφάλαιο αυτό παρουσιάζονται όλες οι μέθοδοι διάσχισης δένδρων.

Το κεφάλαιο 2 αποτελείται από την ανάλυση των δυαδικών δένδρων αναζήτησης και περιλαμβάνει, εκτός από τον ορισμό τους, την διαδικασία κατασκευής ενός δυαδικού δένδρου αναζήτησης καθώς επίσης και την παρουσίαση των διαφόρων ειδών δυαδικών δένδρων που υπάρχουν.

Στο *Κεφάλαιο 3* γίνεται παρουσίαση των λειτουργιών-πράξεων των δυαδικών δένδρων αναζήτησης. Γίνεται αναλυτική περιγραφή εξής λειτουργιών :

1. Λειτουργίες Ερώτησης ,που περιλαμβάνουν τις πράξεις αναζήτηση στοιχείου, την εύρεση μέγιστου και ελάχιστου στοιχείου , και

2. Λειτουργίες Ενημέρωσης, που περιλαμβάνουν τις πράξεις Εύρεση και Διαγραφή Στοιχείου.

Πραγματοποιήθηκε η υλοποίηση τους σε γλώσσα C.

Στο *Κεφάλαιο 4* γίνεται μελέτη και ανάλυση όλων των εφαρμογών των δυαδικών δένδρων αναζήτησης. Πιο συγκεκριμένα μελετήθηκαν οι εφαρμογές:

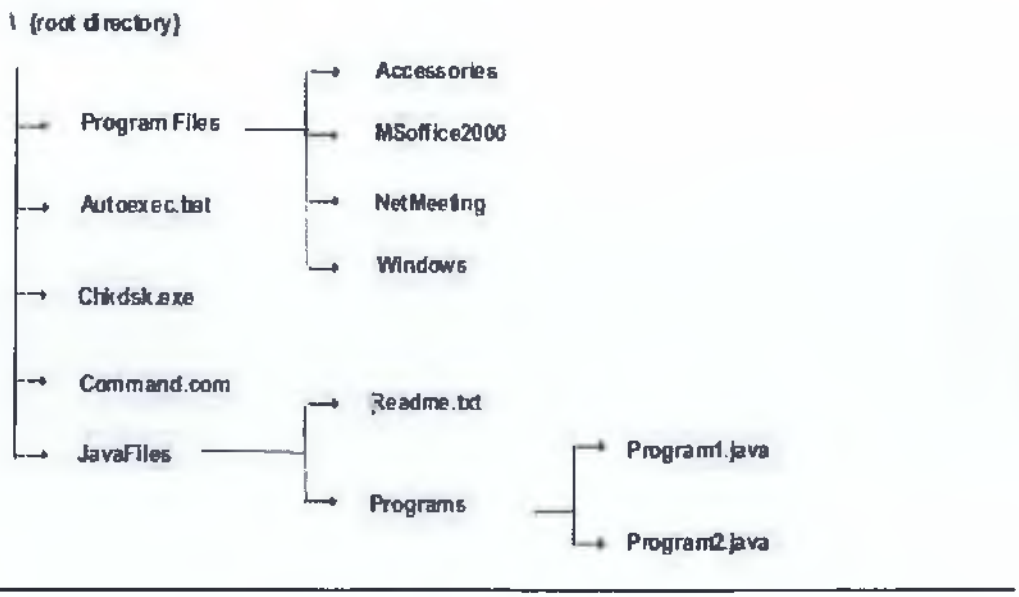
1. Ουρές προτεραιότητας,
2. Σωροί,
3. Δομή Λεξικού,
4. Κώδικες Huffman.

Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1^{Ο} : Εισαγωγή

1.1. Τα δένδρα

Στην παράγραφο αυτή ασχολούμαστε με τις μη-γραμμικές δομές δεδομένων ή δομές δεδομένων μιας διάστασης, που ονομάζονται δένδρα. Συναντάμε τα δένδρα σε πάρα πολλές εφαρμογές της επιστήμης των υπολογιστών. Κλασικό παράδειγμα η οργάνωση του συστήματος των αρχείων, με τη μορφή δένδρου, που ακολουθούν τα περισσότερα λειτουργικά συστήματα (π.χ.UNIX, DOS σχήμα 1.1.1). Τα δένδρα χρησιμοποιούνται επίσης και σαν μοντέλα αναπαράστασης προβλημάτων της καθημερινής μας ζωής. Το «γενεαλογικό δένδρο» (σχήμα 1.1.2.) και η δενδροειδής αναπαράσταση αγώνων κυπέλου μεταξύ ομάδων, αποτελούν χαρακτηριστικά παραδείγματα.



Σχήμα 1.1.1: Δενδροειδής αναπαράσταση αρχείων στα Windows
(Διαφάνεια του 6^{ου} εργαστηρίου του κ. Δ. Φωτάκη)

Αν και η έννοια του δένδρου γίνεται κατανοητή από οποιονδήποτε οπτικό τρόπο αναπαράστασής του, αυτή η δομή δεδομένων πρέπει να οριστεί με αυστηρά μαθηματικό τρόπο. Παρακάτω παρουσιάζονται δύο ισοδύναμοι ορισμοί για το δένδρο. Εξετάζουμε τους δύο αυτούς ορισμούς χρησιμοποιώντας για παράδειγμα το δένδρο σχήματος 1.1.3.

Ορισμός 1.1.1: Δέντρο (tree) είναι ένα σύνολο Τ από κόμβους (nodes), τέτοιο ώστε:

- Το Τ είναι κενό ή

• Το Τ περιλαμβάνει ένα ξεχωριστό κόμβο R, που ονομάζεται ρίζα (root) του Τ και οι υπόλοιποι κόμβοι Τ - {R} χωρίζονται σε μηδέν ή περισσότερα σύνολα κόμβων, T_1, T_2, . . ., Τn, που είναι ξένα μεταξύ τους και τα οποία είναι με τη σειρά τους δένδρα. Τα δένδρα T_1, T_2, . . ., Τn, ονομάζονται υπο-δένδρα του Τ.

Σχήμα 1.1.2: Ένα γενεαλογικό δένδρο

Σύμφωνα με τον πρώτο ορισμό το δένδρο του σχήματος αποτελείται από ένα σύνολο 13 κόμβων Τ = {Α, Β, Γ, Δ, Ε, Ζ, Η, Θ, Ι, Κ, Λ, Μ, Ν}, όπου Α είναι η ρίζα του δένδρου και το σύνολο Τ – {Α} χωρίζεται σε τρία υπο-δένδρα T_1, T_2 και T_3 όπου:

$$T_1 = \{B, E, Z, H\}, T_2 = \{Γ, Θ, I\} \text{ και } T_3 = \{Δ, K, Λ, M, N\}$$

Τα δένδρα T_1, T_2 και T_3 έχουν σαν ρίζα τον κόμβο Β , Γ, και Δ αντίστοιχα. Τα σύνολα T_1–{Β}, T_2–{Γ}
και T_3 – {Δ} χωρίζονται αντίστοιχα στα υπο-δέντρα:

$T_{1.1} = \{E\}$, $T_{1.2} = \{Z\}$, $T_{1.3} = \{H\}$

$T_{2.1} = \{Θ\}$, $T_{2.2} = \{I\}$

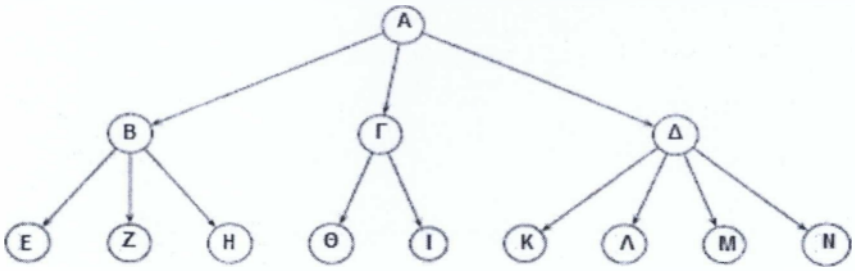
$T_{3.1} = \{K\}$, $T_{3.2} = \{Λ\}$, $T_{3.3} = \{M\}$, $T_{3.4} = \{N\}$

Κάθε ένα από τα τελευταία υπο-δένδρα είναι σύνολα που περιέχουν μόνο έναν κόμβο. Ο κόμβος αυτός αποτελεί την ρίζα του αντίστοιχου υπο-δένδρου και αν αφαιρεθεί προκύπτει το κενό.

Ορισμός 1.1.2: Δένδρο είναι μια συλλογή στοιχείων, τα οποία ονομάζονται κόμβοι. Οι κόμβοι του δένδρου συνδέονται μεταξύ τους με τη βοήθεια ακμών τηρώντας τους παρακάτω κανόνες:

- Υπάρχει μόνο ένας κόμβος στον οποίο δεν καταλήγει καμία ακμή, η ρίζα (root).

- Σε όλους τους υπόλοιπους κόμβους καταλήγει υποχρεωτικά το πολύ μία ακμή ακμή.



Σχήμα 1.1.3: Συνήθης οπτική αναπαράσταση δένδρου

Σύμφωνα με τον δεύτερο ορισμό παρατηρούμε ότι μόνο στον κόμβο Α, δηλαδή τη ρίζα του δένδρου δεν καταλήγει καμία ακμή, ενώ σε όλους τους άλλους κόμβους καταλήγει μόνο μία ακμή. Ο πρώτος ορισμός είναι αναδρομικός και πρέπει να τον έχουμε υπ' όψη μας για τις διάφορες ιδιότητες και τους αλγόριθμους επεξεργασίας δένδρων, ενώ ο δεύτερος ορισμός βρίσκεται σε αντιστοιχίζεται περισσότερο στην οπτική αναπαράσταση ενός δένδρου. Συνήθως σχεδιάζουμε ένα δένδρο με τη ρίζα του επάνω και τους υπόλοιπους κόμβους από κάτω. Μερικές ακόμη ιδιότητες των δένδρων που είναι απαραίτητες:

Κάθε κόμβος εκτός από τη ρίζα έχει ακριβώς ένα κόμβο από πάνω του, ο οποίος ονομάζεται πατέρας (father). Παραδείγματος χάριν στο παραπάνω δένδρο, ο Α είναι ο πατέρας του Δ και ο Γ είναι ο πατέρας του Ι. Οι κόμβοι που βρίσκονται κάτω από έναν κόμβο και συνδέονται με ακμές που ξεκινούν από αυτόν, είναι τα παιδιά του (children). Σε κάποιες περιπτώσεις, όπως για τα γενεαλογικά δένδρα, μπορεί να είναι χρήσιμο να αναφερθούμε στους απογόνους (successors) ή προγόνους (ancestors) ενός κόμβου. Στο παράδειγμα του παραπάνω σχήματος οι κόμβοι Θ και Ι είναι παιδιά του κόμβου Γ. Οι κόμβοι Θ και Ι είναι δύο από τους απόγονους του Α και ο Ι έχει πρόγονο τον Α. Ένας κόμβος ο οποίος δεν έχει κανένα υπο-δένδρο δηλαδή είναι ένας κόμβος χωρίς παιδιά ονομάζεται τερματικός (terminal) κόμβος, εξωτερικός (external) κόμβος ή φύλλο (leaf) του δένδρου. Στο δένδρο του σχήματος 1.1.3. τερματικοί είναι οι κόμβοι Ε, Ζ, Η, Θ, Ι, Κ, Λ, Μ και Ν. Οι υπόλοιποι κόμβοι του δένδρου ονομάζονται μη-τερματικοί(non-terminals) ή εσωτερικοί (internals). Μια ακολουθία κόμβων, ενός δένδρου, οι οποίοι συνδέονται διαδοχικά μεταξύ τους με τη βοήθεια ακμών, ονομάζεται μονοπάτι (path). Οι ακολουθίες < Α, Γ, Ι > και <Α, Δ, Κ> είναι δύο από τα μονοπάτια του παραπάνω σχήματος. Ακόμη, οι κόμβοι ενός δένδρου χωρίζονται σε επίπεδα (levels). Ένας κόμβος βρίσκεται στο επίπεδο n, εάν n είναι ο αριθμός των ακμών του μονοπατιού, που συνδέει τη ρίζα με τον κόμβο αυτό. Στο παράδειγμα του παραπάνω σχήματος ο Α ανήκει στο επίπεδο 0. Οι Β, Γ, Δ ανήκουν στο επίπεδο 1, ενώ οι

κόμβοι Ε, Ζ, Η, Θ, Ι, Κ, Λ, Μ, Ν αποτελούν το επίπεδο 2. Ορίζουμε σαν ύψος (height) ή βάθος (depth) ενός κόμβου τον αριθμό του επιπέδου στο οποίο βρίσκεται ο κόμβος αυτός. Το ύψος (ή βάθος) ενός δένδρου ταυτίζεται με το μέγιστο ύψος (ή βάθος) των κόμβων του δένδρου. Ονομάζουμε βαθμό (degree) ενός κόμβου τον αριθμό των υπο-δένδρων του. Ονομάζουμε δάσος (forest), ένα σύνολο από n (όπου $n \geq 0$) δένδρα, που είναι ξένα μεταξύ τους.

Ειδικές κατηγορίες δένδρων αποτελούν τα δυαδικά δένδρα, τα τριαδικά δένδρα κλπ.

1.2 Δυαδικά δένδρα

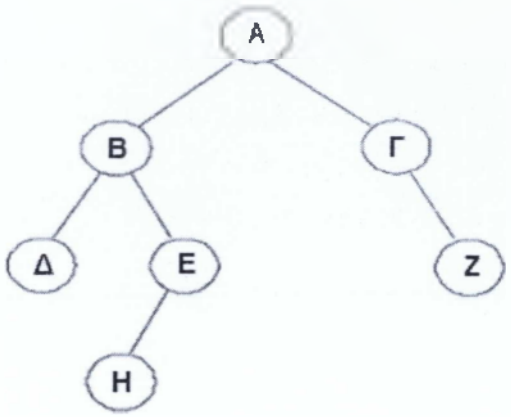
Τα δυαδικά δένδρα είναι μια εισαγωγική δομή δεδομένων στην Πληροφορική που χρησιμοποιούνται με σκοπό την γρήγορη αποθήκευση και ανάκτηση δεδομένων. Σύμφωνα με τον ορισμό των δένδρων, τα δυαδικά δένδρα είναι αναδρομικές δομές οι οποίες αποτελούνται από ένα πεπερασμένο σύνολο κόμβων : διαθέτουν ένα ριζικό κόμβο και δύο υποσύνολα κόμβων ξένα μεταξύ τους τα οποία είναι και αυτά με τη σειρά τους δένδρα.

Εάν διαχωρίσουμε τα δύο υπο-δένδρα σε αριστερό και δεξί για κάθε κόμβο, έχουμε ένα *διατεταγμένο δένδρο*. Τα παιδιά κάθε κόμβου θα αναγνωρίζονται ως δεξί και αριστερό παιδί. Βλέπετε την απεικόνιση ενός δυαδικού δένδρου στο σχήμα 1.2.1. Σε ένα δυαδικό δένδρο (όπως αναλύεται και παρακάτω) ο βαθμός κάθε κόμβου μπορεί να είναι το πολύ 2. Εάν όλοι οι κόμβοι έχουν από δύο παιδιά εκτός αυτών του τελευταίου επιπέδου, το δυαδικό δέντρο ονομάζεται *πλήρες*. Όταν ο κάθε κόμβος ενός δυαδικού δένδρου έχει τον ίδιο αριθμό παιδιών αριστερά και δεξιά του, τότε είναι τέλεια ισορροπημένο. Ένα τέλεια ισορροπημένο δένδρο επιτρέπει τη γρηγορότερη εισαγωγή και ανάκτηση των δεδομένων.

Υπάρχει και η περίπτωση όπου ο κάθε κόμβος έχει το πολύ ένα υπο-δένδρο, οπότε θα δημιουργηθεί λίστα, η οποία αποτελεί μια εκφυλισμένη μορφή δένδρου.

Μαθηματικές ιδιότητες και θεωρήματα για τα δυαδικά δένδρα

Σε ένα δυαδικό δέντρο, στο επίπεδο μηδέν υπάρχουν 2^0 δηλαδή 1 κόμβος. Στο επίπεδο 1 το σύνολο των κόμβων είναι $2^0 + 2^1 = 3$ κόμβοι.



Σχήμα 1.2.1 Δυαδικό δένδρο

Έτσι το μέγιστο δυνατό σύνολο των κόμβων ενός πλήρους δυαδικού δέντρου ύψους *υ* είναι:

$$S = 2^0 + 2^1 + \ldots + 2^k = 2^{\upsilon+1} - 1$$

Αν έχουμε ένα πλήρες δυαδικό δένδρο, τότε ο βαθμός του είναι 2 και προκύπτει από τον παραπάνω τύπο πως ο αριθμός των κόμβων του είναι : $\mathbf{2^{\upsilon+1} - 1}$. Από την σχέση αυτή προκύπτουν τα εξής :

- Ο αριθμός n των κόμβων ενός δυαδικού δένδρου είναι $\mathbf{2^{\upsilon} \le n \le 2^{\upsilon+1} - 1}$ όπου υ το ύψος του δένδρου.
- Οι εσωτερικοί κόμβοι σε ένα πλήρες δυαδικό δένδρο ύψους υ είναι $\mathbf{2^{\upsilon} - 1}$.
- Ο αριθμός L των φύλλων ενός πλήρους δυαδικού δένδρου είναι ίσος με $\mathbf{L = 2^{\upsilon}}$.
- Ο αριθμός n των κόμβων σε ένα πλήρες δυαδικό δένδρο μπορεί επίσης να βρεθεί από τη σχέση **n = 2L – 1,** όπου L ο αριθμός των φύλλων του δένδρου.
- Το ύψος υ ενός δυαδικού δένδρου με n κόμβους είναι μεγαλύτερο ή ίσο από το logn και μικρότερο η ίσο από το n – 1.

Απόδειξη : Το μέγιστο ύψος του δένδρου θα προκύψει αν δεν έχουμε δύο υπο-δένδρα για κάθε κόμβο. Τότε στο δένδρο θα υπάρχει ένα μόνο φύλλο στο κατώτερο επίπεδο, δηλαδή το δένδρο θα είναι εκφυλισμένο σε μια συνδεδεμένη λίστα και θα έχει ύψος ίσο με n – 1, και επομένως θα ισχύει η σχέση $\upsilon \le n - 1$.

Το μικρότερο ύψος θα προκύψει, εάν οι κόμβοι είναι κατανεμημένοι πλήρως ισορροπημένα με τέτοιο τρόπο ώστε να δημιουργηθεί ένα δένδρο με n κόμβους. Όπως έχει αποδειχθεί, ο αριθμός των κόμβων n στην περίπτωση αυτή θα είναι : $n \le 2^{\upsilon+1} - 1$. Από τη σχέση αυτή προκύπτει ότι $n + 1 \le 2^{\upsilon}$. Δηλαδή : $\log (n + 1) \le \log 2^{\upsilon}$ ή $\log n \le \upsilon$, οπότε προκύπτει :

$$\boldsymbol{\log n \le \upsilon \le n - 1}$$

- Τα n διακεκριμένα στοιχεία παράγουν n! διαφορετικά δυαδικά δένδρα.

