Κεφάλαιο 1

Δομές Δεδομένων

1.1 Στοιχειώδεις Αφηρημένες Δομές

1.1.1 Διακριτό Σύνολο

Το **διακριτό σύνολο** $\mathcal{S}$ (discrete set) είναι μία συλλογή n διακριτών (distinct) στοιχείων $\{s_1, s_2, \ldots, s_n\}$ χωρίς κάποια συγκεκριμένη διάταξη αυτών των στοιχείων: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$.

Για το διακριτό σύνολο ορίζονται οι στοιχειώδεις τελεστές:

πληθύς (cardinality) του συνόλου που μαθηματικά συμβολίζεται ως $|\mathcal{S}|$ και αποτελεί το μέγεθος του συνόλου. Έτσι $|\mathcal{S}| = n$. Εάν όμως $|\mathcal{S}| = 0$, τότε το σύνολο $\mathcal{S}$ είναι το κενό σύνολο: $\mathcal{S} = \emptyset$.

είναι μέλος που μαθηματικά αντιστοιχεί στο σύμβολο $\in$ εφόσον αληθεύει και στο σύμβολο $\notin$ εφόσον δεν αληθεύει. Έτσι γράφουμε $s \in \mathcal{S}$ εάν $s = s_k$ για κάποιο $1 \leq k \leq n$ και $s \notin \mathcal{S}$ για κάθε άλλη περίπτωση.

απαλοιφή στοιχείου που μαθηματικά αντιστοιχεί στην πράξη $\mathcal{S} = \mathcal{S} \backslash \{s\}$.

προσθήκη στοιχείου που μαθηματικά αντιστοιχεί στην πράξη $\mathcal{S} = \mathcal{S} \cup \{s\}$.

Για ένα τέτοιο σύνολο $\mathcal{S}$ είναι δυνατόν να ορισθεί μία συνάρτηση που το χαρακτηρίζει πλήρως, η **χαρακτηριστική συνάρτηση** $f_S(\cdot)$ επί ενός καθολικού συνόλου $\mathcal{U}$ με $\mathcal{S} \subseteq \mathcal{U}$:

$$f_S(x) = \begin{cases} \textbf{true} \text{ εάν } x \in \mathcal{S} \\ \textbf{false} \text{ εάν } x \in \mathcal{U} \backslash \mathcal{S}. \end{cases}$$

Για τα διακριτά σύνολα είναι δυνατόν ν' ορισθούν τελεστές για την **ένωση** δύο συνόλων $\mathcal{S} \cup \mathcal{T}$, την **τομή** $\mathcal{S} \cap \mathcal{T}$, την **διαφορά** δύο συνόλων $\mathcal{S} \backslash \mathcal{T}$, η οποία είναι ένα νέο σύνολο απ' όλα τα στοιχεία του $\mathcal{S}$ που δεν είναι και στοιχεία του $\mathcal{T}$ και, τέλος, την **συμμετρική διαφορά** $\mathcal{S} \oplus \mathcal{T} = (\mathcal{S} \backslash \mathcal{T}) \cup (\mathcal{T} \backslash \mathcal{S})$, η οποία είναι ένα νέο σύνολο απ' όλα τα στοιχεία του $\mathcal{S}$ που δεν είναι και στοιχεία του $\mathcal{T}$ αλλά και απ' όλα τα στοιχεία του $\mathcal{T}$ που δεν είναι και στοιχεία του $\mathcal{S}$.

1.1.2 Απεικόνιση

Η **απεικόνιση** (map/mapping/associative store) $\mathcal{F} = \{[x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n]\}$ είναι ένα σύνολο διατεταγμένων ζευγών $[x_k, y_k]$, $(1 \leq k \leq n)$, για τα οποία ισχύει ότι $x_\ell \neq x_k$ για $\ell \neq k$. Τα $x_1, x_2, \ldots, x_n$ είναι δηλαδή διακριτά. Η απεικόνιση $\mathcal{F}$ μπορεί να συγκριθεί με το γράφημα/καμπύλη της συνάρτησης $f : x \in \{x_1, x_2, \ldots, x_n\} \rightarrow y = f(x) \in \{y_1, y_2, \ldots, y_n\}$.

Η απεικόνιση χαρακτηρίζεται πλήρως από δύο πεδία:

πεδίο ορισμού (domain) της $\mathcal{F}$ είναι το σύνολο $\mathcal{D}(\mathcal{F}) = \mathcal{D}(f) = \{x_1, x_2, \ldots, x_n\}$.