  Απόδειξη : Τότε, αν έχουμε μια ακολουθία από n διαφορετικούς αριθμούς a_1, a_2, a_3, ..., a_n μπορούμε να δημιουργήσουμε ένα δυαδικό δένδρο για κάθε διάταξή τους. Επειδή υπάρχουν n! διατάξεις για n στοιχεία, συνεπώς θα υπάρχουν n! διαφορετικά δυαδικά δένδρα, τα οποία μπορούν να κατασκευαστούν με αυτούς τους αριθμούς. Για παράδειγμα, αν n = 3, υπάρχουν 6 διαφορετικά δυαδικά δένδρα τα οποία περιέχουν τους αριθμούς a_1, a_2, a_3, καθώς 3! = 6.

1.3. Διάσχιση δυαδικών δένδρων

Ενώ στις γραμμικές δομές δεδομένων (όπως είναι οι συνδεδεμένες λίστες και οι πίνακες μιας διάστασης) ακολουθείται μόνο ένας δρόμος διάσχισης των στοιχείων, σε ένα δένδρο είναι δυνατό να γίνει διάσχιση με περισσότερους από έναν τρόπους, ο καθένας από τους οποίους μπορεί να έχει χρήσιμες ιδιότητες για την επίλυση του κάθε αλγορίθμου.

Με τον όρο **διάσχιση** νοείται η λειτουργία κατά την οποία επισκεπτόμαστε κάθε κόμβο του δένδρου ακριβώς μια φορά με ένα συστηματικό τρόπο. Η λειτουργία της διάσχισης (διέλευσης ή επίσκεψης) των κόμβων επιλύει προβλήματα, όπως αυτά της ταξινόμησης και της αναζήτησης. Η διάσχιση πραγματοποιείται αν ακολουθήσουμε μια λογική διαδρομή στα τρία αναδρομικά συστατικά του δένδρου:

- Τη ρίζα.
- Το αριστερό υπο-δένδρο.
- Το δεξιό υπο-δένδρο.

Τα συστατικά αυτά μπορούν να διασχισθούν και με διαφορετική σειρά και έτσι μπορούμε να έχουμε διαφορετικούς τρόπους διάσχισης:

1. Προδιατεταγμένη διάσχιση (preorder traversal).
2. Ενδοδιατεταγμένη διάσχιση (inorder traversal).
3. Μεταδιατεταγμένη διάσχιση (postorder traversal).
4. Διάσχιση κατά σειρά επιπέδων (level order traversal).

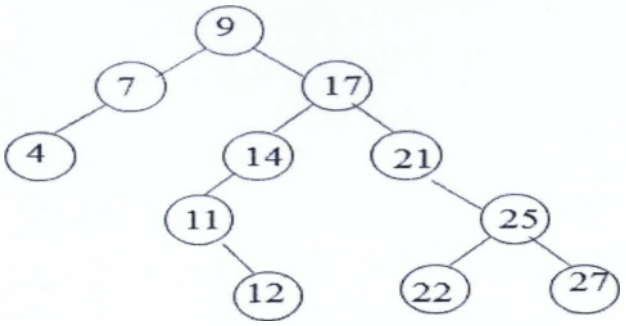
Οι τρεις πρώτες διαδρομές είναι αναδρομικές ενώ η τέταρτη είναι επαναληπτική. Παρακάτω γίνεται η ανάλυσή τους.

1.3.1. Προδιατεταγμένη διάσχιση (preorder traversal)

Σύμφωνα με την προδιατεταγμένη διαδρομή, επισκεπτόμαστε πρώτα τη ρίζα του δένδρου, στη συνέχεια το αριστερό υπο-δένδρο και τέλος το δεξί υπο-δένδρο, ενώ η διάσχιση εκτελείται αναδρομικά. Η διαδρομή αυτή καλείται και επίσκεψη με προτεραιότητα βάθους. Για το δυαδικό

δένδρο που απεικονίζεται στο σχήμα 1.3.1. με την προδιατεταγμένη διαδρομή η διάσχιση θα γίνει με την ακόλουθη σειρά :

9 7 4 17 14 11 12 21 25 22 27



Σχήμα 1.3.1 Δυαδικό δένδρο

Υλοποίηση:

```
  stuck node_tree
{
    data element ;
    node *left, *right ;
    *root;
}
void preorder (node T)
{
    if (T  ! = NULL)
    {
        printf ("%d", T → element) ;
        preorder ( T → left) ;
        preorder ( T → right) ;
    }
}
```

1.3.2. Ενδοδιατεταγμένη διάσχιση (inorder traversal)

Σύμφωνα με την ενδοδιατεταγμένη διαδρομή, επισκεπτόμαστε αρχικά το αριστερό υπο-δένδρο, στη συνέχεια τη ρίζα του δένδρου και τέλος το δεξί υπο-δένδρο με αναδρομικό τρόπο. Για το δυαδικό δένδρο που απεικονίζεται στο σχήμα 1.3.1. με την ενδοδιατεταγμένη διαδρομή η διάσχιση θα γίνει με την ακόλουθη σειρά :

4 7 9 11 12 14 17 21 22 25 27

Υλοποίηση:

```
  stuck node_tree
{
    data element ;
    node *left, *right ;
    *root ;
}
void inorder (node T)
{
    if (T  ! = NULL)
    {
        inorder ( T → left) ;
        printf ("%d", T → element);
        inorder ( T → right) ;
    }
}
```

1.3.3. Μεταδιατεταγμένη διάσχιση (postorder traversal)

Σύμφωνα με την μεταδιατεταγμένη διαδρομή, επισκεπτόμαστε πρώτα το αριστερό υπο-δένδρο, στη συνέχεια το δεξί υπο-δένδρο και ολοκληρώνουμε τη διαδικασία με την επίσκεψη στον ριζικό κόμβο. Για το δυαδικό δένδρο που απεικονίζεται στο σχήμα 1.3.1. με την ενδοδιατεταγμένη διαδρομή η διάσχιση θα γίνει με την ακόλουθη σειρά :

4 7 12 11 14 22 27 25 21 17 9

Υλοποίηση:

```
  stuck node_tree
{
    data element ;
    node *left, *right ;
    *root ;
}
void postorder (node T)
{
    if (T  ! = NULL)
    {
        postorder ( T → left) ;
        postorder ( T → right) ;
        printf ("%d", T → element) ;
    }
}
```

1.3.4. Διάσχιση δένδρου κατά επίπεδο (level order traversal)

Τέλος μπορούμε να διασχίσουμε ένα δένδρο κατά σειρά επιπέδων. Σε αυτή την περίπτωση επισκεπτόμαστε κάθε κόμβο με προτεραιότητα την σειρά του επιπέδου στο οποίο ανήκει. Η επίσκεψη αρχίζει από πάνω προς τα κάτω και από τα αριστερά προς τα δεξιά. Αυτός ο τρόπος διάσχισης καλείται επίσης επίσκεψη με προτεραιότητα πλάτους ή οριζόντια (breadth – first traversal). Η μέθοδος αυτή μας παρέχει πληροφορίες για την πληρότητα των επιπέδων του δένδρου, για το ύψος του, τη μορφολογία του κλπ. Για το δυαδικό δένδρο που απεικονίζεται στο σχήμα 1.3.1. με την διάσχιση κατά σειρά επιπέδων η επίσκεψη θα γίνει με την ακόλουθη σειρά :

9 7 17 4 14 21 11 25 12 22 27

Ο αλγόριθμος είναι επαναληπτικός και χρησιμοποιεί μια δυναμική δομή ουράς αναμονής, στην οποία εισάγονται οι κόμβοι (ουσιαστικά, εισάγονται οι διευθύνσεις των κόμβων). Τα βήματα του αλγορίθμου είναι τα ακόλουθα :

1. Δημιουργία της ουράς αναμονής (queue).
2. Εισάγεται η ρίζα του δένδρου στην ουρά.
3. Όσο η ουρά είναι άδεια, επαναλαμβάνονται τα βήματα 4 και 5.
4. Εξάγεται από την ουρά ο κόμβος x και εκτυπώνεται.
5. Για κάθε παιδί του x γίνεται εισαγωγή στην ουρά.

Για την περιγραφή του αλγορίθμου επίσκεψης κάθε κόμβου ανά σειρά επιπέδων, ακολουθεί κώδικας κατά τον οποίο οι εντολές αντιστοιχίζονται με τα βήματα του αλγορίθμου. Η συνάρτηση delete() εξάγει ένα κόμβο από την ουρά και η insert() εκτελεί την εισαγωγή ενός κόμβου σε αυτή. Οι εξαγωγές και οι εισαγωγές των κόμβων γίνονται με τη χρήση δεικτών, οι οποίοι είναι οι διευθύνσεις των αντίστοιχων κόμβων.

Υλοποίηση :

```
stuck node_tree {
data element ;
node *left, *right ;
*root ; }
insert (q, riza) ;
while (front ! = - 1)
{
    p = delete (q) ;
    printf (" %d ", p → data) ;
    if (p → left ! = NULL) insert (q, p → left) ;
    if (p → right ! = NULL) insert (q, p → right) ;
}
```

ΚΕΦΑΛΑΙΟ 2^{Ο} : Δυαδικά δένδρα αναζήτησης

2.1 Ορισμός και Ιδιότητες

Τα δυαδικά δένδρα αναζήτησης (binary search trees) είναι διατεταγμένα δυαδικά δένδρα στα οποία μεγάλο ρόλο παίζει η διάταξη των παιδιών κάθε κόμβου τους .Τα δυαδικά δένδρα αναζήτησης είναι δενδρικές δομές δεδομένων που υποστηρίζουν, μεταξύ άλλων, τις πράξεις εισαγωγής, διαγραφής και αναζήτησης σε δυναμικά σύνολα. Επομένως, είναι δομές δεδομένων κατάλληλες τόσο για το πρόβλημα του λεξικού όσο και για τη διαχείριση ουρών προτεραιότητας (αν και υπάρχουν και πιο αποδοτικές ουρές προτεραιότητας) σε δυναμικά σύνολα.

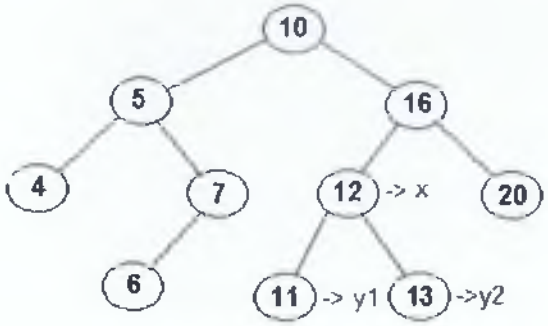
Όπως φαίνεται και παρακάτω, η χρονική πολυπλοκότητα χειρότερης περίπτωσης για αυτές τις πράξεις σε ένα δυαδικό δένδρο αναζήτησης είναι γραμμική στο ύψος του δένδρου. Το ύψος ενός δυαδικού δένδρου αναζήτησης με n στοιχεία μπορεί να είναι από λογαριθμικό στον αριθμό των στοιχείων – O(logn) – μέχρι γραμμικό στον αριθμό των στοιχείων O(n).

Ένα δυαδικό δένδρο αναζήτησης είναι ένα δένδρο με ρίζα (rooted tree) του οποίου κάθε εσωτερικός κόμβος έχει ακριβώς δύο παιδιά. Μόνο οι εσωτερικοί κόμβοι του δένδρου περιέχουν στοιχεία. Τα φύλλα του, που δεν περιέχουν στοιχεία, αντιστοιχούν σε NULL δείκτες και ονομάζονται NULL-φύλλα. Η παρακάτω ιδιότητα χαρακτηρίζει τα δυαδικά δένδρα αναζήτησης :

«Έστω x ένας εσωτερικός κόμβος ενός δυαδικού δένδρου αναζήτησης. Για κάθε εσωτερικό κόμβο y στο αριστερό υπο-δένδρο του x, το στοιχείο του y είναι μικρότερο ή ίσο από το στοιχείο του x, και για κάθε εσωτερικό κόμβο y στο δεξιό υπο-δένδρο του x, το στοιχείο του y είναι μεγαλύτερο ή ίσο από το στοιχείο του x.»

Η ιδιότητα των δυαδικών δένδρων αναζήτησης (binary search tree property) μπορεί να ελεγχθεί στο παράδειγμα του σχήματος 2.1.1. Σαν αποτέλεσμα της ιδιότητας των δυαδικών δένδρων αναζήτησης, η ενδοδιατεταγμένη (inorder) διάσχιση ενός τέτοιου δένδρου τυπώνει 4, 5, 6, 7, 10, 11, 12, 13, 16, 20. Δηλαδή κλήση της Inorder-Traversal (root), όπου root η ρίζα του δένδρου T με n στοιχεία, τυπώνει τα στοιχεία του Tσε αύξουσα σειρά και χρόνο Θ(n).

Απόδειξη : Αναλύεται το ζητούμενο με τη χρήση μαθηματικής επαγωγής. Είναι φανερό ότι η πρόταση ισχύει όταν το δένδρο T έχει ένα μόνον στοιχείο. Έστω ότι το θεώρημα ισχύει για κάθε δυαδικό δένδρο αναζήτησης που έχει το πολύ n στοιχεία.



Σχήμα 2.1 Ιδιότητα δυαδικού δένδρου αναζήτησης

Θεωρούμε ένα δυαδικό δένδρο αναζήτησης με n +1 στοιχεία. Αυτό αποτελείται από τη ρίζα root, το αριστερό υπο-δένδρο T_1 με n – k στοιχεία, και το δεξιό υπο-δένδρο T_2 που έχει k στοιχεία (για κάποιο ακέραιο $k \geq 0$). Η κλήση της Inorder-Traversal (root) τυπώνει πρώτα τα στοιχεία του αριστερού υπο-δένδρου 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 δυαδικών δένδρων αναζήτησης), και μετά τα στοιχεία του δεξιού υπο-δένδρου επίσης σε αύξουσα σειρά. Δηλαδή, όλα τα στοιχεία του δένδρου τυπώνονται σε αύξουσα σειρά.

Όσον αφορά τον χρόνο εκτέλεσης είναι Θ(n – k) +Θ(1) +Θ(k) = Θ(n + 1), όπου ο πρώτος και ο τελευταίος όρος είναι οι χρόνοι εκτέλεσης των αναδρομικών κλήσεων για το αριστερό και το δεξί υπο-δένδρο εξ' αιτίας της επαγωγικής υπόθεσης, και ο δεύτερος όρος είναι ο χρόνος για την κλήση της Inorder-Traversal (root) και την εκτύπωση του στοιχείου της ρίζας.

2.2 Κατασκευή ενός Δυαδικού Δένδρου Αναζήτησης

Ένα δυαδικό δένδρο κατασκευάζεται είτε με συνδέσεις των κόμβων του δυναμικά είτε με τη χρήση ενός πίνακα. Σε κάθε περίπτωση υπάρχουν πλεονεκτήματα και μειονεκτήματα τα οποία παρουσιάζονται ανάλογα με τον τρόπο που θα επιλέξουμε να το κατασκευάσουμε. Συνήθως στα δένδρα, στα στοιχεία των οποίων παρατηρούμε μεταβολές (εισαγωγές ή διαγραφές), επιλέγεται ο δυναμικός τρόπος υλοποίησης. Η αναπαραγωγή ενός δένδρου με πίνακα είναι αποδοτική, όταν το δένδρο είναι πλήρες και όταν δεν έχουμε μεταβολές. Σε άλλη περίπτωση, όταν γίνονται διαγραφές και υπάρχουν μη πλήρεις κόμβοι, δημιουργούνται πολλά κενά στον πίνακα, καταλαμβάνοντας πολύ χώρο στη μνήμη. Το χειρότερο ενδεχόμενο συμβαίνει όταν η

διαγραφή κόμβων απαιτεί τη μετακίνηση δευτερευόντων δένδρων και κάθε κόμβος του υπο-δένδρου πρέπει να μετακινηθεί σε νέα θέση. Σε δυαδικά δένδρα που δεν είναι πλήρη τα κενά μπορούν να αποφευχθούν εύκολα, τοποθετώντας ανύπαρκτους κόμβους εκεί που δεν υπάρχουν. Με τον τρόπο αυτό δημιουργούμε ένα πλήρες δένδρο. Η αναπαράσταση αυτή όμως απαιτεί σημαντικό χώρο στη μνήμη (ο οποίος παραμένει ανεκμετάλλευτος) και για το λόγο αυτό για την κατασκευή ενός δυαδικού δένδρου αναζήτησης θα προτιμήσουμε τον αναδρομικό τρόπο υλοποίησης.

Η δημιουργία ενός δυαδικού δένδρου αναζήτησης με δυναμικό τρόπο γίνεται με συνεχείς εισαγωγές κόμβων. Όταν απαιτείται μια εισαγωγή ενός στοιχείου, δημιουργείται ένας κενός κόμβος με την μορφή που έχει ορισθεί. Ο κόμβος αυτός αρχικά είναι κενός και στη συνέχεια συμπληρώνεται με το στοιχείο που πρέπει να εισαχθεί και τα πεδία των δεικτών, τα οποία χρησιμοποιούνται για τη σύνδεση του κόμβου. Τα πεδία αυτά αρχικά έχουν την τιμή NULL. Η εισαγωγή ενός νέου κόμβου γίνεται με τη σύνδεση του. Ο κόμβος τελικά θα συνδεθεί με εξωτερικό κόμβο σε θέση τέτοια ώστε να τηρείται η βασική ιδιότητα του δυαδικού δένδρου αναζήτησης.

Η εισαγωγή των κόμβων γίνεται ως εξής: Επισκεπτόμαστε το δένδρο ξεκινώντας από τη ρίζα. Στη θέση αυτή συγκρίνουμε το περιεχόμενο του κόμβου με το περιεχόμενο της ρίζας. Αν το περιεχόμενο του προς την εισαγωγή κόμβου είναι μικρότερο, τότε επισκεπτόμαστε το αριστερό υπο-δένδρο, διαφορετικά επισκεπτόμαστε το δεξί. Η δεύτερη επίσκεψη ακολουθεί την ίδια λογική και οι επισκέψεις συνεχίζονται αναδρομικά, μέχρι να μη υπάρχει το επισκεπτόμενο υπο-δένδρο. Τότε ο δείκτης της τελευταίας ρίζας που έχουμε επισκεφθεί είναι προφανώς NULL και θα πάρει την τιμή της διεύθυνσης του νέου κόμβου.