πεδίο τιμών (value range) της $\mathcal{F}$ είναι το σύνολο $\mathcal{R}(\mathcal{F}) = \mathcal{R}(f) = \{y_1, y_2, \ldots, y_n\}$.
Για $x \notin \mathcal{D}(\mathcal{F})$ η τιμή $f(x)$ δεν είναι ορισμένη και θα χρησιμοποιήσουμε το σύμβολο *null* ως συμβολισμό αυτού του γεγονότος. Έτσι $x \notin \mathcal{D}(\mathcal{F}) \Rightarrow f(x) = null$.

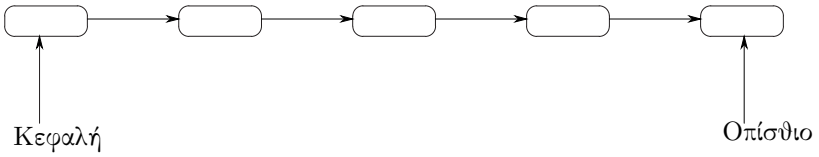
Δύο στοιχειώδεις τελεστές για την απεικόνιση είναι:

ανάθεση (assignment) που καταχωρεί το ζεύγος $[x, f(x)]$ στην απεικόνιση $\mathcal{F}$ με $f(x) = y$. Εάν η απεικόνιση περιέχει ήδη κάποιο ζεύγος $[x, y]$ αυτό αντικαθίσταται από το νέο ζεύγος $[x, f(x)]$. Η ανάθεση $f(x) = null$ διαγράφει το ζεύγος $[x, y]$, εφόσον αυτό υπάρχει, από την απεικόνιση $\mathcal{F}$.

πληθύς της απεικόνισης $\mathcal{F}$ είναι το μέγεθός της $|\mathcal{F}|$.

1.1.3 Κατάλογος ή Γραμμική Λίστα

Ένας κατάλογος ή γραμμική λίστα είναι μία διατεταγμένη ακολουθία στοιχείων $\mathcal{L} = [x_1, x_2, \ldots, x_n]$. Τα στοιχεία του καταλόγου αναφέρονται και ως κόμβοι. Οι κόμβοι δεν απαιτείται να είναι διακριτοί. Υπάρχει δηλαδή το ενδεχόμενο να επαναλαμβάνονται. Η διάταξή τους όμως στα πλαίσια του καταλόγου είναι δομικό χαρακτηριστικό του καταλόγου αφού καθορίζει τις γραμμικές σχετικές θέσεις των στοιχείων του. Εφόσον $|\mathcal{L}| = n > 0$, το πρώτο στοιχείο, x_1 , της λίστας ονομάζεται **μέτωπο** ή **κεφαλή** (front/head), ενώ το τελευταίο της στοιχείο x_n ονομάζεται **οπίσθιο** ή **πυθμένας** (rear/tail/bottom). Το kστό στοιχείο της λίστας είναι το x_k, $1 \leq k \leq n$. Μπορούμε ν' απεικονίσουμε (Σχήμα 1.1) την λίστα ως ένα προσανατολισμένο γράφημα που αποτελεί μία απλή συμβατή διαδρομή Hamilton.



Σχήμα 1.1: Γραμμική λίστα

Στοιχειώδεις τελεστές για τους καταλόγους είναι οι ακόλουθοι:

πληθύς της λίστας $\mathcal{L}$ είναι το μέγεθός της $|\mathcal{L}| = n$. Εάν $n = 0$, η λίστα είναι κενή.

πρόσβαση στην κεφαλή επιτρέπει την ανάγνωση ή την αλλαγή περιεχομένου στην κεφαλή $x_1 = \mathcal{L}(1)$ της λίστας.

πρόσβαση στον πυθμένα επιτρέπει την ανάγνωση ή την αλλαγή περιεχομένου στον πυθμένα $x_n = \mathcal{L}(|\mathcal{L}|)$ της λίστας.

ένθεση στην κεφαλή ενός στοιχείου x συνεπάγεται ότι η λίστα $\mathcal{L}$ αντικαθίσταται από την $[x, \mathcal{L}]$.

απαλοιφή κεφαλής συνεπάγεται ότι η λίστα $\mathcal{L}$ αντικαθίσταται από την $[x_2, x_3, \ldots, x_n]$.

ένθεση στον πυθμένα της λίστας ενός στοιχείου x συνεπάγεται ότι η λίστα $\mathcal{L}$ αντικαθίσταται από την $[\mathcal{L}, x]$.

απαλοιφή πυθμένα συνεπάγεται ότι η λίστα $\mathcal{L}$ αντικαθίσταται από την $[x_1, x_2, \ldots, x_{|\mathcal{L}|-1}]$.

Πλην των ανωτέρων, είναι δυνατόν να ορισθούν πρόσθετοι τελεστές όπως:

πρόσβαση στο kστό κόμβο της λίστας για ανάγνωση ή αλλαγή του περιεχομένου του.

απαλοιφή του kστού κόμβου της λίστας συνεπάγεται ότι η $\mathcal{L}$ αντικαθίσταται από την λίστα $[x_1, x_2, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n]$.

ένθεση πρό του kστού κόμβου ενός νέου στοιχείου x συνεπάγεται ότι η $\mathcal{L}$ αντικαθίσταται από την λίστα $[x_1, x_2, \ldots, x_{k-1}, x, x_k, \ldots, x_n]$. Η ένθεση νέου στοιχείου μετά τον κόμβο k είναι προφανώς ένθεση πρό του $k+1$στού κόμβου.

υπολίστα (sublist) ορίζεται για δύο ακέραιους i και k και συνεπάγεται την δημιουργία της λίστας $\mathcal{L}[i..k] = [x_i, x_{i+1}, \ldots, x_k]$ από στοιχεία της λίστας $\mathcal{L}$. Σε περίπτωση που $k > |\mathcal{L}|$, το k κολοβώνεται στην πληθύ της λίστας $\mathcal{L}$. Σε περίπτωση που $i < 1$, η υπολίστα ορίζεται με αρχή την κεφαλή της $\mathcal{L}$.

διάσπαση (split) σημαίνει ότι η λίστα $\mathcal{L}$ τεμαχίζεται σε δύο ή περισσότερες υπολίστες.

σύνδεση αλληλουχίας (concatenation) δύο καταλόγων $\mathcal{L}_1$ και $\mathcal{L}_2$ συνεπάγεται την δημιουργία μιας νέας λίστας $\mathcal{L} = \mathcal{L}_1//\mathcal{L}_2 = [\mathcal{L}_1, \mathcal{L}_2]$. Επειδή η γραμμική διάταξη των στοιχείων μιας λίστας είναι το βασικό της χαρακτηριστικό, έχομε ότι $\mathcal{L}_1//\mathcal{L}_2 \neq \mathcal{L}_2//\mathcal{L}_1$.

αντιγραφή σημαίνει την δημιουργία αντιγράφου μιας γραμμικής λίστας.

αναζήτηση τιμής σημαίνει την εξέταση των κόμβων της λίστας για την ανεύρεση ενός κόμβου με συγκεκριμένο περιεχόμενο.

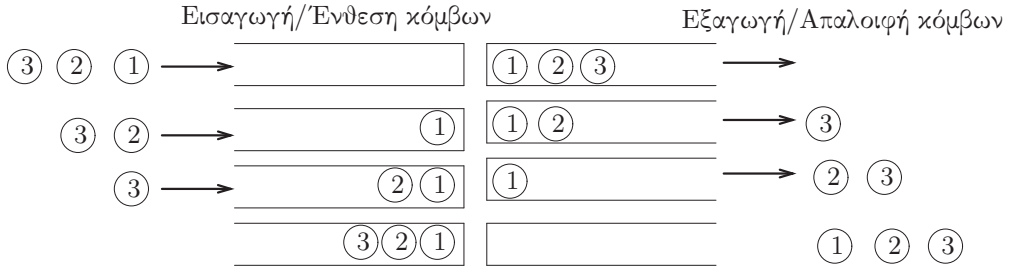
μονότονη διάταξη τιμών (sort) σημαίνει αναδιάταξη των κόμβων της λίστας σε αύξουσα ή φθίνουσα διάταξη τιμών που περιέχονται στους κόμβους.

1.1.4 Στοίβα και Ουρά

Η **στοίβα** (stack) και η **ουρά** (queue) είναι ειδικές περιπτώσεις γραμμικών καταλόγων με τους στοιχειώδεις τελεστές ένθεσης, απαλοιφής και πρόσβασης περιορισμένους στην κεφαλή ή στον πυθμένα. Πιο συγκεκριμένα,

- Η στοίβα είναι μία γραμμική λίστα εφοδιασμένη με τους στοιχειώδεις τελεστές: **πρόσβαση στην κεφαλή**, **ένθεση στην κεφαλή**, **απαλοιφή κεφαλής** και **πληθύς**.
- Η ουρά είναι μία γραμμική λίστα εφοδιασμένη με τους στοιχειώδεις τελεστές: **πρόσβαση στην κεφαλή**, **απαλοιφή της κεφαλής**, **ένθεση στον πυθμένα** και **πληθύς**.