2.3 Είδη δυαδικών δένδρων αναζήτησης

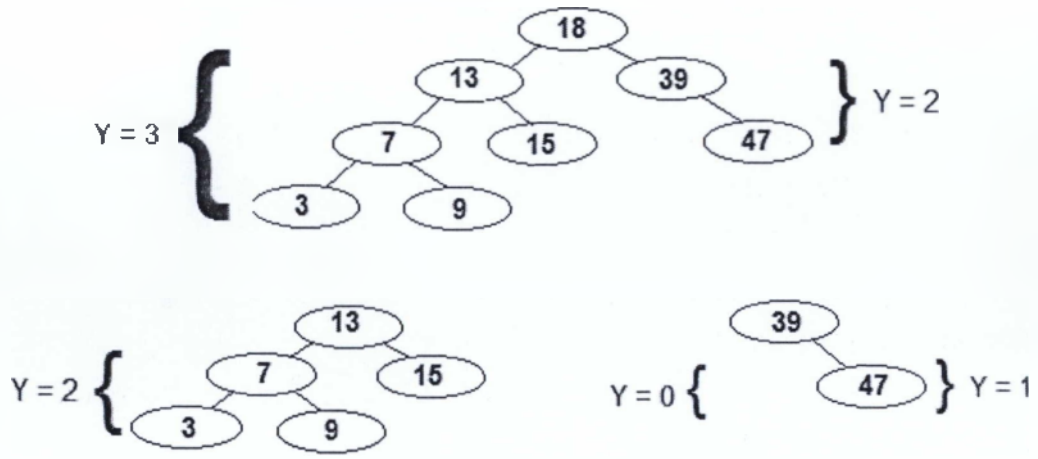
2.3.1 Ισοζυγισμένα Δυαδικά Δένδρα Αναζήτησης (AVL)

Αν τεθούν όρια στο ύψος ενός δυαδικού δένδρου αναζήτησης τότε εκείνο θα μπορεί να τελέσει καλύτερα τις τρεις βασικές πράξεις του (εισαγωγή, διαγραφή και αναζήτηση κόμβου). Το δένδρο στο οποίο έγινε αρχικά κάτι τέτοιο είναι το δένδρο AVL. Η ονομασία του αποτελείται από τα αρχικά των ερευνητών που το εισήγαγαν το 1962 Adel'son, Vel'skii και Landis. Η πρόταση τους ήταν η παρακάτω αμετάβλητη συνθήκη :

«Ένα δυαδικό δένδρο Τ καλείται δένδρο AVL αν και μόνο αν τα ύψη των δύο υπο-δένδρων κάθε εσωτερικού κόμβου διαφέρουν το πολύ κατά ένα.»

Για την τήρηση της ιδιότητας αυτής πρέπει να διατηρήσουμε σε ένα κόμβο το ύψος του. Το δεξί υπο-δένδρο όπως επίσης και το αριστερό υπο-δένδρο ενός δένδρου AVL είναι κι αυτά με τη σειρά τους δένδρα AVL. Ακόμη ένα κενό δένδρο μπορεί να είναι ένα δένδρο AVL. Οι ιδιότητες αυτές πρέπει διατηρούνται ακόμη και αν γίνονται κάποιες παρεμβάσεις (πχ. αναδιάταξη), καθώς νέα στοιχεία μπορεί να προστεθούν στο δένδρο.

Ένα δένδρο AVL καλείται ψηλό από αριστερά όταν το ύψος του αριστερού του υπο-δένδρου είναι μεγαλύτερο από εκείνο του δεξιού του υπο-δένδρου κατά ένα, και αντίστοιχα καλείται ψηλό από τα δεξιά όταν το ύψος του δεξιού του υπο-δένδρου είναι μεγαλύτερο από το ύψος του αριστερού του υπο-δένδρου. Υπάρχουν αρκετές περιπτώσεις στις οποίες η εισαγωγή ή η διαγραφή κόμβου σε ένα δένδρο AVL παραβιάζει τη συνθήκη του χαρακτηρισμού του. Δηλαδή η ιδιότητα ενός τέτοιου δένδρου μπορεί να πάψει να ισχύει με την προσθήκη νέων κόμβων. Η επαναφορά της ιδιότητας αυτής σε ισχύ επιτυγχάνεται με περιστροφή του δένδρου, ανάλογα με την περίπτωση αναδιάταξης που υπάρχει κάθε φορά.



Σχήμα 2.3.1.1 Παράδειγμα δένδρου AVL

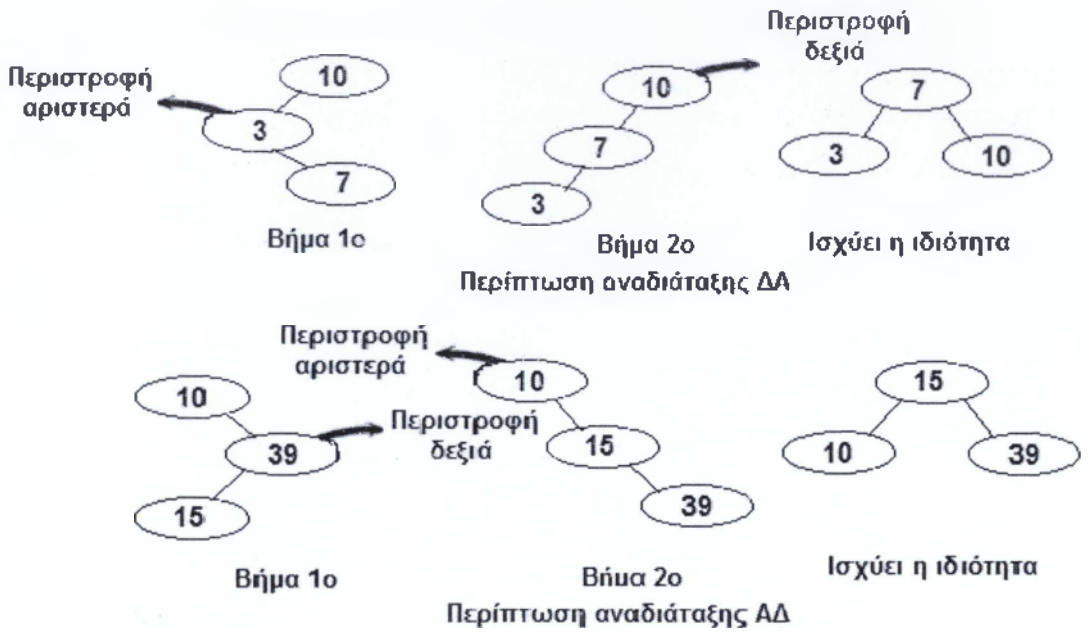
Υπάρχουν τέσσερις περιπτώσεις αναδιάταξης των δένδρων AVL και παρουσιάζονται παρακάτω :

1. ΑΑ – ένα αριστερό υπο-δένδρο ενός δένδρου AVL που είναι ψηλό από αριστερά, γίνεται επίσης ψηλό από τα αριστερά και μετά την αναδιάταξη.
2. ΔΔ - ένα δεξί υπο-δένδρο ενός δένδρου AVL που είναι ψηλό από δεξιά, γίνεται επίσης ψηλό από τα δεξιά και μετά την αναδιάταξη.
3. ΔΑ – ένα υπο-δένδρο ενός δένδρου AVL ψηλό από αριστερά, γίνεται ψηλό από δεξιά.
4. ΑΔ – ένα υπο-δένδρο ενός δένδρου AVL ψηλό από δεξιά, μετά την αναδιάταξη γίνεται ψηλό από δεξιά.

Στο σχήμα 2.3.1.2. φαίνεται η αναδιάταξη των κόμβων ενός δένδρου AVL και στις τέσσερις περιπτώσεις. Με τον όρο περιστροφή εννοούμε την ανταλλαγή ρόλων των δύο εμπλεκόμενων κόμβων. Η περιστροφή έχει αντίθετη φορά από την κατεύθυνση που «γέρνει» το δένδρο για να εξισορροπηθεί η συμμετρία.

Σχήμα 2.3.1.2.α Περιπτώσεις αναδιάταξης ενός δένδρου AVL

Η διπλή περιστροφή ουσιαστικά είναι δύο απλές περιστροφές αντίθετης φοράς. Περιστροφές απλές ή διπλές που αποκαθιστούν την ισορροπία του δένδρου λέγονται και τερματικές. Οι περιστροφές όπως φαίνεται και στο παράδειγμα σχεδίου 2.3.1.2 μεταβάλουν τη μορφή του δένδρου, γι αυτό καλούνται δομικές επαναζυγιστικές πράξεις (reconstruction rebalancing operations). Αντίθετα πράξεις που απλά αλλάζουν κάποια βοηθητική πληροφορία (πχ. το ύψος) καλούνται μη δομικές επαναζυγιστικές πράξεις.



Σχήμα 2.3.1.2.β Περιπτώσεις αναδιάταξης *ενός δένδρου AVL*

Αναζήτηση κόμβου σε δένδρα AVL

Η αναζήτηση σε ένα δένδρο AVL γίνεται με τον ίδιο τρόπο που γίνεται η αναζήτηση σε ένα οποιοδήποτε μη-ισορροπημένο δυαδικό δένδρο αναζήτησης. Λόγω της ισορροπίας ύψους του δένδρου μια αναζήτηση γίνεται σε O(logn) χρόνο. Δεν χρειάζεται να τελεστούν ειδικές πράξεις ούτε αλλάζει η δομή του δένδρου από τις αναζητήσεις. Εάν κάθε κόμβος καταγράφει το μέγεθος του υπο-δένδρου του τότε η αναζήτηση κάθε κόμβου απαιτεί χρόνο O(logn) επίσης.

Εισαγωγή κόμβου σε δένδρα AVL

Η εισαγωγή νέων στοιχείων γίνεται στα φύλλα του δένδρου, όπως και στα δυαδικά δένδρα αναζήτησης. Εντοπίζεται το φύλλο στο οποίο θα γίνει η εισαγωγή και δημιουργείται ο νέος κόμβος. Γίνεται επίσκεψη πίσω, μέχρι την κορυφή του δένδρου, για τον έλεγχο της ισχύος της συνθήκης ισορροπίας AVL σε κάθε βήμα προς τα πίσω, και επαναφορά σε ισορροπία, όπου απαιτείται. Το σχεδιάγραμμα του αλγορίθμου της εισαγωγής έχει ως εξής:

Algorithm InsertItem (Item i)

Input Ένα στοιχείο i

Output Τοποθέτηση του i σε έναν νέο κόμβο, εάν δεν υπάρχει

1. Κλήση της αντίστοιχης μεθόδου εισαγωγής των απλών αζύγιστων δυαδικών δένδρων

2. Με αφετηρία τον κόμβο της εισαγωγής κινούμαστε προς τα επάνω, ώστε να επιδιορθώσουμε τα ύψη, μέχρι να εντοπιστεί ο πρώτος κόμβος u που δεν είναι ισοζυγισμένος.

3. Ισορροπία του κόμβου u βάση της κατάλληλης περιστροφής (απλής ή διπλής όπως φαίνεται και στο σχήμα 2.3.1.2)

end InsertItem

Διαγραφή κόμβου σε δένδρα AVL

Η διαγραφή ενός στοιχείου είναι μια διαδικασία παρόμοια κατά κάποιο τρόπο με την εισαγωγή ενός στοιχείου στα δένδρα AVL και αυτό γιατί σε μια διαγραφή ενός κόμβου είναι πιθανό το δένδρο να χάσει την ισορροπία του και να μην τηρείται πλέον η συνθήκη, γι αυτό να είναι απαραίτητη η κατάλληλη περιστροφή για την εξισορρόπησή του. Υπάρχουν δύο περιπτώσεις που μπορεί να εμφανιστούν μετά την διαγραφή κάποιου κόμβου:

1. Μετά την διαγραφή του κόμβου, παρατηρείται παραβίαση της συνθήκης. Όπως ήδη αναφέρθηκε μπορούμε να κάνουμε απλή ή διπλή περιστροφή για να επαναφέρουμε την ισορροπία στο δένδρο. Δεν αρκεί όμως πάντα αυτό, καθώς αρκετές φορές για να λύσουμε το συνολικό πρόβλημα ίσως χρειαστεί να ανατρέξουμε σε προγόνους των κόμβων που εξετάζουμε για να γίνει κι εκεί έλεγχος για παραβίαση της συνθήκης. Αυτού του είδους οι περιστροφές οι οποίες αντιμετωπίζουν τοπικά αλλά όχι και συνολικά το πρόβλημα ονομάζονται μη τερματικές.
2. Μετά την διαγραφή του κόμβου και την παραβίαση της συνθήκης, με μια απλή ή διπλή τερματική περιστροφή επανέρχεται η ισορροπία στο υπο-δένδρο.

Παρακάτω παρουσιάζεται η περιγραφή της πράξης της διαγραφής κόμβου:

Algorithm DeleteItem (Item i)

Input Ένα στοιχείο i

Output Η διαγραφή του κόμβου που περιέχει το στοιχείο i

1. Κλήση της αντίστοιχης μεθόδου των απλών δυαδικών δένδρων αναζήτησης
2. Με αφετηρία τον κόμβο όπου έγινε η διαγραφής κινούμαστε προς τα επάνω, ώστε να επιδιορθώσουμε τα ύψη
3. Σε κάθε κόμβο u που παρατηρείται παραβίαση της συνθήκης, εκτελείται η κατάλληλη περιστροφή

end DeleteItem

Ανάλυση Πολυπλοκότητας: Το ύψος h ενός δένδρου AVL με n στοιχεία είναι O(logn). Ένα δένδρο AVL n στοιχείων στη χειρότερη περίπτωση επιδεικνύει λογαριθμική συμπεριφορά και για τις τρεις πράξεις. Ακόμη, μια εισαγωγή στοιχείου κοστίζει O(1) ενώ μια διαγραφή O(logn) επαναζυγιστικές πράξεις.

2.3.2 Ερυθρόμαυρα Δέντρα (red-black)

Ένα ερυθρόμαυρο δένδρο είναι μια δομή η οποία ερευνήθηκε το 1972 από τον Rudolf Bayer και είχε ονομαστεί τότε «συμμετρικό δυαδικό B-δένδρο», αλλά κέρδισε τελικά το μοντέρνο όνομά του το 1978 από τους Leonidas J. Guidas και Robert Sedgewick.

Καθώς είναι ένα ισορροπημένο δένδρο εγγυάται πως οι πράξεις της εισαγωγής, αναζήτησης και διαγραφής κόμβου εκτελούνται σε χρόνο O(logn) όπως και τα AVL δένδρα, όπου n είναι ο συνολικός αριθμός των στοιχείων του δένδρου.

Ένα ερυθρόμαυρο δένδρο είναι ένα δυαδικό δένδρο αναζήτησης που εισάγει και διαγράφει στοιχεία με τέτοιο τρόπο έτσι ώστε να μένει πάντα ισορροπημένο. Είναι ένας τύπος δυαδικού δένδρου που χρησιμοποιείται στην επιστήμη των υπολογιστών για να οργανώνει σχετικά μεταξύ τους κομμάτια μνήμης, όπως είναι τα σύμβολα κειμένου και οι αριθμοί.

Ορισμός : Ένα δένδρο T ονομάζεται ερυθρόμαυρο (red-black), όταν όλοι οι κόμβοι του έχουν κόκκινο ή μαύρο χρώμα σύμφωνα με τους ακόλουθους κανόνες :

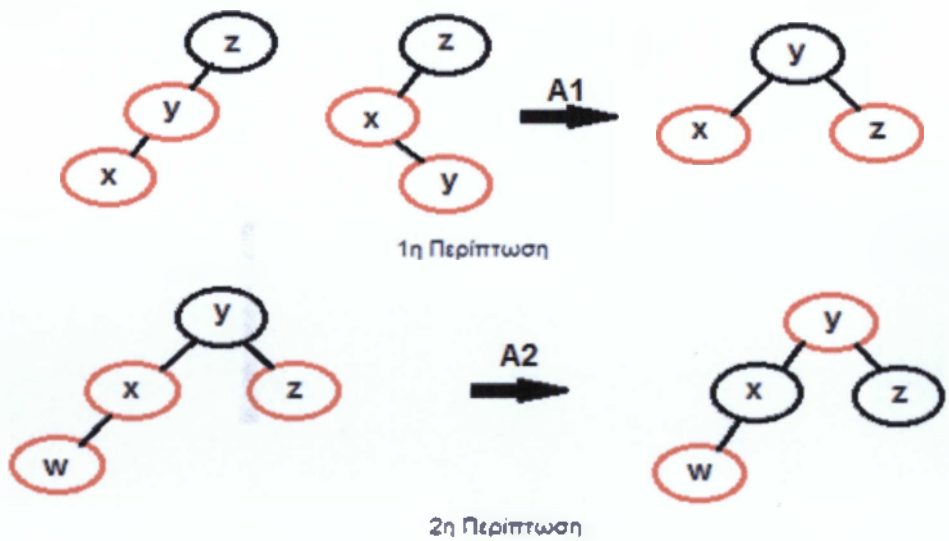
- Η ρίζα του έχει μαύρο χρώμα.
- Όλα τα φύλλα του (τα φύλλα του ερυθρόμαυρου δένδρου δεν περιέχουν στοιχεία) έχουν επίσης μαύρο χρώμα.
- Δεν υπάρχουν δύο διαδοχικοί κόκκινοι κόμβοι, δηλαδή κάθε κόκκινος κόμβος έχει μαύρα παιδιά.
- Όλα τα μονοπάτια από τη ρίζα του δένδρου προς τα φύλλα έχουν τον ίδιο αριθμό μαύρων κόμβων, ή αλλιώς διαθέτουν το ίδιο μαύρο βάθος.

Βασικές πράξεις

Εισαγωγή κόμβου : Η εισαγωγή ενός νέου στοιχείου i εκτελείται αρχικά όπως στα μη-ισορροπημένα δυαδικά δένδρα. Ο κόμβος u στον οποίο θα εισαχθεί το στοιχείο i χρωματίζεται κόκκινος και διαθέτει δύο μαύρα φύλλα. Με τον τρόπο αυτό δεν παραβιάζεται ο τέταρτος κανόνας σχετικά με το μαύρο βάθος των μονοπατιών του δένδρου, αφού ο u ουσιαστικά αντικαθιστά ένα μαύρο φύλλο. Προκαλείται όμως παραβίαση του τρίτου κανόνα καθώς τώρα υπάρχουν δύο διαδοχικοί κόκκινοι κόμβοι εάν ο πατέρας του i είναι κόκκινος κόμβος. Υπάρχουν λοιπόν δύο περιπτώσεις όπως παρουσιάζονται και στο σχήμα 2.3.2.1.

1η περίπτωση : Ο πατέρας του υψηλότερου κόκκινου κόμβου είναι μαύρος, οπότε και το πρόβλημα επιλύεται με περιστροφές. Η πράξη αυτή καλείται A1, είναι δομική καθώς αλλάζει την μορφολογία του δένδρου και τερματική καθώς επιλύεται οριστικά το πρόβλημα, μετακινώντας τον υψηλότερο πλεονάζοντα κόκκινο κόμβο στο υπο-δένδρο του μαύρου πατέρα.

2η περίπτωση : Ο υψηλότερος κόκκινος κόμβος έχει έναν κόκκινο «αδερφό» οπότε θα αντιστρέψουμε τα χρώματα τους με τον μαύρο πατέρα τους. Η πράξη αυτή καλείται A2-αντιστροφή χρωμάτων (color flip). Η πράξη αυτή όπως φαίνεται όμως δεν αλλάζει τη μορφολογία του δένδρου αλλά δεν επιλύει και οριστικά το πρόβλημα κάθως απλά το μεταθέτει στον (πρώην μαύρο) πατέρα όπου θα πρέπει να διενεργήσουμε εκ νέου έλεγχο. Στο παράδειγμα του σχήματος 2.3.2.2. παρουσιάζεται η πράξη της εισαγωγής στοιχείου με χρήση της A1 για την τήρηση των κανόνων.



Σχήμα 2.3.2.1 Περιπτώσεις εισαγωγής στοιχείου σε ερυθρόμαυρα δένδρα. Η 1η περίπτωση είναι μια απλή ή και διπλή περιστροφή και η 2η περίπτωση είναι η αντιστροφή χρωμάτων.

Περιγραφή του αλγορίθμου της εισαγωγής στοιχείου

Algorithm InsertItem (Item i)

Input Ένα στοιχείο i

Output Τοποθέτηση του i σε έναν νέο κόμβο, εάν δεν υπάρχει

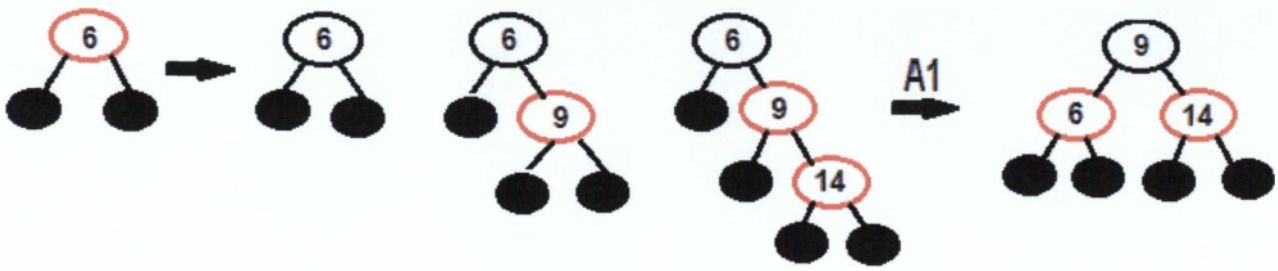
1. Κλήση της αντίστοιχης μεθόδου εισαγωγής των απλών αζύγιστων δυαδικών δένδρων ώστε να εισαχθεί το i σε νέο κόκκινο κόμβο με μαύρα φύλλα

2. Εάν δεν δημιουργήθηκαν δύο διαδοχικοί κόκκινοι κόμβοι

3. Επιστροφή;

4. Όσο υπάρχουν δύο διαδοχικοί κόκκινοι κόμβοι {
5. ανεβαίνουμε προς τα επάνω εκτελώντας όσο γίνεται πράξεις Α2;
6. εκτελούμε και μια πράξη Α1; }
7. Αν η ρίζα r (root) χρωματίστηκε κόκκινη
8. τη χρωματίζουμε μαύρη;

end InsertItem



Σχήμα 2.3.2.2. Εισαγωγή στοιχείων σε ερυθρόμαυρο δένδρο

Διαγραφή κόμβου : Η διαδικασία της διαγραφής κόμβου αφού καλέσει την αντίστοιχη πράξη των μη-ισορροπημένων δυαδικών δένδρων, εξετάζει τον κόμβου που αφαιρέθηκε. Εάν είναι κόκκινος, τότε δεν χρειάζεται να κάνουμε κάτι άλλο. Εάν όμως είναι μαύρος τότε έχουμε παραβίαση της τέταρτης ιδιότητας των ερυθρόμαυρων δένδρων αφού πλέον δεν έχουν τα μονοπάτια το ίδιο μαύρο βάθος. Τότε πιθανό είναι να εμφανίστηκαν κατόπιν της διαγραφής του μαύρου κόμβου δύο διαδοχικοί κόκκινοι κόμβοι, και η μόνη λύση για να ξεπεράσουμε αυτό το πρόβλημα είναι να χρωματίσουμε μαύρο τον χαμηλότερο από αυτούς. Διαφορετικά εφαρμόζουμε τις κατάλληλες επαναζυγιστικές πράξεις. Οι επαναζυγιστικές πράξεις διαγραφής στοιχείου ερυθρόμαυρου δένδρου με βάση το παράδειγμα του σχήματος 2.3.2.3 παρουσιάζονται παρακάτω:

- Η Α1 επαναφέρει την ισορροπία, μετακινώντας με απλή ή διπλή περιστροφή τον κόκκινο κόμβο στο ελλειμματικό υπο-δένδρο ως μαύρο πλέον, διατηρώντας το όποιο χρώμα του παλιού κορυφαίου κόμβου και στον νέο κορυφαίο κόμβο που προκύπτει με την περιστροφή.
- Η Α2a αποκαθιστά το πλήθος των μαύρων κόμβων στο ελλειμματικό μονοπάτι χρωματίζοντας μαύρο τον κόκκινο πατέρα, αφαιρώντας έναν μαύρο κόμβο από το αδερφό μονοπάτι, χρωματίζοντας κόκκινο τον μαύρο αδερφό ώστε να καλύπτει τη συνθήκη.
- Η Α2b μεταθέτει το πρόβλημα στον πατέρα, μειώνοντας τον αριθμό των μαύρων κόμβων στο αδερφό μονοπάτι χωρίς δομικές αλλαγές (οι υπόλοιπες πράξεις είναι τερματικές).

- Η Α3 διενεργώντας απλή περιστροφή δημιουργεί τις συνθήκες για την εφαρμογή της Α1 ή της Α2a, ανάλογα το χρώμα που έχει ο νέος αδερφός του ελλειμματικού κόμβου (τερματική πράξη).

Περιγραφή του αλγορίθμου της εισαγωγής στοιχείου

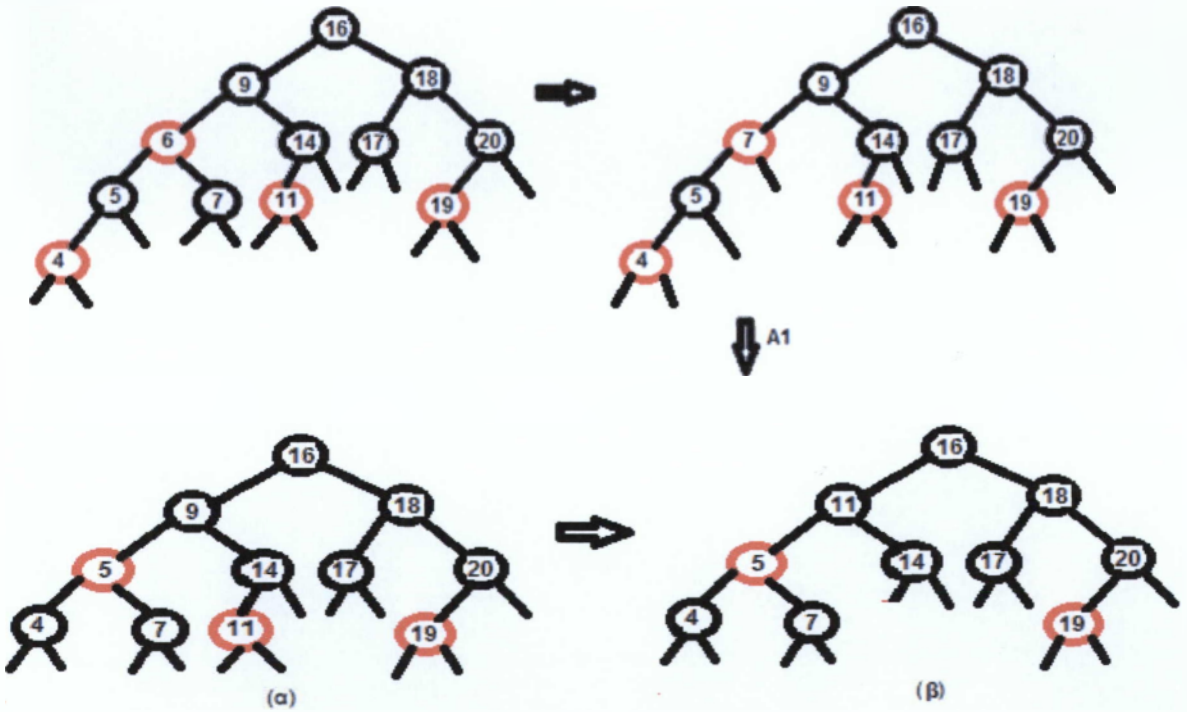
Algorithm DeleteItem (Item i)

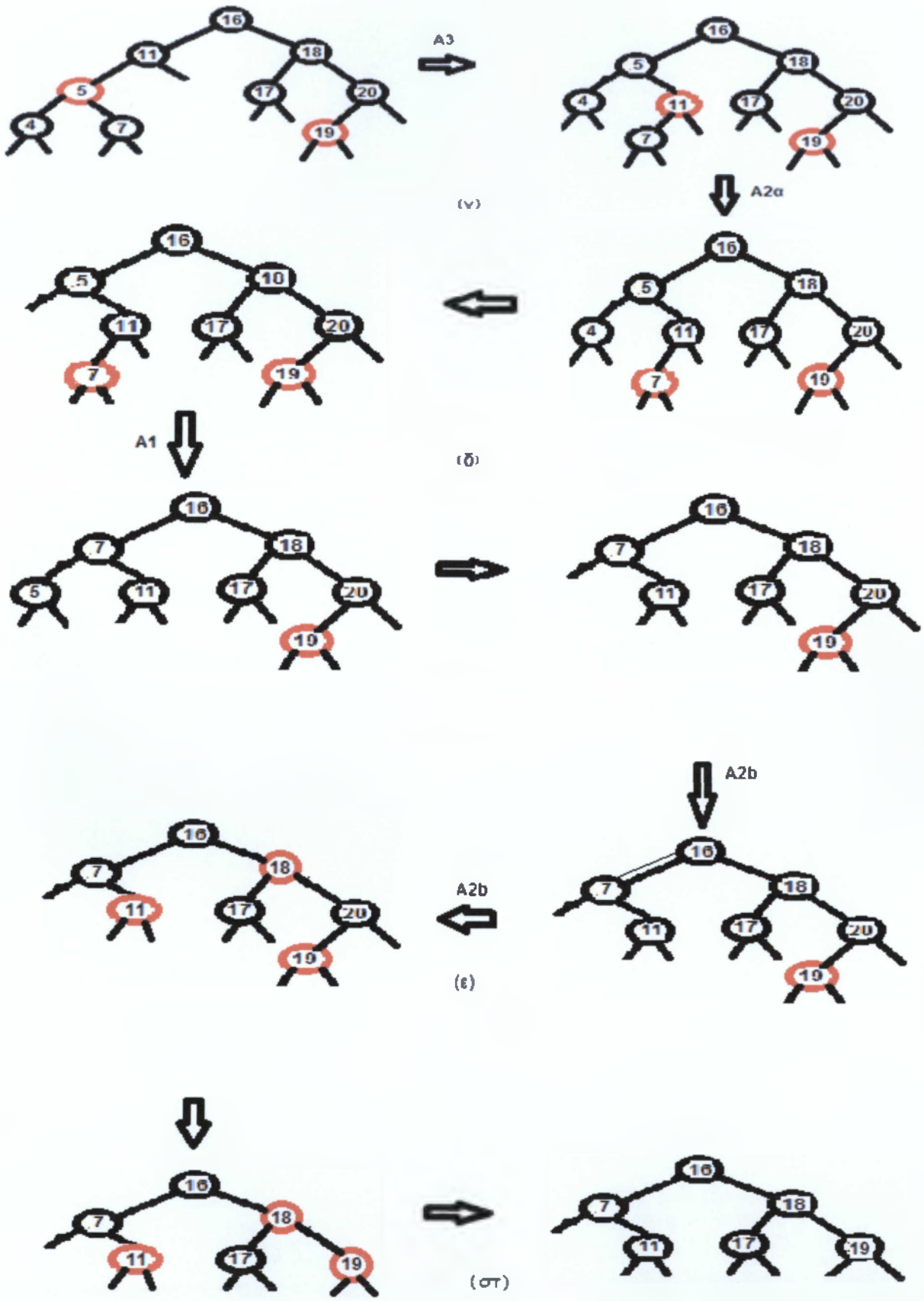
Input Ένα στοιχείο i

Output Η διαγραφή του κόμβου που περιέχει το στοιχείο i, επιστρέφοντας τον εναπομείναντα εμπλεκόμενο κόμβο.

1. Κλήση της αντίστοιχης μεθόδου των απλών δυαδικών δένδρων;
2. Εάν έχει διαγραφεί κόκκινος κόμβος
3. Επιστροφή;
4. Αλλιώς εάν προέκυψαν δύο διαδοχικοί κόκκινοι κόμβοι
5. Χρωματίζουμε μαύρο τον χαμηλότερο από αυτούς
6. Αλλιώς
7. Όσο δεν έχουμε φτάσει στη ρίζα και δεν έχει αναπληρωθεί η απώλεια
8. Ανεβαίνουμε το δένδρο εκτελώντας τις απαραίτητες κατά περίπτωση επαναζυγιστικές πράξεις (σχήμα 2.3.2.4.);

end DeleteItem





Σχήμα 2.3.2.4. Συνεχόμενες επαναζυγιστικές πράξεις κατά τη διαγραφή κόμβων. (α)-(γ) διαδοχικές διαγραφές των 6, 9, 14, (δ)-(ε) διαδοχικές διαγραφές των 4, 5, (στ) διαγραφή του 20.

Πολυπλοκότητα : Το ύψος h ενός ερυθρόμαυρου δένδρου T ενός συνόλου n στοιχείων είναι Θ(logn). Μια διαγραφή σε ένα ερυθρόμαυρο δένδρο T, n στοιχείων απαιτεί O(1) δομικές πράξεις και O(logn) αναχρωματισμούς. Μια ακολουθία n εισαγωγών και διαγραφών, σε ένα άδειο αρχικά ερυθρόμαυρο δένδρο T απαιτεί O(n) επαναζυγιστικές πράξεις.

Τα AVL και τα ερυθρόμαυρα δένδρα ανήκουν στην ίδια κατηγορία δυαδικών δένδρων αναζήτησης κι έτσι μοιάζουν πολύ μαθηματικά. Οι λειτουργίες ισορρόπησης των δένδρων είναι διαφορετικές αλλά απαιτούν χρόνο O(logn).

2.3.3 Βαθμοζυγισμένα Δυαδικά Δένδρα Αναζήτησης (rank-balanced)

Τα ισορροπημένα δένδρα αναζήτησης είναι θεμελιώδη στην επιστήμη των υπολογιστών. Από την ανακάλυψη των AVL δένδρων το 1962 έχουν προταθεί πολλές εναλλακτικές λύσεις με σκοπό είτε την απλούστερη υλοποίηση τους, ή την ταχύτερη απόδοση ή και τα δύο. Απλούστερες υλοποιήσεις ισορροπημένων δένδρων αποτελούν η υλοποίηση του Anderson, των δυαδικών B-δένδρων του Bayer, καθώς και η υλοποίηση του Sedgewick των ερυθρόμαυρων δένδρων. Αυτές οι δομές δεν είναι συμμετρικές, και απλοποιούν την επαναζύγιση εξαλείφοντας περίπου τις μισές περιπτώσεις. Ο Anderson απλοποίησε περισσότερο την υλοποίηση παράγοντας την επαναζύγιση σε δύο διαδικασίες (skew and split) αλλά προσθέτοντας και άλλες ιδέες. Τα ερυθρόμαυρα δένδρα από την άλλη, έχουν αλγορίθμους ενημέρωσης που μπορούν να εγγυηθούν την αποδοτικότητα. Η επαναζύγιση μετά από μια εισαγωγή ή διαγραφή κοστίζει στη χειρότερη περίπτωση O(1) περιστροφές και O(1) χρόνο.

Η σχεδίαση και η ανάλυση των ισορροπημένων δένδρων είναι ένα αντικείμενο το οποίο δεν έχει πλήρως εξερευνηθεί. Τα βαθμοζυγισμένα δένδρα (rank-balanced trees) τα οποία αποτελούν μια παραλλαγή των AVL δένδρων, που έχουν παρόμοιες ιδιότητες με τα ερυθρόμαυρα δένδρα, είναι ακόμη καλύτερα σε ορισμένους τρόπους. Ένα βαθμοζυγισμένο δένδρο χωρίς την πράξη της διαγραφής είναι σαν ένα AVL δένδρο, ενώ με διαγραφές το ύψος του είναι το πολύ ίσο με το ύψος ενός AVL δένδρου ΚΑΙ με τον ίδιο αριθμό εισαγωγών αλλά χωρίς διαγραφές. Τα βαθμοζυγισμένα δένδρα είναι ένα υποσύνολο των ερυθρόμαυρων δένδρων με διαφορετικό κανόνα ισορρόπησης και διαφορετικούς αλγόριθμους για επαναζύγιση. Η εισαγωγή και η διαγραφή κοστίζουν δύο περιστροφές στη χειρότερη περίπτωση και O(1) χρόνο. Τα ερυθρόμαυρα δένδρα χρειάζονται τρείς περιστροφές στη χειρότερη περίπτωση για μια διαγραφή. Η εισαγωγή και η διαγραφή γίνονται από πάνω προς τα κάτω σε O(1) χρόνο.

Στα βαθμοζυγισμένα δένδρα χρησιμοποιούμε μια εκθετική συνάρτηση για να μετρήσουμε την κατανεμημένη αποδοτικότητα των λειτουργιών σε ένα ισορροπημένο δένδρο σε σχέση με το ύψος των κόμβων του.

Για να είναι η πράξη της αναζήτησης, της εισαγωγής και της διαγραφής πιο αποδοτική περιορίζουμε το ύψος του δένδρου θέτοντας ένα κανόνα βαθμού (rank rule) στο δένδρο. Ένα τέτοιο δένδρο, είναι ένα δυαδικό δένδρο όπου κάθε ένας από τους κόμβους του x, έχει έναν ακέραιο rank r(x). Έστω πως οι κόμβοι έχουν βαθμό – 1. Ο βαθμός ενός τέτοιου δένδρου είναι ο βαθμός της ρίζας του.