Μπορούμε ν' αντιληφθούμε την στοίβα ως ένα σωληνάριο με κλειστό πυθμένα (Σχήμα 1.2). Η ένθεση νέων στοιχείων στην στοίβα αντιστοιχεί σε εισαγωγή σφαιριδίων από το ανοικτό στόμιο του σωληναρίου. Η απαλοιφή στοιχείων από την στοίβα αντιστοιχεί στην εξαγωγή σφαιριδίων από το ανοικτό στόμιο του σωληναρίου. Το σφαιρίδιο που πρώτο εισήχθη στο σωληνάριο είναι αυτό που εξάγεται τελευταίο. Π.χ. στο Σχήμα 1.2, το σφαιρίδιο 1 εισήλθε πριν από το 2, το οποίο εισήλθε πριν από το 3. Κατά την εξαγωγή τους όμως από το σωληνάριο, το 3 εξέρχεται πρίν από το 2, ενώ το 2 εξέρχεται πριν από το 1. *Η στοίβα επιβάλλει δηλαδή μία διάταξη εξαγωγής που είναι αντίστροφη από την διάταξη εισαγωγής.* Λέμε ότι η στοίβα επιβάλλει την διάταξη «τελευταίος μέσα, πρώτος έξω» που στην διεθνή βιβλιογραφία είναι γνωστή ως διάταξη LIFO (Last In First Out).



Σχήμα 1.2: Στοίβα

Σε αναλογία με τη στοίβα, η ουρά θα μπορούσε να γίνει αντιληπτή ως ένα σωληνάριο ανοικτό και στα δύο του άκρα (Σχήμα 1.3). Το ένα ανοικτό στόμιο χρησιμοποιείται για την εισαγωγή σφαιριδίων, ενώ το άλλο για την εξαγωγή τους. Το στόμιο εισαγωγής αντιστοιχεί στον πυθμένα του σωληναρίου, ενώ το στόμιο εξαγωγής στην κεφαλή. Το σφαιρίδιο που πρώτο εισήχθη στο σωληνάριο είναι αυτό που εξάγεται πρώτο. Π.χ. στο Σχήμα 1.3 το σφαιρίδιο 1 εισήλθε πριν από το 2, το οποίο εισήλθε πριν από το 3. Κατά την εξαγωγή τους, το σφαιρίδιο 1 εξέρχεται πριν από το 2, το οποίο εξέρχεται πριν από το 3. *Η ουρά διατηρεί δηλαδή την διάταξη εισαγωγής ως διάταξη εξαγωγής.* Λέμε ότι η ουρά επιβάλλει την διάταξη «πρώτος μέσα, πρώτος έξω» που στην διεθνή βιβλιογραφία είναι γνωστή ως διάταξη FIFO (First In First

Σχήμα 1.3: Ουρά

Out). Η διάταξη αυτή εμφανίζεται στην καθημερινότητα π.χ. κατά την αναμονή μας για εξυπηρέτηση σε ταχυδρομείο ή στην τράπεζα.

Υπάρχουν δύο ακόμη είδη ουρών. Η **διπλή ουρά** (double-ended queue, deque) είναι μία γραμμική λίστα για την οποία ορίζονται και οι επτά στοιχειώδεις τελεστές των καταλόγων: πληθύς, πρόσβαση στην κεφαλή, πρόσβαση στον πυθμένα, ένθεση στην κεφαλή, απαλοιφή κεφαλής, ένθεση στον πυθμένα, απαλοιφή πειθμένα. Μπορεί συνεπώς να προσομοιωθεί από σωληνάριο με δύο ανοικτά άκρα όπου η εισαγωγή και εξαγωγή σφαιριδίων γίνεται και από τα δύο άκρα. Η διπλή ουρά μπορεί να λειτουργήσει τόσο ως στοίβα όσο και ως απλή ουρά. Υπάρχει επίσης η περίπτωση ν' απαγορευθεί η εξαγωγή ή η εισαγωγή σ' ένα από τα δύο άκρα του σωληναρίου, οπότε έχουμε διπλή ουρά με περιορισμένη απαλοιφή ή ένθεση.