Ο αρχικός κανόνας βαθμού είναι ότι κάθε φύλλο είναι ένας κόμβος με βαθμό μηδέν, ο βαθμός του κάθε κόμβου είναι το ύψος του ενώ το αριστερό και δεξί υπο-δένδρο έχουν ύψη που διαφέρουν το πολύ κατά ένα. Για να κωδικοποιήσουμε το βαθμό αποθηκεύουμε με κάθε κόμβο (εκτός από τη ρίζα) ένα bit που θα μας δείχνει κάθε φορά το βαθμό του κόμβου (1 ή 2). Ο κανόνας αυτός εγγυάται ένα λογαριθμικό όριο ύψους. Ειδικότερα, ο ελάχιστος αριθμός από κόμβους n_k ικανοποιεί την επανάληψη $n_0=1$, $n_1=2$, $n_k= 1+ n_{k+1} + n_{k-2}$ για $k>1$.

Τα AVL δένδρα υποστηρίζουν την αναζήτηση σε O(logn) χρόνο, αλλά μία εισαγωγή ή διαγραφή μπορούν να προκαλέσουν παραβίαση του κανόνα. Για να διατηρήσουμε τον κανόνα, αλλάζουμε τους βαθμούς συγκεκριμένων κόμβων και κάνουμε περιστροφές για να επαναζυγίσουμε το δένδρο . Η προαγωγή ή η υποβίβαση ενός κόμβου x αυξάνει ή αντίστοιχα μειώνει το βαθμό του κόμβου κατά ένα. Μια περιστροφή σε ένα αριστερό παιδί x με γονέα y κάνει τον y δεξί παιδί του x ενώ διατηρείται η συμμετρική διάταξη. Μια περιστροφή κοστίζει O(1) χρόνο. Στην περίπτωση μιας εισαγωγής εάν ο γονέας του πρόσφατα εισερχόμενου κόμβου ήταν φύλλο ο νέος κόμβος θα έχει διαφορά βαθμού μηδέν, άρα παραβιάζει τον κανόνα. Με τις κατάλληλες περιστροφές και προαγωγές ή υποβαθμίσεις ρυθμίζουμε σωστά το δένδρο.

2.3.3 Βέλτιστα Δυαδικά Δένδρα Αναζήτησης (Optimal Binary Search Trees)

Σε περίπτωση που δεν σχεδιάζουμε κάποια μετατροπή στη δομή ενός δυαδικού δένδρου αναζήτησης (πχ. εισαγωγή και διαγραφή στοιχείων), και γνωρίζουμε με ακρίβεια πόσο συχνά γίνεται η επίσκεψη ενός κόμβου, μπορούμε να κατασκευάσουμε ένα βέλτιστο δυαδικό δένδρο αναζήτησης (optimal binary search tree), το οποίο είναι ένα δένδρο αναζήτησης όπου το μέσο κόστος της αναζήτησης ενός στοιχείου είναι ελάχιστο.

Ακόμη και αν δεν είμαστε σίγουροι πως το κόστος της αναζήτησης θα είναι ελάχιστο, ένα τέτοιο δυαδικό δένδρο αναζήτησης είναι σε θέση να αυξήσει σχετικά το την ταχύτητα της αναζήτησης στους κόμβους του. Παραδείγματος χάριν, εάν έχουμε ένα δυαδικό δένδρο αναζήτησης με στοιχεία του ελληνικές λέξεις που χρησιμοποιούνται για έναν ορθογραφικό έλεγχο, θα μπορούσαμε να ισορροπήσουμε το δένδρο βασιζόμενοι στη συχνότητα εμφάνισης των λέξεων στο κείμενο, τοποθετώντας λέξεις όπως το άρθρο *τα* κοντά στη ρίζα του δένδρου και λέξεις όπως *βαθμοζυγισμένα* κοντά στα φύλλα του δένδρου. Ένα τέτοιο δένδρο μπορεί να συγκριθεί με τα δένδρα Huffman, τα οποία απλά προσπαθούν να τοποθετούν μεγάλης συχνότητας στοιχεία κοντά στη ρίζα του δένδρου για να παράγουν ξεκάθαρες πληροφορίες για κωδικοποίηση – ωστόσο, τα δένδρα Huffman αποθηκεύουν μόνο δεδομένα στους κόμβους τους και τα δεδομένα αυτά δεν χρειάζονται ταξινόμηση.

Αν δεν γνωρίζουμε τη συχνότητα με την οποία επισκεπτόμαστε τα στοιχεία του δένδρου μπορούμε να χρησιμοποιήσουμε άλλους τύπους δένδρων (όπως τα splay δένδρα) για να κάνουμε αναζητήσεις χωρίς να μας επηρεάζει η συχνότητα των αναζητήσεων.

Κεφάλαιο 3°: Λειτουργίες Δυαδικών Δένδρων Αναζήτησης

Η δομή ενός δένδρου παρέχει πολλές δυνατότητες για την επεξεργασία των στοιχείων του. Υπάρχουν διάφορες λειτουργίες που τελούνται πάνω στα δυαδικά δένδρα αναζήτησης. Πράξεις όπως η εκτύπωση των περιεχομένων του κάθε κόμβου, η αναδιάταξη των στοιχείων του δένδρου και η σύγκριση κάθε στοιχείου με κάποιο άλλο δεδομένο. Ένας κόμβος είναι προσβάσιμος ανάλογα με τη μέθοδο διάσχισης θα χρησιμοποιήσουμε κάθε φορά. Ο τρόπος που επιλέγουμε να διασχίσουμε τους κόμβους παίζει καθοριστικό ρόλο στον εντοπισμό τους, ξεκινώντας από τη ρίζα.

Οι λειτουργίες της εισαγωγής, διαγραφής και αναζήτησης στοιχείου υλοποιούνται βασιζόμενες στη μέθοδο διάσχισης που έχουμε επιλέξει. Συμπερασματικά, η διάσχιση ενός δένδρου είναι μια λειτουργία που προηγείται από όλες τις υπόλοιπες λειτουργίες και παίζει καθοριστικό ρόλο στην υλοποίηση τους.

Οι λειτουργίες που τελούνται σε ένα δυαδικό δένδρο αναζήτησης είναι δύο ειδών και παρουσιάζονται παρακάτω:

- Λειτουργίες Ερώτησης
  - Αναζήτηση στοιχείου
  - Ελάχιστο και μέγιστο
  - Επόμενος και προηγούμενος
- Λειτουργίες Ενημέρωσης
  - Εισαγωγή στοιχείου
  - Διαγραφή στοιχείου

3.1 Λειτουργίες Ερώτησης

3.1.1. Αναζήτηση στοιχείου : Η λειτουργία της αναζήτησης είναι ο βασικότερος λόγος δημιουργίας των δυαδικών δένδρων αναζήτησης. Η δομή του πίνακα μας δίνει τη δυνατότητα να κάνουμε αναζήτηση σε λογαριθμικό χρόνο εάν ο πίνακας είναι ταξινομημένος. Είναι όμως κατάλληλη για αναζήτηση όταν δεν γίνονται εισαγωγές ή διαγραφές στον πίνακα, όπως προαναφέραμε, καθώς επηρεάζουν την ταξινόμηση των στοιχείων.

Η συνδεδεμένη γραμμική λίστα δεν αντιμετωπίζει τέτοιο πρόβλημα, έχει όμως ένα μειονέκτημα: η αναζήτηση γίνεται σε χρόνο O(n). Εάν ο αριθμός n είναι πολύ μεγάλος τότε η αναζήτηση με χρήση συνδεδεμένης λίστας δεν είναι αποδοτική.

Με τη χρήση δυαδικού δένδρου η διαδικασία της αναζήτησης δεν αντιμετωπίζει κανένα από τα παραπάνω προβλήματα αφού προσφέρει τον ελάχιστο χρόνο αναζήτησης O(logn), ενώ ταυτόχρονα υπάρχει η δυνατότητα να εισάγουμε και να διαγράφουμε στοιχεία γρήγορα.

Η διαδικασία της αναζήτησης ολοκληρώνεται αναδρομικά με τα εξής βήματα:

- Αν το δένδρο είναι κενό τότε επιστρέφεται η τιμή NULL.
- Ελέγχουμε την τιμή του κόμβου της ρίζας με το κλειδί αναζήτησης x.
- Αν η τιμή του x είναι μικρότερη, κάνουμε την αναζήτηση στο αριστερό υπο-δένδρο.
- Ενώ εάν η τιμή του x είναι μεγαλύτερη, αντίστοιχα κάνουμε την αναζήτηση στο δεξί υπο-δένδρο.

Η συνάρτηση που ακολουθεί υλοποιεί τον αλγόριθμο και χρησιμοποιείται για την αναζήτηση σε ένα δυαδικό δένδρο αναζήτησης. Η συνάρτηση έχει παραμέτρους τη διεύθυνση της ρίζας r (root) και το κλειδί της αναζήτησης x. Εάν που η αναζήτηση είναι επιτυχής επιστρέφει το δείκτη του κόμβου που περιέχει το κλειδί. Σε περίπτωση που η αναζήτηση είναι ανεπιτυχής επιστρέφεται ο δείκτης NULL. Για τον παρακάτω κώδικα θεωρούμε τα στοιχεία του δένδρου σαν ακέραιους αριθμούς.

Υλοποίηση αλγορίθμου

```
struct node_tree
{
    data element;
    node left, right;
} *root;
node find (data x, node r)          /* συνάρτηση αναζήτησης στοιχείου */
{
    if (r == NULL) return NULL;
    if (x < r → element) return find(x, r → left);
    if (x > r → element) return find(x, r → right);
    return r;
}
```

Ο αλγόριθμος των συγκρίσεων που απαιτούνται για μια αναζήτηση n στοιχείων στη χειρότερη περίπτωση ισούται με το ύψος h του δένδρου (μήκος μεγαλύτερης διαδρομής). Άρα ο χρόνος αναζήτησης είναι O(h). Ο χρόνος αυτός μπορεί να γίνει βέλτιστος όταν η τιμή του ύψους του δένδρου είναι ελάχιστη (πλήρη δένδρα). Το αναμενόμενο κόστος μιας αναζήτησης σε ένα

τυχαίο δυαδικό δένδρο αναζήτησης (επιτυχημένη ή όχι) είναι λογαριθμικό ως προς το πλήθος των στοιχείων n του δένδρου.

3.1.2. Ελάχιστο και μέγιστο

Το ελάχιστο στοιχείο ενός δυαδικού δένδρου αναζήτησης βρίσκεται πάντα στον πιο αριστερό κόμβο ενώ αντίστοιχα το μέγιστο στοιχείο ενός δυαδικού δένδρου αναζήτησης βρίσκεται πάντα στον πιο δεξιό κόμβο. Επιλέγουμε τη ενδοδιατεταγμένη διάσχιση καθώς ξέρουμε πως τυπώνει τον πιο αριστερό κόμβο πρώτο και τον πιο δεξιό κόμβο τελευταίο. Ο παρακάτω κώδικας επιστρέφει το ελάχιστο στοιχείο του δένδρου.

Υλοποίηση εύρεσης ελαχίστου στοιχείου

```
node tree_min (node x)                    /* συνάρτηση tree_min */
{
    while (x → left  ! =  NULL)
    x = x → left;
    return x;
}
```

Η παραπάνω συνάρτηση επιστρέφει ένα δείκτη που αποτελεί τη διεύθυνση του κόμβου με το ελάχιστο στοιχείο. Η αναζήτηση αυτή γίνεται σε χρόνο O(logn).

Το ελάχιστο στοιχείο ενός δυαδικού δένδρου όπως προαναφέρθηκε βρίσκεται στον πιο αριστερό κόμβο κάτι που σημαίνει πως εάν ακολουθήσουμε τον δεξιό δείκτη προς τα κάτω μέχρι να πάρει την τιμή NULL βρίσκουμε αυτό το στοιχείο πολύ εύκολα. Με ανάλογο τρόπο όπως παραπάνω βρίσκουμε και το μέγιστο στοιχείο ενός δένδρου. Ακολουθώντας τους δεξιούς δείκτες φτάνουμε στον πιο δεξιό κόμβο – φύλλο .

Υλοποίηση εύρεσης μεγίστου στοιχείου

```
node tree_max (node x)                /* συνάρτηση tree_max */
{
    while (x → right  ! =  NULL)
    x = x → right;
    return x;
}
```

Η συνάρτηση tree_max επιστρέφει ένα δείκτη που αποτελεί τη διεύθυνση του κόμβου με το μέγιστο στοιχείο και πρέπει να σημειωθεί πως η αναζήτηση αυτή γίνεται επίσης σε χρόνο O(logn).

3.1.3. Επόμενος και Προηγούμενος

Σε μερικές περιπτώσεις μπορεί να ενδιαφερθούμε για τον επόμενο (successor) ή τον προηγούμενο (predecessor) του κόμβου x. Αυτοί οι κόμβοι συμβολίζονται με succ(x) και pred(x) αντίστοιχα. Το στοιχείο του επόμενου κόμβου είναι το μικρότερο στοιχείο που ξεπερνά την τιμή του κόμβου x, ενώ αντίστοιχα το στοιχείο του προηγούμενου κόμβου είναι το μεγαλύτερο στοιχείο του δένδρου που δεν ξεπερνά την τιμή του κόμβου x.

Η διαδικασία εύρεσης του επόμενου κόμβου είναι παρόμοια με τη διαδικασία εύρεσης του προηγούμενου κόμβου και είναι η εξής:

Στην περίπτωση που το δεξί παιδί του x είναι εσωτερικός κόμβος (όχι NULL-φύλλο), τότε ο επόμενός του στην ενδοδιατεταγμένη διάσχιση είναι ο πιο αριστερός κόμβος του δεξιού υπο-δένδρου του x. Στην περίπτωση που το δεξί παιδί του x είναι NULL-φύλλο, θεωρούμε το υψηλότερο δυνατό υπο-δένδρο του οποίου ο x είναι ο πιο δεξιός κόμβος. Από όλους τους κόμβους αυτού του υπο-δένδρου, η συγκεκριμένη μέθοδος διάσχισης τυπώνει το στοιχείο του x τελευταίο και αμέσως μετά τυπώνει τον πατέρα της ρίζας του υπο-δένδρου. Δηλαδή 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 δένδρου και δεν υπάρχει επόμενος.

Παρακάτω παρουσιάζεται η υλοποίηση του αλγορίθμου εύρεσης επόμενου στοιχείου. Ο χρόνος εκτέλεσης χειρότερης περίπτωσης είναι O(h) με h το ύψος του δένδρου, αφού στη χειρότερη περίπτωση θα χρειαστεί να ακολουθήσουμε ολόκληρο το μονοπάτι από τον πιο δεξιό κόμβο μέχρι τη ρίζα.

```
Tree – Successor (x)

if x → right ≠ NULL then
    x = x → right;
    while (x → left ≠ NULL)
        x = x → left;
    return (x);
```

```
y = x → par;
while (y ≠ NULL || x = y → right)
    x = y;
    y = x → par;
return (y);
```

Έτσι λοιπόν αντίστοιχα για τον προηγούμενο κόμβο ισχύει:

```
Tree – Predecessor (x)
if x → left ≠ NULL then
    x = x → left;
    while (x → right ≠ NULL)
        x = x → right;
    return (x);
y = x → par;
while (y ≠ NULL || x = y → left)
    x = y;
    y = x → par;
return (y);
```

Ο χρόνος εκτέλεσης χειρότερης περίπτωσης είναι O(h) με h το ύψος του δένδρου, όπως ακριβώς και για την εύρεση του επόμενου στοιχείου.

3.2 Λειτουργίες Ενημέρωσης

Οι λειτουργίες εισαγωγής και διαγραφής στοιχείου οδηγούν στη μεταβολή του συνόλου των στοιχείων. Το δυαδικό δένδρο αναζήτησης πρέπει να αλλάξει ώστε το περιεχόμενο του να ανταποκρίνεται στο ενημερωμένο σύνολο στοιχείων ενώ ταυτόχρονα πρέπει να συνεχίσει να διατηρεί την ιδιότητα των δυαδικών δένδρων αναζήτησης.

3.2.1 Εισαγωγή στοιχείου

Για την εισαγωγή στοιχείων (κόμβων) σε ένα δυαδικό δένδρο αναζήτησης αρχικά επισκεπτόμαστε τη ρίζα του δένδρου. Στη θέση αυτή συγκρίνουμε το περιεχόμενο του κόμβου με το περιεχόμενο της ρίζας. Αν το προς εισαγωγή περιεχόμενο είναι μικρότερο από το

περιεχόμενο της ρίζας τότε επισκεπτόμαστε το αριστερό υπο-δένδρο, ενώ εάν είναι μεγαλύτερο από το περιεχόμενο της ρίζας επισκεπτόμαστε το δεξί υπο-δένδρο. Η δεύτερη επίσκεψη ακολουθεί τη ίδια διαδικασία, όπως και η πρώτη, και αναδρομικά συνεχίζονται οι επισκέψεις έως ότου το επισκεπτόμενο υπο-δένδρο δεν θα υπάρχει. Τότε ο δείκτης της τελευταίας ρίζας που έχουμε επισκεφθεί θα είναι NULL και θα του δώσουμε την τιμή της διεύθυνσης του νέου κόμβου.

Η εισαγωγή ενός στοιχείου μπορεί να γίνει με αναδρομικό άλλα και με επαναληπτικό αλγόριθμο. Και οι δύο παρουσιάζονται παρακάτω και υλοποιούνται με τη χρήση δύο συναρτήσεων.

Αναδρομικός αλγόριθμος

Κάθε νέο στοιχείο εισάγεται στη θέση του NULL-φύλλου που καταλήγει η αναζήτηση γι αυτό το στοιχείο. Η εισαγωγή ενός νέου στοιχείου δεν επηρεάζει την ιδιότητα των δένδρων αυτών. Δημιουργούμε λοιπόν τη συνάρτηση insert() η οποία έχει παραμέτρους το στοιχείο x που αντιπροσωπεύει την τιμή του νέου κόμβου και τη διεύθυνση της ρίζας του δένδρου r. Την πρώτη φορά που καλείται η συνάρτηση δεν υπάρχει δένδρο οπότε το r είναι NULL. Επίσης ο δείκτης r γίνεται NULL όταν φτάσουμε σε κόμβο-φύλλο. Δημιουργούμε πρώτα τον κόμβο και στη συνέχεια τον συνδέουμε. Κάθε φορά που καλείται η συνάρτηση επιστρέφει τη διεύθυνση του επόμενου δένδρου στο οποίο θα συνεχίσουμε την σύγκριση και τελικά μας επιστρέφει τη διεύθυνση της ρίζας του δένδρου.

Υλοποίηση αλγορίθμου εισαγωγής στοιχείου

```
struct node_tree                    /* ορισμός της δομής του κόμβου του δένδρου */
{
    data element;
    diktis left, right;
} *root;
root = insert (data num, root);
node insert (data x, diktis r)
{
    if (r == NULL)
    {
        r = (diktis) malloc (sizeof (node));
        r → element = x;
        r → left = r → right = NULL;
    }
```

```
    else
      {
        if (x < r → element)
          r → left = insert (x, r → left);
        else if (x > r → element)
          r → right = insert (x, r → right);
      }
    return r ;
}
```

Η διαδικασία εισαγωγής ενός νέου στοιχείου σε ένα δυαδικό δένδρο αναζήτησης απαιτεί χρόνο O(logn). Η λειτουργία της εισαγωγή έχει χρόνο χειρότερης περίπτωσης O(h), αφού η αναζήτηση ολοκληρώνεται σε χρόνο O(h) και η εισαγωγή του νέου στοιχείου στη θέση που υποδεικνύει η αποτυχημένη αναζήτηση γίνεται σε χρόνο O(1). Η εισαγωγή μιας ακολουθίας στοιχείων σε ένα αρχικά κενό δένδρο μας οδηγεί σε ένα δένδρο με ύψος Θ(n).

Επαναληπτικός αλγόριθμος

Ο αλγόριθμος αυτός υλοποιείται με μη αναδρομικό τρόπο και παρουσιάζεται μέσα από ένα παράδειγμα. Δημιουργούμε εένα δυαδικό δένδρο αναζήτησης από μια σειρά τυχαίων αριθμών. Οι αριθμοί τοποθετούνται στον πίνακα A[ ]. Με την εισαγωγή του πρώτου στοιχείου στον πίνακα, στη θέση A[0] δημιουργείται η ρίζα του δένδρου. Όλα τα υπόλοιπα στοιχεία που θα εισαχθούν τοποθετούνται σε νέους κόμβους του δένδρου με την ίδια λογική που τοποθετούνται και μέσω του αναδρομικού αλγόριθμου.

Κάθε νέος κόμβος θα συνδεθεί σε έναν ήδη υπάρχοντα, τον οποίο πρέπει να εντοπίσουμε. Η διαδρομή προς τον ζητούμενο κόμβο πραγματοποιείται με επαναληπτική διαδικασία ξεκινώντας από τη ρίζα, τη διεύθυνση της οποίας αναθέτουμε σε ένα δείκτη. Ακολουθούμε το μονοπάτι προς τα κάτω, και ανάλογα με την τιμή του κόμβου που επισκεπτόμαστε συνεχίζουμε τη διαδρομή στο αριστερό ή στο δεξί παιδί του.

Η διαδρομή αυτή ολοκληρώνεται ακολουθώντας τον δείκτη p. Επειδή όμως ο p είναι αυτός που παίρνει τελικά την τιμή NULL, είναι απαραίτητη και η διεύθυνση της προηγούμενης θέσης του p. Την διεύθυνση αυτή την έχει ο δείκτης old, ο οποίος τελικά έχει τη διεύθυνση του κόμβου πάνω από τον οποίο θα τοποθετηθεί ο νέος κόμβος. Η μεταβλητή fl αντιστοιχίζεται στη σημαία (flag) για να μας θυμίζει από πια θέση του τελευταίου κόμβου (δεξιά ή αριστερά) ο δείκτης p πήρε την τιμή NULL.

Υλοποίηση επαναληπτικού αλγορίθμου εισαγωγής (για αρχικά κενό δένδρο)

```
struct node
  {
     data x;
     struct node *left, *right;          /* ορισμός της δομής του κόμβου σε ένα δένδρο */
  }  *root;
  for (i = 0; i < size; i++)                  /* γέμισμα πίνακα */
     b[i] = rand();
  neos = malloc (sizeof (struct node_tree));      /* δημιουργία του ριζικού κόμβου */
  neos → data = b[0];
  neos → left = NULL;
  neos → right = NULL;
  root = neos;
  for (i = 1; i < size; i++)              /* δημιουργία και εισαγωγή νέων κόμβων */
    {/
      neos = malloc (sizeof (struct node_tree));
      neos → data = b[0];
      neos → left = NULL;
      neos → right = NULL;
      p = root;
      while(1)
        {
           old = p;
           if (neos → data < p → data)
             {
                p = p→ left;
                fl = 1;
             }
           else
             {
                p = p → right;
                fl = 2;
             }
           if ((p == NULL) && (fl == 1))
             {
                old → left = neos;
                break;
             }
           if ((p == NULL) && (fl == 2))
             {
                old → right = neos;
                break;
```

```
        }
    }
}
```

3.2.1 Διαγραφή στοιχείου

Αντίθετα με τη διαδικασία της εισαγωγής ενός στοιχείου η διαγραφή αποτελεί μια πιο σύνθετη διαδικασία. Χρειαζόμαστε να μπορούμε να διαγράψουμε κόμβους χωρίς να μεταβάλλονται τα βασικά χαρακτηριστικά της δομής τους. Άρα πρέπει πρώτα να ελέγξουμε εάν ένας κόμβος έχει παιδιά. Υπάρχουν λοιπόν οι παρακάτω περιπτώσεις :

a) Αν ο κόμβος με διεύθυνση p που προκύπτει πως θα διαγραφεί δεν έχει παιδιά, τότε δεν υπάρχει κίνδυνος για μεταβολή της δομής του δένδρου. Ο κόμβος αποδεσμεύεται με την αλλαγή του δείκτη, ο οποίος τον συνδέει στο δένδρο και ανήκει στον πατέρα του. Ο δείκτης που περιέχει την διεύθυνση p του κόμβου παίρνει την τιμή NULL.

b) Αν ο κόμβος έχει παιδί, τότε αυτός διαγράφεται με την αλλαγή του δείκτη που βρίσκεται στον πατέρα του.

c) Αν ο κόμβος έχει δύο παιδιά, τότε αντικαθιστάται ή από το δεξιότερο κόμβο του αριστερού του υπο-δένδρου ή από τον αριστερότερο κόμβο του δεξιού του υπο-δένδρου.

Η συνάρτηση delete () είναι εκείνη με την οποία γίνεται η διαγραφή του κάθε κόμβου. Όταν υπάρχουν δύο παιδιά καλούμε τη συνάρτηση tree_min έτσι ώστε να εντοπίσουμε το ελάχιστο στοιχείο στο δεξιό υπο-δένδρο του κόμβου που βρίσκεται το κλειδί x. Τελικά η delete () επιστρέφει τη διεύθυνση του κόμβου της ρίζας.

Υλοποίηση αναδρομικού αλγόριθμου διαγραφής στοιχείου

```
Node delete (data x, node r)
{
   Node temp;
   if (r == NULL)
   printf («\n element not found»);
   else
      {
         if (x < t → element)
         r → left = delete (x, r → left);
         else if (x > r → element
         r → right = delete (x, r → right);
         else
            {
```

```
            if (r → left && r → right)
              {
                Temp = tree_min (r → right);
                R → element = temp → element;
                Right = delete (r → element, r → right);
              }
            Else if (r → left == NULL)
                R = r → right;
            Else
                R = r →left;
        }
    }
    return r;
}
```

Η πράξη της διαγραφής έχει χρόνο εκτέλεση χειρότερης περίπτωσης O(h) αφού οι πράξεις της αναζήτησης και εύρεσης προηγούμενου και επόμενου στοιχείου ολοκληρώνονται σε χρόνο O(h) και η αντικατάσταση του κόμβου που διαγράφεται από τον κόμβο που περιέχει το κλειδί γίνεται σε σταθερό χρόνο.

Κεφάλαιο 4°: Εφαρμογές Δυαδικών Δένδρων Αναζήτησης

4.1 Προβλήματα 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).

Υπάρχουν πολλοί τρόποι υλοποίησης των λεξικών:

- Μη – ταξινομημένη συνδεδεμένη λίστα.
- Ταξινομημένος πίνακας.
- Δένδρο 2 – 3 (2 – 3 tree).
- Πίνακας Κατακερματισμού (hashing).

Ένα δένδρο 2 – 3 είναι ένα δένδρο στο οποίο κάθε κόμβος που δεν είναι φύλλο έχει 2 ή 3 γιούς, και όλα τα φύλλα έχουν το ίδιο ύψος. Ένα δένδρο που αποτελείται μόνο από ένα κόμβο, τη ρίζα, μπορεί επίσης να είναι ένα δένδρο 2 – 3 .

Για να αναπαραστήσουμε σύνολα με ολική διάταξη, με τη χρήση αυτών των δένδρων, αποθηκεύουμε τα στοιχεία του συνόλου στα φύλλα. Εάν a και b είναι δύο στοιχεία, για τα οποία ισχύει η σχέση a < b, τότε το φύλλο που φέρει το στοιχείο a βρίσκεται στα αριστερά του φύλλου που φέρει το στοιχείο b. Κάθε εσωτερικός κόμβος μπορεί να έχει το πολύ τρεις γιούς. Στο εξής θα αναγνωρίζεται καθένας από αυτούς, με τη σειρά, ως αριστερός, κεντρικός και δεξιός. Τα στοιχεία ενός συνόλου που αποθηκεύονται στα φύλλα του δένδρου είναι ταξινομημένα κατά αύξουσα σειρά. Σε κάθε εσωτερικό κόμβο αποθηκεύουμε τις εξής πληροφορίες :

1. Το μεγαλύτερο στοιχείο στο υπο-δένδρο του αριστερού γιου.
2. Το μεγαλύτερο στοιχείο στο υπο-δένδρο του κεντρικού γιού.

Πράξεις σε λεξικά :

Member (a, v) : η πράξη αυτή υλοποιείται πολύ εύκολα με την υλοποίηση της αντίστοιχης πράξης των δυαδικών δένδρων αναζήτησης. Ξεκινάμε από τη ρίζα και προχωράμε προς τα φύλλα χρησιμοποιώντας τα δεδομένα των κόμβων που συναντάμε.

Έστω v ένας εσωτερικός κόμβος, και l(v), m(v) η επιγραφή που αντιστοιχεί στο μεγαλύτερο στοιχείο του υπο-δένδρου του αριστερού γιού, και το μεγαλύτερο στοιχείο του υπο-δένδρου του

κεντρικού γιού του v. Σε κάθε εσωτερικό κόμβο v συγκρίνουμε το στοιχείο a με την επιγραφή l(v). Εάν a ≤ l(v), τότε το στοιχείο a(εάν υπάρχει) βρίσκεται στο αριστερό υπο-δένδρο. Στην περίπτωση που a > l(v) συγκρίνουμε το στοιχείο a με την επιγραφή m(v). Εάν a ≤ m(v) τότε το στοιχείο a (εάν υπάρχει) βρίσκεται στο κεντρικό υπο-δένδρο. Τέλος στην περίπτωση που a > m(v), το a (εάν υπάρχει) βρίσκεται στο δεξί υπο-δένδρο. Σε όλες τις περιπτώσεις συνεχίζουμε την αναζήτηση προς την κατεύθυνση που μας οδηγεί ο έλεγχος που κάνουμε.

```
member (a, v) :
if v = leaf
 {
   if a = e(v)
     return «yes»;
   else return «no»;
 }
else if a ≥ l(v)
 {
   w = left;
   return member (a, w);
 }
else if a ≤ m(v)
 {
   w = center;
   return member (a, w);
 }
else if w= right
  return member (a, w);
else return «no»;
end
```

Παρατηρείται εύκολα πως ο χρόνος που ξοδεύουμε σε κάθε εσωτερικό κόμβο είναι σταθερός. Έτσι, η πολυπλοκότητα της πράξης αυτής είναι ανάλογη του ύψους του δένδρου 2 – 3, δηλαδή O(logn).

Insert (a, S): για την υλοποίηση της πράξης αυτής θα πρέπει πρώτα να εντοπίσουμε τη θέση που θα τοποθετήσουμε το νέο φύλλο w που περιέχει το στοιχείο a. Αυτό γίνεται με τον ίδιο τρόπο όπως και στην πράξη member. Έστω πως το σύνολο S περιέχει περισσότερα από ένα στοιχεία, η αναζήτηση σταματά σε έναν εσωτερικό κόμβο v, ο οποίος έχει σαν γιούς του 2 ή 3 φύλλα. Αν ο v έχει μόνο δύο γιούς u_1 και u_2, τότε απλά τοποθετούμε τον w σαν τρίτο γιό στην

σωστή σειρά, που σημαίνει πως εάν a < $e(u_1)$ τότε ο w γίνεται αριστερός γιός του v και η επιγραφή a (l(v) :=a). Εάν a>$e(u_1)$ και a<$e(u_2)$ τότε ο w γίνεται κεντρικός γιός του ω και βέβαια η επιγραφή m(v) γίνεται ίση με a (m(v) :=a). Τέλος εάν a > $e(u_2)$ τότε ο w γίνεται δεξιός γιός του v. Σε κάθε περίπτωση οι επιγραφές l και m των προγόνων του v μπορεί να χρειαστεί να τροποποιηθούν.

Delete (a, S): η πράξη αυτή είναι παρόμοια με την πράξη insert. Έστω ότι το στοιχείο a που θέλουμε να διαγράψουμε είναι αποθηκευμένο σε ένα φύλλο, του οποίου πατέρας είναι ο κόμβος v. Έχουμε ήδη αναφέρει πως ο v μπορεί να έχει μέχρι τρεις γιους. Στην περίπτωση που είναι τρεις τότε απλά διαγράφουμε τον κόμβο a, αφήνοντας τον v με δύο γιούς. Εάν χρειάζεται μεταβάλλουμε τις τιμές l και m του v και αναδρομικά των προγόνων του. Εάν ο v έχει μόνο δύο γιους με u_1 τον αριστερό αδερφό του u_2, τότε διαγράφουμε το a, τοποθετούμε τον δεύτερο γιό του v, u_2, σαν δεξιό γιο του u_1 και διαγράφουμε τον v. Εάν ο u έχει τρεις γιους διαγράφουμε το a παίρνουμε το δεξιό γιο του u και τον τοποθετούμε σαν αριστερό γιό του v.

Η εισαγωγή και η διαγραφή στοιχείου γίνεται σε χρόνο O(logn). Αυτό συμβαίνει γιατί το ύψος ενός δένδρου 2 – 3 είναι το πολύ log n. Οι πράξεις αυτές απαιτούν πρώτα την εύρεση της θέσης που θα εισαχθεί ένα νέο στοιχείο ή θα διαγραφεί ένα άλλο. Αυτό αντιστοιχεί σε μια διάσχιση κατά μήκος ενός μονοπατιού από τη ρίζα προς κάποιο φύλλο. Άρα η φάση αυτή μπορεί να εκτελεσθεί σε O(logn) χρόνο. Η δεύτερη φάση είναι η αντίστροφη ακριβώς διαδρομή προς τη ρίζα που επίσης μπορεί να εκτελεσθεί σε χρόνο O(logn) και σε κάθε βήμα της διαδρομής ξοδεύουμε τον ίδιο χρόνο. Επομένως ο συνολικός χρόνος είναι O(logn). Συνεπώς ένα τέτοιο δένδρο μπορεί να χρησιμοποιηθεί και σαν λεξικό αφού μια ακολουθία πράξεων O(n) μπορεί να εκτελεσθεί σε χρόνο O(logn).

Εκτός από τις πράξεις member, insert και delete, στο λεξικό μπορούμε να εκτελέσουμε και τις πράξεις :

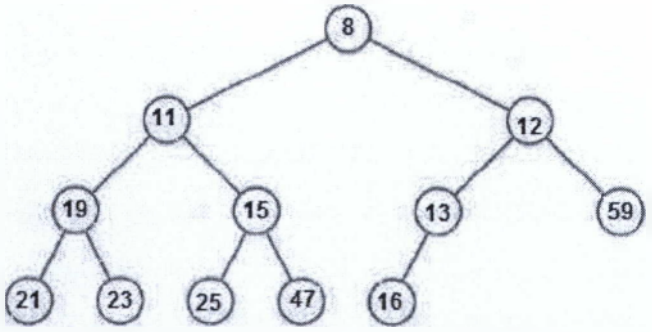
- Εκτύπωση στοιχείου σε αύξουσα ή φθίνουσα σειρά.
- Προηγούμενο και επόμενο στοιχείο.
- Μέγιστο και ελάχιστο στοιχείο.
- n – οστό μικρότερο στοιχείο.

4.2 Σωροί

Η δομή δεδομένων που ονομάζεται (δυαδικός) σωρός είναι ένα αντικείμενο τύπου συστοιχίας που μπορεί να θεωρηθεί σχεδόν ως ένα πλήρες δυαδικό δένδρο όπως φαίνεται και στο σχήμα 4.2.1. Ο κάθε κόμβος του δένδρου αντιστοιχεί σε μια συστοιχία και περιέχει την τιμή του στοιχείου αυτού. Το δένδρο είναι συμπληρωμένο σε όλα τα επίπεδα, εκτός ίσως από το τελευταίο που είναι συμπληρωμένο από τα αριστερά μέχρι κάποιο σημείο. Μια συστοιχία Α που αντιπροσωπεύει ένα σωρό είναι ένα αντικείμενο με δύο πεδία : το μήκος [Α], που αποτελεί το πλήθος των στοιχείων της συστοιχίας, και το πλήθος-σωρού [Α], που αποτελεί το πλήθος των στοιχείων του σωρού που είναι αποθηκευμένα εντός της συστοιχίας Α. Δηλαδή, αν και τα Α [1...μήκος[Α]] μπορεί να περιέχουν ισχύοντες αριθμούς, κανένα στοιχείο εκτός του Α [πλήθος – σωρού [Α]], όπου πλήθος – σωρού [Α] ≤ μήκος [Α], δεν αποτελεί στοιχείο του σωρού. Η ρίζα του δένδρου είναι το Α[1](= 5) και για κάθε θέση i ενός κόμβου, οι θέσεις του πατρικού ΠΑΤΕΡΑΣ(i), του αριστερού θυγατρικού ΑΡΙΣΤΕΡΟΣ(i) και του δεξιού θυγατρικού ΔΕΞΙΟΣ(i) υπολογίζονται εύκολα :

ΠΑΤΕΡΑΣ(i)	ΑΡΙΣΤΕΡΟΣ(i)	ΔΕΞΙΟΣ(i)
επιστροφή [i/2]	επιστροφή 2i	επιστροφή 2i +1

Συνήθως, η διαδικασία ΑΡΙΣΤΕΡΟΣ μπορεί να υπολογίσει το 2i με μία μόνο εντολή, απλά μετατοπίζοντας την αλληλουχία ψηφίων της δυαδικής αναπαράστασης του i κατά μία θέση προς τα αριστερά. Αντίστοιχα, η διαδικασία ΔΕΞΙΟΣ μπορεί να υπολογίσει γρήγορα το 2i + 1 μετατοπίζοντας την αλληλουχία ψηφίων του i κατά μία θέση προς τα αριστερά και προσθέτοντας ως ψηφίο χαμηλότερης τάξης το 1.Η διαδικασία ΠΑΤΕΡΑΣ μπορεί να υπολογίσει το [i/2] μετατοπίζοντας το i κατά μια ψηφιακή θέση προς τα δεξιά. Σε μια υλοποίηση της ταξινόμησης σωρού, οι τρεις αυτές διαδικασίες συνήθως υλοποιούνται ως «μακροεντολές» ή εγγραμισμένες (in line) διαδικασίες.