Η **ουρά προτεραιότητας** είναι μία τελείως διαφορετική οντότητα. Στην καθημερινότητά μας την απαντούμε π.χ. στις ιατρικές επισκέψεις, όπου αναξάρτητα από το χρόνο προσέλευσης των ασθενών, η σειρά με την οποία εξετάζονται καθορίζεται από την προκαθορισμένη ώρα ραντευού τους. Έτσι ο ασθενής A με ραντεβού στις 10 μπορεί να προσήλθε χρονικά πριν από τον ασθενή B με ραντεβού στις 9 αλλά σε αντίθεση με την απλή ουρά, ο ασθενής B έχει προτεραιότητα και εξετάζεται πριν από τον A. Μπορούμε να προσομοιώσουμε μία ουρά προτεραιότητας με ουρά εάν προσάψουμε στην τελευταία και τελεστή αναζήτησης τιμής ή/και μονότονης διάταξης τιμών: Θα υποθέσουμε ότι σε κάθε x, η συνάρτηση $p(\cdot)$ αντιστοιχίζει μία ακέραια τιμή $p(x)$ που καθορίζει την προτεραιότητα του x και ότι τα στοιχεία της ουράς $\mathcal{Q} = [x_1, x_2, \ldots, x_n]$ με κεφαλή το x_1 και πυθμένα το x_n είναι σε φθίνουσα διάταξη προτεραιότητας, δηλαδή $p(x_1) \geq p(x_2) \geq \cdots \geq p(x_n)$. Κατά την ένθεση του x στο $\mathcal{Q}$, η ένθεση γίνεται μεν στον πυθμένα, όμως η τιμή $p(x)$ σε σχέση με τις αντίστοιχες τιμές $p(x_k)$ των στοιχείων x_k της ουράς $\mathcal{Q}$ είναι αυτή που καθορίζει την τελική θέση του x στο $\mathcal{Q}$. Έτσι, εάν $p(x_i) > p(x) > p(x_{i+1})$, τότε το x καταλαμβάνει την θέση $i+1$ στο $\mathcal{Q}$, ενώ το x_{i+1} μετατοπίζεται προς τα δεξιά μαζί μ' όλα τα στοιχεία που έπονται και η πληθύς του $\mathcal{Q}$ αυξάνει κατά μία μονάδα. Εάν $p(x_n) > p(x)$, το x καθίσταται το νέο στοιχείο του πυθμένα, ενώ εάν $p(x) > p(x_1)$, τότε το x καθίσταται η νέα κεφαλή του $\mathcal{Q}$. Προφανώς η ουρά προτεραιότητας δεν ανταποκρίνεται στην έννοια της γραμμικής λίστας. Ο σωρός (heap) είναι μία άλλη αφηρημένη δομή δεδομένων που ανταποκρίνεται καλύτερα στην έννοια της ουράς προτεραιότητας. Εξετάζουμε τον σωρό χωριστά.

1.2 Υλοποίηση Στοιχειωδών Δομών

1.2.1 Υλοποίηση Διακριτού Συνόλου

1.2.1.1 Υλοποίηση με απλή παράταξη

Εάν το σύνολο $\mathcal{S}$ είναι υποσύνολο του καθολικού συνόλου $\mathcal{U} = \{1, 2, \ldots, n\}$, τότε μπορούμε να χρησιμοποιήσουμε ένα $n-$διάνυσμα **S** με πεδίο τιμών $\{\mathbf{true}, \mathbf{false}\}^n$ για την αναπαράσταση του συνόλου μέσω της χαρακτηριστικής του συνάρτησης. Η χαρακτηριστική συνάρτηση μέσω μιας τέτοιας υλοποίησης επιτρέπει να ελεγχθεί εάν $s \in \mathcal{S}$ ή εάν $s \notin \mathcal{S}$ σε χρόνο $O(1)$ και μπορεί επίσης να ενημερωθεί σε χρόνο $O(1)$ κατά την απαλοιφή ή προσθήκη στοιχείου. Η πληθύς του συνόλου είναι ένας ακέραιος ο οποίος ενημερώνεται κατά την προσθήκη ή απαλοιφή στοιχείου σε χρόνο $O(1)$ και η τιμή του ελέγχεται επίσης σε χρόνο $O(1)$.

Αρχικά το σύνολο είναι κενό και η πληθύς του μηδενική.

$$\boxed{\begin{array}{l}\textbf{logical ::}\ S(1:n) \ = \ \textbf{false} \\ \textbf{integer ::}\ cardinality \ = \ 0\end{array}}$$

Η προσθήκη ενός στοιχείου $x \in \mathcal{U}$ στο $\mathcal{S}$ αντιστοιχεί στις αναθέσεις

$$\boxed{S(x) \ = \ \textbf{true};\ cardinality \ = \ cardinality \ + \ 1}$$

και είναι επομένως πολυπλοκότητας $O(1)$.

Η απαλοιφή ενός στοιχείου x από το $\mathcal{S}$ αντιστοιχεί στις αναθέσεις

$$\boxed{S(x) \ = \ \textbf{false};\ cardinality \ = \ cardinality \ - \ 1}$$

και άρα είναι πολυπλοκότητας $O(1)$.

Ο έλεγχος για το εάν ένα στοιχείο $x \in \mathcal{U}$ είναι ή όχι μέλος του $\mathcal{S}$ αντιστοιχεί στην ανάθεση

$$\boxed{is\_member \ = \ S(x)}$$

με απαίτηση χρόνου $O(1)$.