Σχήμα 4.2.1 Αναπαράσταση σωρού θεωρούμενος ως δυαδικό δένδρο

Υπάρχουν δύο τύποι δυαδικών σωρών : οι σωροί μεγίστου και οι σωροί ελαχίστου. Και στις δύο περιπτώσεις οι τιμές στους κόμβους ικανοποιούν μια ιδιότητα του σωρού, της οποίας τα ιδιαίτερα χαρακτηριστικά εξαρτώνται από τον τύπο του σωρού. Σε ένα σωρό μεγίστου, η ιδιότητα του σωρού είναι ότι για κάθε κόμβο i εκτός από τη ρίζα ισχύει Α[ΠΑΤΕΡΑΣ(i)] ≥ Α[i], δηλαδή η τιμή οπουδήποτε κόμβου είναι το πολύ ίση με αυτή του πατρικού του. Άρα, σε ένα σωρό μεγίστου η ρίζα έχει το μεγαλύτερο στοιχείο, και το υπο-δένδρο που εκφύεται από οποιοδήποτε κόμβο περιέχει τιμές που δεν υπερβαίνουν αυτήν που έχει ο ίδιος ο κόμβος. Οι σωροί ελαχίστου είναι δομημένοι με την αντίθετη διάταξη: Α[ΠΑΤΕΡΑΣ(i)] ≤ Α[i]. Σε ένα σωρό ελαχίστου η έχει το μικρότερο στοιχείο.

Για τον αλγόριθμο ταξινόμησης σωρού, χρησιμοποιούνται οι σωροί μεγίστου, ενώ οι σωροί ελαχίστου χρησιμοποιούνται συνήθως σε ουρές προτεραιότητας. Θεωρώντας τον σωρό ως δένδρο ορίζουμε το ύψος ενός κόμβου του σωρού ως το πλήθος των ακμών και την μακρύτερη απλή διαδρομή από τον κόμβο μέχρι κάποιον καταληκτικό κόμβο. Το ύψος του σωρού ορίζεται ως το ύψος της ρίζας του. Δεομένου ότι ο σωρός n στοιχείων βασίζεται σε ένα πλήρες δυαδικό δένδρο, το ύψος του οποίου είναι Θ(logn). Οι βασικές πράξεις στους σωρούς έχουν χρόνο εκτέλεσης το πολύ ανάλογο προς το ύψος του δένδρου O(logn) και είναι οι παρακάτω:

- Η διαδικασία *Αποκατάσταση σωρού μεγίστου* καλείται σε χρόνο O(logn) και είναι η βασική διαδικασία που εξασφαλίζει τη διατήρηση της ιδιότητας του σωρού.
- Η διαδικασία *Κατασκευή σωρού μεγίστου* εκτελείται σε γραμμικό χρόνο και δημιουργεί ένα σωρό μεγίστου από μία μη ταξινομημένη συστοιχία εισόδου.
- Η διαδικασία *Ταξινόμηση σωρού* εκτελείται σε χρόνο O(nlogn) και ταξινομεί μια συστοιχία επί τόπου.
- Οι διαδικασίες *Εισαγωγή σε σωρό μεγίστου, Εξαγωγή μεγίστου σωρού, Αύξηση κλειδιού σωρού* και *Μέγιστο σωρού* εκτελούνται σε χρόνο O(logn) και καθιστούν δυνατή τη χρήση σωρού ως ουράς προτεραιότητας.

Διατήρηση ιδιότητας σωρού : Η διαδικασία *Αποκατάσταση σωρού μεγίστου* είναι ένα σημαντικό υποπρόγραμμα για τη διαχείριση σωρών μεγίστου. Δέχεται ως είσοδο μια συστοιχία Α και τη θέση i κάποιου στοιχείου της συστοιχίας. Η λειτουργία της προϋποθέτει ότι τα δυαδικά δένδρα που εκφύονται από τους κόμβους ΑΡΙΣΤΕΡΟΣ και ΔΕΞΙΟΣ είναι σωροί μεγίστου αλλά το Α[i] ενδέχεται να είναι μικρότερο από τους θυγατρικούς κόμβους, ενώ έτσι γίνεται παραβίαση της συνθήκης. Η *Αποκατάσταση του σωρού μεγίστου* μετακινεί την τιμή του Α[i] προς τα κάτω έτσι ώστε το υπο-δένδρο που εκφύεται από το στοιχείο που βρίσκεται στη θέση i να καταστεί σωρός μεγίστου.

Υλοποίηση:

```
l ← left;
r ← right;
if l ≤ plithos[A] & A[l] < A[i]
 {
   # - max ← l;
   else # - max ← r;
 }
if r ≤ plithos[A] & A[r] > A[# - max]
   # - max ← r;
if # - max ≠ l
   A[i] ↔ A[# - max];
   Reserve max heap (A, # - max)
```

Ο χρόνος εκτέλεσης της *Αποκατάστασης σωρού μεγίστου* σε ένα υπο-δένδρο μεγέθους n που εκφύεται από κάποιον κόμβο i αντιστοιχεί στο χρόνο Θ(1) που απαιτείται για να αποκατασταθούν οι σωστές σχέσεις μεταξύ των στοιχείων A[i], A[ΑΡΙΣΤΕΡΟΣ(i)] και A[ΔΕΞΙΟΣ(i)], συν το χρόνο εκτέλεσης της *Αποκατάστασης σωρού μεγίστου* σε ένα υπο-δένδρο που εκφύεται από κάποιον από τους θυγατρικούς του κόμβου του i. Ο χρόνος εκτέλεσης στη χειρότερη περίπτωση, που προκύπτει όταν η τελευταία γραμμή του δένδρου είναι ακριβώς ημι-συμπληρωμένη , περιγράφεται από την αναδρομική σχέση

$$T(n) \leq T(2n/3) + \Theta(1)$$

και σύμφωνα με το κεντρικό θεώρημα O(logn)- ή εναλλακτικά ο χρόνος εκτέλεσης της διαδικασίας αυτής σε ένα κόμβο h είναι O(h).

Ο αλγόριθμος της ταξινόμησης σωρού (heap sort) κατασκευάζει ως πρώτο βήμα μέσω της *Κατασκευής σωρού μεγίστου* ένα σωρό μεγίστου από μια συστοιχία εισόδου A[1...n], όπου n=μήκος [A]. Δεδομένου ότι το μέγιστο στοιχείο της ρίζας είναι αποθηκευμένο στη ρίζα A[1], μπορούμε να τοποθετήσουμε στη σωστή τελική του θέση εναλλάσσοντας το με το στοιχείο A[n]. Εάν στη συνέχεια απορρίψουμε τον κόμβο n από τον σωρό παρατηρούμε ότι η υπο-συστοιχία A[1...(n – 1)] μπορεί να μετατραπεί εύκολα σε σωρό μεγίστου. Οι θυγατρικοί σωροί της ρίζας παραμένουν σωροί μεγίστου, αλλά το νέο στοιχείο της ρίζας ενδέχεται να παραβιάζει την ιδιότητα του σωρού μεγίστου. Ωστόσο αυτό το πρόβλημα αντιμετωπίζεται με την κλήση της *Αποκατάστασης σωρού μεγίστου* στο τέλος της οποίας η συστοιχία A[1...(n – 1)] αντιπροσωπεύει ένα σωρό μεγίστου με μέγεθος n – 1, μέχρι ένα σωρό με μέγεθος 2.

Ταξινόμηση σωρού (heap sort)

1. build heap (A)
2. for (i = size; i >= 2; i --)
3. temp = A[1];
4. A[1] = A[i];
5. A[i] = temp;
6. plithos[A] ← plithos[A] – 1;
7. plithos(A,1);

Η διαδικασία ταξινόμησης σωρού απαιτεί χρόνο O(nlogn) δεδομένου ότι η κλήση στην *Κατασκευή σωρού μεγίστου* απαιτεί χρόνο O(n) και καθεμία από τις n – 1 κλήσεις στην *Αποκατάσταση σωρού μεγίστου* απαιτεί χρόνο O(logn).

4.3 Ουρά Προτεραιότητας

Σε περιπτώσεις παροχής υπηρεσιών διάθεσης και χρήσης κοινών προκύπτει η ανάγκη της παροχής ειδικής προτεραιότητας σε εκείνους που αναμένουν να ικανοποιηθούν. Ένα παράδειγμα μιας τέτοιας ουράς είναι τα επείγοντα εξωτερικά ιατρεία όπου οι ασθενείς δεν εξετάζονται βάση του χρόνου άφιξης τους αλλά βάση της σοβαρότητας της κατάστασής τους. Ακόμη τα λειτουργικά συστήματα χρησιμοποιούν συνεχώς ειδικές ουρές οι οποίες παρέχονται όχι με χρονολογική σειρά αλλά με σειρά σημαντικότητας.

Σε αυτές τις περιπτώσεις οι γνωστές απλές ουρές αναμονής FIFO (first in first out) δεν μπορούν να ανταποκριθούν σε ειδικές απαιτήσεις προτεραιοτήτων. Έχουμε την ανάγκη να χρησιμοποιήσουμε μια δομή δεδομένων στην οποία σημασία έχει η προτεραιότητα κάθε στοιχείου και να βγαίνει πάντα πρώτο το στοιχείο με τη μεγαλύτερη προτεραιότητα. Η ουρά αυτή ονομάζεται ουρά προτεραιότητας (priority queue).

Η ουρά προτεραιότητας είναι μια συλλογή στοιχείων στην οποία ορίζονται οι πράξεις της εισαγωγής και εξαγωγής στοιχείων όπως και στην απλή ουρά, με τη διαφορά ότι κατά την εξαγωγή το στοιχείο που εξάγεται πρώτο είναι το στοιχείο με τη μεγαλύτερη προτεραιότητα.

Η χρήση μιας ουράς προτεραιότητας μοιάζει με τη χρήση μιας ουράς και μιας στοίβας, αλλά η αποδοτική υλοποίηση της είναι δυσκολότερη. Για την ακρίβεια η ουρά προτεραιότητας είναι μια γενίκευση των δομών στοίβας και ουράς, επειδή μπορούμε να υλοποιήσουμε αυτές τις δομές με ουρές προτεραιότητας χρησιμοποιώντας κατάλληλες αναθέσεις προτεραιότητας στα στοιχεία.

Μια δομή δεδομένων με την οποία είναι δυνατό να κάνουμε σχεδόν άμεσα εισαγωγές και διαγραφές καθώς επίσης και να εντοπίσουμε το μέγιστο και το ελάχιστο στοιχείο λέγεται ουρά προτεραιότητας.

Υπάρχουν πέντε τρόποι υλοποίησης μιας ουράς προτεραιότητας και παρουσιάζονται παρακάτω:

- Με ταξινομημένο πίνακα, όταν τα στοιχεία είναι εξαρχής γνωστά.
- Με τη χρήση μιας συνδεδεμένης λίστας.
- Με μια διατεταγμένη συνδεδεμένη λίστα.
- Με τη χρήση ενός δυαδικού δένδρου.
- Με τη χρήση σωρού.

Η δομή εκείνη που ικανοποιεί καλύτερα την απαίτηση εξαγωγής του στοιχείου με μεγαλύτερη προτεραιότητα κάθε φορά, με πιο αποτελεσματικό και αποδοτικό τρόπο είναι ο σωρός (heap), η οποία μας δίνει άμεσο χρόνο της τάξης O(1).

Η χρήση ενός δυαδικού δένδρου παρέχει μια επίσης καλή δυνατότητα υλοποίησης . Στην περίπτωση αυτή τόσο η πράξη της εισαγωγής όσο και εκείνη της εξαγωγής, γίνονται σχετικά πιο εύκολα (σε χρόνο O(logn)), όσο είναι δηλαδή το ύψος ενός ισοζυγισμένου δυαδικού δένδρου αν και το κόστος δημιουργίας του δένδρου δεν παραβλέπεται.

Η ουρά προτεραιότητας είναι ένας πολύ καλός τρόπος υλοποίησης της ταχυταξινόμησης και παρουσιάζει καλύτερη επίδοση από τον αλγόριθμο ταξινόμησης σωρού που παρουσιάστηκε σε προηγούμενη παράγραφο. Μια ουρά προτεραιότητας είναι μια δομή δεδομένων για τη τήρηση ενός συνόλου στοιχείων S, σε καθένα από τα οποία αντιστοιχεί μια τιμή που ονομάζεται κλειδί. Μια ουρά προτεραιότητας υποστηρίζει της παρακάτω πράξεις:

1. Εισαγωγή (S, x) εισάγει το στοιχείο x στο σύνολο S.
2. Μέγιστο (S) επιστρέφει το στοιχείο του συνόλου με το μεγαλύτερο κλειδί.
3. Εξαγωγή μεγίστου αφαιρεί και επιστρέφει το στοιχείο του συνόλου με το μεγαλύτερο κλειδί.
4. Εξαγωγή ελαχίστου αφαιρεί και επιστρέφει το στοιχείο του συνόλου με το μικρότερο κλειδί.
5. Αύξηση κλειδιού αντικαθιστά την τιμή του κλειδιού του στοιχείου με μια άλλη τιμή η οποία θα πρέπει να είναι ίση ή μεγαλύτερη της τρέχουσας τιμής του κλειδιού.
6. Μείωση κλειδιού αντικαθιστά την τιμή του κλειδιού του στοιχείου με μια άλλη τιμή η οποία θα πρέπει να είναι ίση ή μικρότερη της τρέχουσας τιμής του κλειδιού.

Μια ουρά προτεραιότητας μεγίστου τηρεί τις προς εκτέλεση εργασίες και τις σχετικές τους προτεραιότητες και όταν μια εργασία ολοκληρωθεί ή διακοπεί, επιλέγεται μεταξύ των εργασιών σε εκκρεμότητα εκείνη με την υψηλότερη προτεραιότητα μέσω της Εξαγωγής μεγίστου, ενώ παράλληλα ανά πάσα στιγμή μπορεί να προστεθεί μια νέα εργασία μέσω της Εισαγωγής.

Μια ουρά προτεραιότητας ελαχίστου υποστηρίζει της πράξεις της Εισαγωγής, Εξαγωγής ελαχίστου και Μείωση κλειδιού. Σε κάθε βήμα χρησιμοποιούμε την πράξη Εξαγωγή ελαχίστου για να επιλέξουμε το επόμενο στοιχείο προς εξαγωγή. Αν θέλουμε να προσθέσουμε νέα στοιχεία μπορούμε να το κάνουμε μέσω της Εισαγωγής.

Ένα πλήρες δυαδικό δένδρο αντικαθιστάται από το σωρό (heap) για την υλοποίηση της ουράς προτεραιότητας. Σε μια συγκεκριμένη εφαρμογή, όπως είναι ο χρονοπρογραμματισμός εργασιών, τα στοιχεία της ουράς προτεραιότητας αντιστοιχούν στα αντικείμενα της εφαρμογής. Δεδομένου ότι οι πράξεις σωρού μεταβάλλουν τις θέσεις των στοιχείων του σωρού εντός τη συστοιχίας, σε μία πραγματική υλοποίηση όταν μετακινείται ένα στοιχείο του σωρού θα πρέπει επίσης να ενημερώνεται και η λαβή του (το αν θα είναι δείκτης ή ακέραιος αριθμός κλπ.) στο αντίστοιχο αντικείμενο της εφαρμογής.

Παρακάτω παρουσιάζονται περιγραφικά οι διαδικασίες μέσω των οποίων υλοποιούμε μια ουρά προτεραιότητα μεγίστου. Αντίστοιχες είναι και οι ενέργειες που πρέπει να κάνουμε σε μία ουρά προτεραιότητας ελαχίστου. Η πράξη Μέγιστο υλοποιείται με την βοήθεια της διαδικασίας Μέγιστο σωρού σε χρόνο Θ(1).

```
Μέγιστο Σωρού (A)
  return A[1];
```

Η διαδικασία Εξαγωγή μεγίστου σωρού είναι O(logn), καθώς πέρα από το χρόνο που απαιτεί η Αποκατάσταση σωρού μεγίστου, εκτελείται απλά ένα σταθερό πλήθος εργασιών. Η διαδικασία αυτή υλοποιεί την πράξη Εξαγωγή Μεγίστου, ενώ μοιάζει πάρα πολύ με τη διαδικασία Ταξινόμηση σωρού (heap sort).

```
Εξαγωγή Μεγίστου (A)
  if plithos[A] < 1
    then error ;
  max ← A[1];
  A[1] ← A[plithos[A]];
  plithos[A] ← plithos[A] - 1
  reserve max heap (A, 1);
  return max;
```

Η διαδικασία Αύξηση κλειδιού σωρού υλοποιεί τη πράξη Αύξηση κλειδιού. Το στοιχείο της ουράς προτεραιότητας του οποίου το κλειδί πρόκειται να αυξηθεί προσδιορίζεται μέσω ενός αύξοντα αριθμού της συστοιχίας. Η διαδικασία αυτή αρχικά δίνει στο κλειδί του στοιχείου A[i] τη νέα του τιμή. Επειδή όμως η αλλαγή αυτή ενδέχεται να επηρεάσει τη συνθήκη, η διαδικασία διατρέχει μια διαδρομή από τον κόμβο αυτό προς τη ρίζα για να προσδιορίσει την κατάλληλη θέση για το επαυξημένο κλειδί. Συγκρίνει λοιπόν, το εκάστοτε τρέχον στοιχείο με τον πατέρα του εναλλάσσοντας τα κλειδιά τους συνεχίζοντας τη διαδικασία μέχρι το κλειδί του τρέχοντος στοιχείου να είναι μικρότερο και έτσι ικανοποιείται η ιδιότητα του σωρού

```
Αύξηση Κλειδιού Σωρού (A, i, k)
 if k < A[i]
   then error
A[i] = k;
while i > 1 & A[father(i)] < A[i]
   temp = A[father(i)];
   A[father(i)] = A[i];
   A[i] = temp;
   i = father(i);
```

Η διαδικασία Εισαγωγή σε σωρό μεγίστου υλοποιεί τη πράξη Εισαγωγή. Δέχεται ως είσοδο το κλειδί του νέου στοιχείου που πρόκειται να εισαχθεί στο σωρό μεγίστου. Αρχικά, η διαδικασία, επεκτείνει το σωρό μεγίστου προσθέτοντας στο δένδρο ένα νέο καταληκτικό κόμβο με κλειδί $-\infty$. Στη συνέχεια καλεί την διαδικασία Αύξηση κλειδιού σωρού για να αποδώσει στο κλειδί αυτού του νέου κόμβου τη σωστή του τιμή και να αποκαταστήσει την ιδιότητα του σωρού μεγίστου.

```
Εισαγωγή σε σωρό μεγίστου (A, k)
plithos[A] ← plithos[A] + 1
A[plithos[A]] = - ∞;
Afxisi (A, plithos[A], k);
```

Ο χρόνος εκτέλεσης της Εισαγωγής σε σωρό μεγίστου σε ένα σωρό με n στοιχεία είναι O(logn).