Η προηγούμενη υλοποίηση μπορεί βεβαίως να μεταφερθεί σε διάνυσμα n δυφίων (bits), όπου n η πληθύς του καθολικού συνόλου $\mathcal{U}$. Πράξεις επί των συνόλων, όπως η ένωση ή η τομή δύο συνόλων δύνανται να επιτευχθούν με τους τελεστές «λογικού και» και «λογικού είτε». Μια τέτοια υλοποίηση είναι ιδιαίτερα αποδοτική εάν το n είναι μικρό καθώς κάθε πράξη αντιστοιχεί τότε σε μία εντολή μηχανής. Εάν όμως το n είναι μεγάλο, π.χ. μεγαλύτερο από το πλήθος δυφίων σε μία λέξη (word), ενώ η πληθύς του κάθε συνόλου μικρή σε σύγκριση με το n, τότε η υλοποίηση δεν είναι αποδοτική διότι οι πράξεις της ένωσης και της τομής απαιτούν χρόνο ανάλογο του n και όχι ανάλογο της πληθύος των επιμέρους συνόλων. Οι πράξεις που μπορούν τα τελεστούν στα δυφία δίδονται στον Πίνακα 1.1 ενώ οι λεπτομερής υλοποίηση επαφίεται στον αναγνώστη. Στην § 1.3.1 θα εξετάσουμε με λεπτομέρεια μία διαφορετική υλοποίηση των διακριτών συνόλων.

Πίνακας 1.1: Τελεστές Δυφίων

Fortran	C++	Επεξήγηση
iand	&	«Λογικό και» ανά δυφίο
ieor	^	«Λογικό αποκλειστικό είτε» ανά δυφίο
ior	|	«Λογικό συμπεριληπτικό είτε» ανά δυφίο
not	∼	Λογικό συμπλήρωμα
ibclr		Εκκαθάριση διφύου
ibits		Απόσπαση διφύου
btest		Δοκιμή τιμής δυφίου
ibset		Ανάθεση τιμής σε δυφίο
ishft	<<	Ολίσθηση προς τ' αριστερά
ishft	>>	Ολίσθηση προς τα δεξιά
ishftc		Κυκλική ολίσθηση διφύων
bit_size	**sizeof**	Πλήθος δυφίων ενός ακεραίου
mvbits		Συνδυασμός διφύων και ανάθεση σε ακέραιο
transfer		Μεταφορά διφύων σε ακέραιο

1.2.2 Υλοποίηση Απεικόνισης

1.2.2.1 Υλοποίηση με απλή παράταξη

Εάν το πεδίο ορισμού $\mathcal{D}(\mathcal{F})$ της απεικόνισης $\mathcal{F}$ είναι ένα ακέραιο εύρος, π.χ. $[1..n]$, τότε η απεικόνιση δύναται να παρασταθεί ως ένα $n-$διάνυσμα **f** με τύπο δεδομένων αυτόν του πεδίου τιμών $\mathcal{R}(\mathcal{F})$ της $\mathcal{F}$. Π.χ. εάν το πεδίο τιμών είναι οι πραγματικοί αριθμοί, τότε η απεικόνιση μπορεί να δηλωθεί ως

$$\boxed{\textbf{real ::}\ f(1:n)\ =\ null}$$

όπου ως *null* έχει ορισθεί κάποια σταθερά του ίδιου τύπου δεδομένων με το πεδίο τιμών της $\mathcal{F}$ αλλά έξω απ' αυτό το πεδίο. Στην περίπτωση των πραγματικών αριθμών που εξετάζουμε θα μπορούσαμε να έχουμε ορίσει την τιμή του *null* ως τον μέγιστο πραγματικό αριθμό της υπολογιστικής πλατφόρμας. Στην Fortran π.χ., αυτό επιτυγχάνεται ως εξής:

$$\boxed{\textbf{real}, \textbf{parameter ::}\ null = \textbf{huge}(\ 0.0\ )}$$

Όλα τα n στοιχεία του διανύσματος f λαμβάνουν την τιμή $3.4028235E+38$ που είναι ο μεγαλύτερος αριθμός τύπου **real** στο σύστημά μου (σ' άλλα συστήματα ενδέχεται να είναι διαφορετικός). Οποιαδήποτε τιμή μεγαλύτερη απ' αυτήν ισοδυναμεί σε υπερεκχείλιση.

Επιπρόσθετα, ένας ακέραιος, *cardinality*, χρησιμοποιείται για την καταγραφή της πληθύος της απεικόνισης. Ο ακέραιος έχει αρχική τιμή 0.

$$\boxed{\textbf{integer ::}\ cardinality = 0}$$

Η καταχώρηση ενός νέου ζεύγους $[x, y]$ στην απεικόνιση $\mathcal{F}$ αντιστοιχεί στις αναθέσεις

$$\boxed{f(x) \ = \ y; cardinality \ = \ cardinality \ + \ 1}$$

και είναι επομένως πολυπλοκότητας $O(1)$.

Η διαγραφή ενός ζεύγους $[x, y]$ από την απεικόνιση $\mathcal{F}$ αντιστοιχεί στις αναθέσεις

$$\boxed{f(x) \ = \ null; cardinality \ = \ cardinality \ - \ 1}$$

που εκτελούνται σε χρόνο $O(1)$.

Ο έλεγχος εάν η απεικόνιση $\mathcal{F}$ είναι ορισμένη ή όχι για κάποιο x γίνεται μέσω της ανάθεσης

$$\boxed{is\_defined \ = \ (\ f(x) \ / = \ null \ )}$$

και άρα είναι πολυπλοκότητας $O(1)$.

1.2.3 Υλοποίηση Στοίβας

1.2.3.1 Υλοποίηση με απλή παράταξη

Ας υποθέσουμε ότι η ένθεση και η απαλοιφή στοιχείων στην στοίβα αφορά σε στοιχεία ακεραίου τύπου. Τέτοια στοιχεία μπορεί ενδεχομένως να είναι ψευδοδείκτες (pseudopointers) ή αναδρομείς (cursors) που καταδεικνύουν θέσεις για κάποια άλλα αντικείμενα που δεν επιθυμούμε να μετακινούμε π.χ. λόγω του όγκου των δεδομένων τους. Θα υποθέσουμε ακόμη ότι πριν από την χρήση της στοίβας είναι δυνατόν να καθορισθεί το μέγιστο πλήθος στοιχείων, n, που θα κληθεί να χειρισθεί. Μπορούμε τότε να παραστίσουμε την στοίβα μ' ένα ακέραιο διάνυσμα:

$$\boxed{\textbf{integer}, \textbf{allocatable} :: stack(\ : \ )}$$

Όταν στην διάρκεια των υπολογισμών καταστεί γνωστό το μέγιστο απαιτούμενο μέγεθος n της στοίβας, εξασφαλείζουμε τον απαιτούμενο χώρο στην μνήμη με την εντολή

$$\boxed{\textbf{allocate}(\ stack(n) \ )}$$

Για γλώσσες προγραμματισμού που παρέχουν μόνον στατικές παρατάξεις, το διάνυσμα καταχωρείται εξ αρχής για κάποιο σταθερό n, π.χ.

$$\boxed{\textbf{integer} :: stack(100)}$$

Εκτός από το διάνυσμα θ' απαιτηθεί και μία ακέραια μεταβλητή, $last$, η οποία έχει διπλό ρόλο: Από την μία είναι ένας ψευδοδείκτης που καταδεικνύει την κεφαλή ή κορυφή της στοίβας, την θέση δηλαδή του τελευταίου στοιχείου που ενετέθει στην στοίβα, και από την άλλη καταμετρά το πλήθος των στοιχείων στην στοίβα. Αρχικά η στοίβα είναι κενή και άρα η μεταβλητή αυτή έχει μηδενική τιμή.

$$\boxed{\textbf{integer} :: last \ = \ 0}$$

Η ένθεση ενός νέου στοιχείου, $node$, στην στοίβα γίνεται πάντοτε στην κεφαλή και μπορεί ν' αποδοθεί από τις αναθέσεις

$$\boxed{last \ = \ last \ + \ 1; \ stack(last) \ = \ node}$$

που είναι πολυπλοκότητας $O(1)$.

Η πρόσβαση στο στοιχείο της κεφαλής αντιστοιχεί στην ανάθεση

$$\boxed{node \ = \ stack(last)}$$

επίσης πολυπλοκότητας $O(1)$

Τέλος η απαλοιφή του στοιχείου της κορυφής αντιστοιχεί στην ανάθεση

$$\boxed{last \ = \ last \ - \ 1}$$

με πολυπλοκότητα $O(1)$.

Για την πληθύ της στοίβας έχομε πάντοτε

$$\boxed{cardinality \ = \ last}$$

Επειδή για στατικά διανύσματα υπάρχει το ενδεχόμενο υπέρβασης του μέγιστου μεγέθους της στοίβας και επειδή είναι επιθυμητό ν' αποφευχθεί η μη ορθή προσπάθεια πρόσβασης σε στοιχείο κορυφής όταν η στοίβα είναι κενή, προστίθενται συνήθως δύο ακόμη στοιχειώδεις τελεστές για την επίτευξη αυτών των ελέγχων:

$$\boxed{is\_empty \ = \ (\ last \ == \ 0\ );\ is\_full \ = \ (\ last \ == \ \mathbf{size}(\ stack\ )\ )}$$

Και οι τρεις προηγούμενοι τελεστές είναι πολυπλοκότητας $O(1)$.