4.4 Κωδικοποίηση Huffman (Huffman encoding)

Η κωδικοποίηση Huffman είναι μια τεχνική συμπίεσης δεδομένων η οποία μπορεί να εξοικονομήσει χώρο στα αρχεία κειμένου, αλλά και σε άλλα είδη αρχείων. Η ιδέα είναι να εγκαταλείψουμε τον τρόπο που τα αρχεία κειμένου συνήθως αποθηκεύονται. Αντί να χρησιμοποιούμε τα συνηθισμένα εφτά ή οχτώ bits για κάθε χαρακτήρα, με τη μέθοδο Huffman χρησιμοποιούμε λίγα bits για λέξεις που χρησιμοποιούνται συχνά και περισσότερα για τους χαρακτήρες που σπάνια χρησιμοποιούνται.

Ένα τέτοιο αρχείο πληροφοριών μπορεί να αναπαρασταθεί με διάφορους τρόπους. Μας ενδιαφέρει η σχεδίαση ενός δυαδικού αλφαριθμητικού κώδικα με βάση τον οποίο ο κάθε χαρακτήρας αναπαριστάται από μία μοναδική δυαδική συμβολοσειρά. Χρησιμοποιώντας ένα κώδικα σταθερού μήκους απαιτούνται 3 ψηφία για να αναπαραστήσουμε έξι χαρακτήρες:α=000, β=001, ... ζ=101. Με τη μέθοδο αυτή απαιτούνται πάρα πολλά ψηφία για να κωδικοποιήσουμε ένα αρχείο.

Μπορούμε να μειώσουμε τον όγκο των κωδικοποιημένων δεδομένων αν αντί για κώδικα σταθερού μήκους, χρησιμοποιήσουμε κώδικα μεταβλητού μήκους, ο οποίος αποδίδει κωδικούς μικρού μήκους στους χαρακτήρες που εμφανίζονται συχνά και κωδικούς μεγάλου μήκους στους χαρακτήρες που εμφανίζονται σπανιότερα. Έτσι συνολικά για την κωδικοποίηση ενός αρχείου χρειαζόμαστε λιγότερα ψηφία.

Έστω ότι θέλουμε να κωδικοποιήσουμε τη συμβολοσειρά «a simple string to be encoded using a minimal number of bits». Κωδικοποιώντας τη συμβολοσειρά αυτή με δυαδική αναπαράσταση των πέντε ψηφίων όπου το i αντιπροσωπεύεται από το i-οστό γράμμα του αλφαβήτου – 0 για το κενό – προκύπτει η παρακάτω ακολουθία:

0000100000100111010010110110000011000010100000100111010010010010010111000111000001010001111000000001000101000000010101110000110111110010000101001000000010101100110100101110001110000000001000000011010100101110010010110100001011000000000111010101011010001000101100100000001111001100000000001001001101001001 1

Για να αποκωδικοποιήσουμε αυτό το μήνυμα, διαβάζουμε πέντε ψηφία τη φορά και μετατρέπουμε σύμφωνα με τη δυαδική κωδικοποίηση. Σε αυτό το παράδειγμα το c εμφανίζεται μόνο μια φορά, και απαιτεί τον ίδιο χρόνο με το l που εμφανίζεται έξι φορές.

Με τον κώδικα Huffman επιτυγχάνεται οικονομία στο χώρο, κωδικοποιώντας χαρακτήρες που χρησιμοποιούνται συχνά με όσο το δυνατό λιγότερα ψηφία και έτσι ελαχιστοποιούμε το συνολικό αριθμό ψηφίων του μηνύματος. Αρχικά μετράμε τη συχνότητα κάθε χαρακτήρα στο

μήνυμα. Το παρακάτω πρόγραμμα γεμίζει ένα πίνακα count[0…26] με τις μετρήσεις συχνότητας για ένα μήνυμα σε ένα πίνακα χαρακτήρων A[I…M].

```
For I = 0; I >=26; i++
   Count [i] ← 0;
For I = 1; I >=M; i++
   Count [index (A[i])] ←Count [index (A[i])] + 1;
```

Ο πίνακας που δημιουργείται φαίνεται παρακάτω:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 11 3 3 1 2 5 1 2 0 6 0 0 2 4 5 3 1 0 2 4 3 2 0 0 0 0 0

Στη συνέχεια πρέπει να δημιουργήσουμε ένα δένδρο κωδικοποίησης σύμφωνα με τις συχνότητες. Δημιουργούμε ένα κόμβο από τα φύλλα προς τη ρίζα του δένδρου για κάθε μη μηδενική συχνότητα. Παίρνουμε μετά δύο κόμβους με τις μικρότερες συχνότητες και δημιουργούμε ένα νέο κόμβο με αυτούς τους δύο σαν γιούς και με τιμή συχνότητας το άθροισμα των τιμών των συχνοτήτων των γιών, ενώ συνεχίζοντας με αυτό τον τρόπο φτιάχνουμε όλο και μεγαλύτερα υπο-δένδρα. Τελικά όλοι οι κόμβοι συνδυάζονται σε ένα μόνο δένδρο.

Οι κόμβοι με τις χαμηλότερες συχνότητες καταλήγουν χαμηλά στο δένδρο με αυτό τον τρόπο, ενώ οι κόμβοι με υψηλές συχνότητες καταλήγουν κοντά στη ρίζα του δένδρου. Πάνω σε κάθε κόμβο καταγράφονται και οι μετρητές συχνότητας, ενώ οι αναγραφόμενοι αριθμοί σε κάθε κόμβο είναι το άθροισμα των επιγραφών των δύο γιών του.

Ουσιαστικά, είναι πιο αποδοτικό να χρησιμοποιήσουμε κώδικα μεταβλητού μήκους για την κωδικοποίηση μας καθώς τα ψηφία ελαχιστοποιούνται (που σημαίνει περισσότερος ελεύθερος χώρος στη μνήμη) και αποκωδικοποιούμε χρησιμοποιώντας τα δυαδικά δένδρα καθώς ένας βέλτιστος κώδικας για ένα αρχείο πάντα αναπαριστάται από ένα πλήρες δυαδικό δένδρο του οποίου κάθε εσωτερικός κόμβος έχει ακριβώς δύο παιδιά.

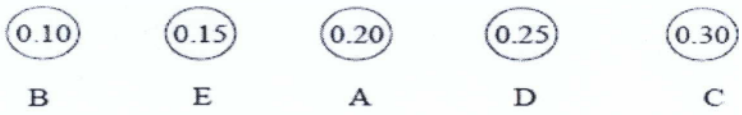
Το δένδρο αναπαριστά ένα βέλτιστο, χωρίς πρόθεμα κώδικα (prefix – free) για ένα σύνολο από C χαρακτήρες με ακριβώς |C| φύλλα και |C| - 1 εσωτερικούς κόμβους. Οι κώδικες στους οποίους κανένας κωδικός δεν αποτελεί πρόθεμα κάποιου άλλου ονομάζονται απροθηματικοί.

Χρησιμοποιώντας ένα δυαδικό δένδρο ο χρόνος χειρότερης περίπτωσης υλοποίησης του κώδικα Huffman είναι $O(n^2)$ ενώ ο χρόνος καλύτερης περίπτωσης επιτυγχάνεται με ένα πλήρες δυαδικό δένδρο και είναι $O(n\log_2 n)$.

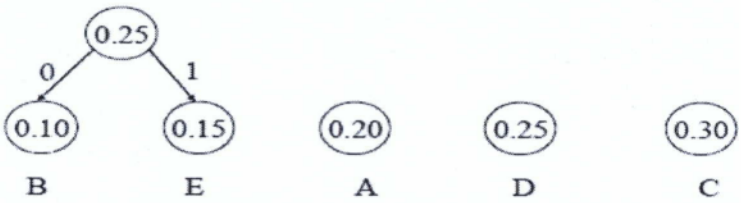
Για να παρατηρήσουμε καλύτερα την υλοποίηση αυτή ακολουθεί το παρακάτω παράδειγμα, όπου παρουσιάζονται κάποιο χαρακτήρες με τις συχνότητες τους και ακολουθεί η δημιουργία του δένδρου. Έστω τα στοιχεία:

ΧΑΡΑΚΤΗΡΑΣ	A	B	C	D	E
ΣΥΧΝΟΤΗΤΑ	0.2	0.1	0.3	0.25	0.25

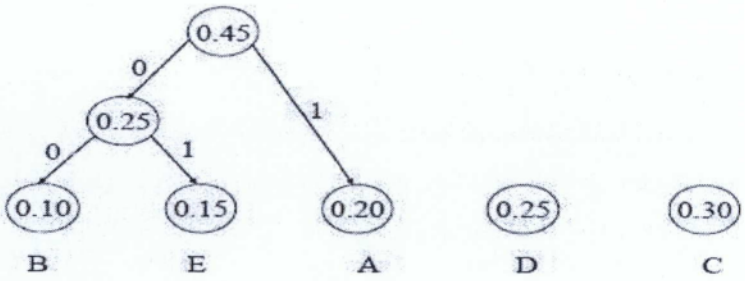
1)



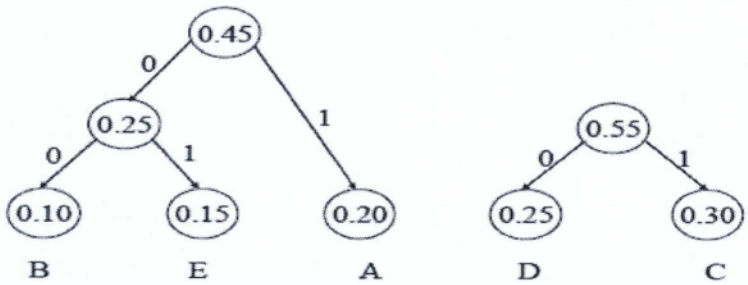
2)

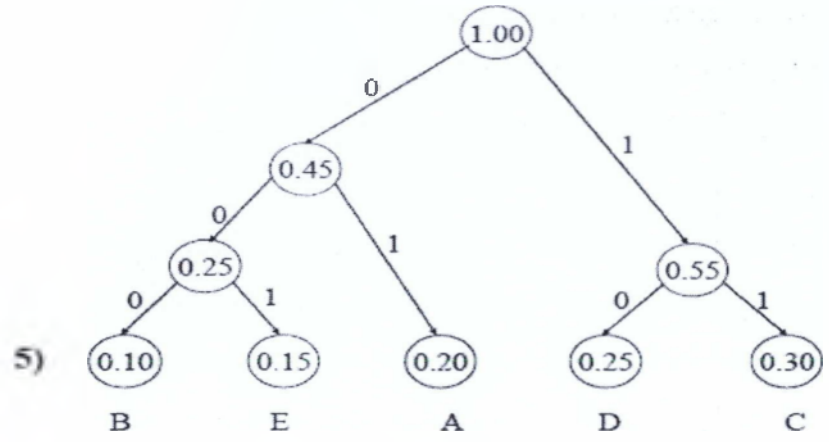


3)



4)





Σχήμα 4.4.1 Σχεδίαση δένδρου Huffman

Ο αλγόριθμος του Huffman υπολογίζει το δέντρο T που αντιστοιχεί σε έναν βέλτιστο κώδικα προχωρώντας από τα φύλλα προς τη ρίζα. Ο αλγόριθμος ξεκινά από ένα σύνολο C απομονωμένων φύλλων, τα οποία ενώνει σε ένα πλήρες δυαδικό με την εισαγωγή |C|-1 εσωτερικών κόμβων. Σε κάθε εσωτερικό κόμβο αντιστοιχούμε συχνότητα ίση με το άθροισμα των συχνοτήτων όλων των χαρακτήρων/φύλλων του υπο-δένδρου. Σε κάθε βήμα, τα δύο υπο-δένδρα με τις μικρότερες συχνότητες ενώνονται και αντικαθιστώνται από ένα υπο-δένδρο που δημιουργείται με την προσθήκη ενός νέου εσωτερικού κόμβου. Στο νέο κόμβο αντιστοιχούμε συχνότητα ίση με το άθροισμα των συχνοτήτων των ριζών των δύο υποδένδρων. Ο αλγόριθμος ολοκληρώνεται όταν απομείνει μόνο ένα δέντρο που περιέχει όλους τους χαρακτήρες του C σαν φύλλα.

Ο αλγόριθμος του Huffman μπορεί να υλοποιηθεί χρησιμοποιώντας μία ουρά προτεραιότητας (priority queue) Q, το κορυφαίο στοιχείο της οποίας αντιστοιχεί πάντα στο υπο-δένδρο με τη μικρότερη συχνότητα. Για παράδειγμα, η δομή heap υλοποιεί μια ουρά προτεραιότητας n στοιχείων με χρόνο O(log n) για εισαγωγή και διαγραφή ενός στοιχείου, και χρόνο υπολογισμού του στοιχείου με τη μικρότερη συχνότητα O(1). Αφού ο αλγόριθμος Huffman εκτελεί |C|– 1 επαναλήψεις, κάθε μία από τις οποίες απαιτεί χρόνο O(log |C|), ο χρόνος εκτέλεσης του αλγορίθμου είναι O(|C|log |C|).

Περιγραφή αλγορίθμου Huffman(C)

```
n = |C|
Q = CreatePriorityQueue (C)
for (i =1; i >= n – 1; i++)
{
    z = CreateNode ();
```

```
        left [z] = x ← Min (Q);
        DeleteMin (Q);
        right [z] = y ← Min (Q);
        DeleteMin (Q);
        f[z] = f[x] + f[y];
        Insert (Q, z);
    }
    Return Min (Q);
```

Η πορεία του αλγορίθμου για το παραπάνω παράδειγμα φαίνεται στο σχήμα 4.4.1. Δεδομένου ότι χρησιμοποιούμε 5 χαρακτήρες το αρχικό μέγεθος της ουράς είναι n = 5, και η κατασκευή του δένδρου απαιτεί τέσσερις συγχωνεύσεις. Το τελικό δένδρο αντιπροσωπεύει τον βέλτιστο απροθηματικό κώδικα. Ο κωδικός ενός χαρακτήρα συνίσταται στην ακολουθία των επιγραφών των ακμών στη διαδρομή από τη ρίζα του δένδρου μέχρι τον χαρακτήρα.

Για να λύσουμε τον χρόνο εκτέλεσης του αλγορίθμου Huffman υποθέτουμε ότι η ουρά Q υλοποιείται ως δυαδικός σωρός ελαχίστου. Για ένα σύνολο C με n χαρακτήρες , η απόδοση αρχικού περιεχομένου στη ουρά στη γραμμή 2 μπορεί να πραγματοποιηθεί σε χρόνο O(n) μέσω της διαδικασίας Κατασκευή σωρού ελαχίστου. Ο βρόχος για τις γραμμές 3 – 8 εκτελείται ακριβώς n – 1 φορές, δεδομένου ότι κάθε πράξη σωρού απαιτεί χρόνο O(logn), ο βρόχος συνεισφέρει στο χρόνο εκτέλεσης μια ποσότητα O(nlogn). Επομένως, ο συνολικός χρόνος εκτέλεσης της διαδικασίας Huffman για ένα σύνολο χαρακτήρων είναι O(nlogn).

Επίλογος

Ανακεφαλαιώνοντας μπορούμε να ισχυριστούμε πως ο στόχος που είχα θέσει για την εκπόνηση αυτής της εργασίας έχει σε γενικές γραμμές επιτευχθεί. Πραγματοποιήθηκε μια σφαιρική έρευνα πάνω στα δένδρα και ειδικότερα τα δυαδικά δένδρα αναζήτησης, της λειτουργίες που γίνονται πάνω σε αυτά και τις εφαρμογές τους. Ακόμη μελετήθηκαν διάφορα είδη δυαδικών δένδρων αναζήτησης όπως τα δένδρα AVL και τα ερυθρόμαυρα δένδρα, καθώς και οι λειτουργίες αυτών.

Στόχος μου ήταν να ανακαλύψω και να παρουσιάσω τη χρησιμότητα των δυαδικών δένδρων και ειδικότερα των δυαδικών δένδρων αναζήτησης και θα ήθελα να επισημάνω πως οι δομές δεδομένων προσφέρουν :

- Εύκολα κατανοητή αναπαράσταση της απεικόνισής τους με τα σχεδιαγράμματα δενδροειδούς μορφής.
- Γρήγορη, εύκολη και αποδοτική αναζήτηση στοιχείων μέσα σε ένα σύνολο.
- Εισαγωγές και διαγραφές δεδομένων μιας ακολουθίας στοιχείων με πολύ καλούς χρόνους εκτέλεσης.

Όσον αφορά τις εφαρμογές των δυαδικών δένδρων αναζήτησης θα ήθελα να αναφέρω σαν παράδειγμα της σημασίας τους και της χρησιμότητας τους τα δένδρα Huffman. Τα δένδρα αυτά στηριζόμενα πάνω στα δυαδικά δένδρα αναζήτησης, έχουν ανοίξει νέους ορίζοντες στην επιστήμη των υπολογιστών και της Πληροφορικής με τη συγκεκριμένη μέθοδο συμπίεσης του χώρου που καταλαμβάνουν στη μνήμη του υπολογιστή τα διάφορα δεδομένα. Για τους παραπάνω λόγους είναι απαραίτητη η χρήση των δυαδικών δένδρων αναζήτησης και η δημιουργία νέων εφαρμογών τους.

Βιβλιογραφία

1. THOMAS H. CORMEN, CHARLES E. LEISON, RONALD L. RIVEST, CLIFFORD STEIN «Εισαγωγή στους αλγόριθμους».

2. ΓΙΩΡΓΟΣ ΠΑΠΑΓΕΩΡΓΙΟΥ, ΤΙΜΟΣ ΑΣΛΑΝΙΔΗΣ, ΦΩΤΩ ΑΦΡΑΤΗ «Αλγόριθμοι: Μέθοδοι σχεδίασης και ανάλυση πολυπλοκότητας».

3. ΙΩΑΝΝΗΣ ΠΑΠΟΥΤΣΗΣ «Εισαγωγή στις Δομές Δεδομένων και στους αλγόριθμους».

4. Δ. ΦΩΤΑΚΗΣ, διαφάνειες εργαστηρίων.

5. Δ. ΦΩΤΑΚΗΣ «Δένδρα Αναζήτησης».

6. Κ. ΕΜΜΑΝΟΥΗΛΙΔΗΣ «Βαθμοζυγισμένα Δυαδικά Δένδρα Αναζήτησης».

7. ΣΑΒΒΑΣ ΗΛΙΑΣ «Αλγόριθμοι και πολυπλοκότητα».

8. ΚΩΣΤΑΣ ΣΤΕΡΓΙΟΥ «Δένδρα Αναζήτησης».

9. ΔΗΜΗΤΡΗΣ ΜΙΧΑΗΛ «Δομές δεδομένων: Αναδρομή και δένδρα».