1.2.4 Υλοποίηση Ουράς

1.2.4.1 Υλοποίηση με απλή παράταξη

Θα εξετάσουμε την υλοποίηση της ουράς υπό παρόμοιες συνθήκες μ' αυτές τις στοίβας. Έτσι θα υποθέσουμε ότι τα στοιχεία είναι ακέραια. Επομένως θα παραστήσουμε την ουρά μ' ένα ακέραιο δυναμικό διάνυσμα

$$\boxed{\mathbf{integer},\ \mathbf{allocatable}\ ::\ queue(\ :\ )}$$

για το οποίο κάποια στιγμή υπολογίζουμε το μέγιστο απαιτούμενο μέγεθος και του κατανέμουμε μνήμη

$$\boxed{\mathbf{allocate}(\ queue(n)\ )}$$

ή, ελλείψει δυναμικών παρατάξεων, καταχωρούμε μνήμη εξ αρχής, π.χ.

$$\boxed{\mathbf{integer}\ ::\ queue(100)}$$

Θα χρησιμοποιηθούν και δύο ακέραιες μεταβλητές, $first$ και $last$, οι οποίες αποτελούν ψευδοδείκτες που καταδεικνύου τις θέσεις των στοιχείων της κεφαλής και του πυθμένα της ουράς στο διάνυσμα $queue$. Αρχικά η ουρά είναι κενή και άρα οι μεταβλητές αυτές έχουν μηδενικές τιμές.

$$\boxed{\mathbf{integer}\ ::\ first \ = \ 0, last \ = \ 0}$$

Η ένθεση ενός νέου στοιχείου, $node$, στην ουρά γίνεται πάντοτε στον πυθμένα και μπορεί ν' αποδοθεί από τις αναθέσεις

$$\boxed{last \ = \ last \ + \ 1;\ queue(last) \ = \ node}$$

Η πρόσβαση στο στοιχείο της κεφαλής αντιστοιχεί στην ανάθεση

$$\boxed{node \ = \ queue(first + 1)}$$

διότι ο ψευδοδείκτης $first$ καταδεικνύει πάντοτε μία θέση πριν από το αρχαιότερο στοιχείο της ουράς, π.χ. μετά την ένθεση του πρώτου στοιχείου στην ουρά, έχομε $first = 0$ και $last = 1$. Η απαλοιφή επομένως του στοιχείου της κορυφής αντιστοιχεί στην ανάθεση

$$\boxed{first \ = \ first \ + \ 1}$$

Ο έλεγχος για το εάν η ουρά είναι κενή ή δεν έχει υπερβεί το μέγιστο μέγεθος του διανύσματος *queue* γίνεται με τους ακόλουθους δύο τελεστές

$$\boxed{is\_empty \ = \ (\ first \ == \ last\ );\ is\_full \ = \ (\ last \ == \ \mathbf{size}(\ queue\ )\ )}$$

Το πλήθος στοιχείων στην ουρά δίδεται από την διαφορά των δύο δεικτών, έτσι έχομε

$$\boxed{cardinality \ = \ last \ - \ first}$$

Παρατηρούμε ότι τόσο η ένθεση νέων στοιχείων όσο και η εξάλειψή τους από την ουρά οδηγούν σε αύξηση των τιμών των ψευδοδεικτών. Υπάρχει επομένως το ενδεχόμενο μετά από κάποιες προσθαφαιρέσεις στοιχείων να έχουν μετακινηθεί οι ψευδοδείκτες τόσο πολύ στα δεξιά ώστε να είναι αδύνατη η ένθεση νέων στοιχείων ακόμη και όταν όλες οι θέσεις του διανύσματος στ᾽ αριστερά του δείκτη $first$ είναι ελεύθερες για χρήση. Για να επιτευχθεί η χρήση αυτού του ελεύθερου χώρου θα πρέπει να επιτρέψουμε τους δείκτες να κινηθούν κυκλικά από το τέλος του διανύσματος στην αρχή του. Η υλοποίηση της ουράς μ᾽ αυτόν τον τρόπο είναι γνωστή ως κυκλική ουρά (circular queue).

Υπάρχουν ουσιαστικά δύο ισοδύναμοι τρόποι υλοποίησης της κυκλικής ουράς. Ο πρώτος απαιτεί συνεχή έλεγχο για υπέρβαση του μέγιστου μεγέθους της ουράς. Έτσι η μέχρι τώρα απλή αύξηση της μεταβλητής $last$ αντικαθίσταται από αύξηση υπό συνθήκη

```
if ( last == size(queue) ) then
    last = 1              ! κυκλική επανέναρξη
else
    last = last + 1  ! απλή αύξηση
endif
```

και η μέχρι τώρα απλή αύξηση της μεταβλητής $first$ αντικαθίσταται από αύξηση υπό συνθήκη

```
if ( first == size(queue) ) then
    first = 0              ! κυκλική επανέναρξη
else
    first = first + 1  ! απλή αύξηση
endif
```

Το πλήθος στοιχείων στην ουρά υπολογίζονται τώρα είτε, ως συνέχεια της αρχικής πρότασης, με προσθήκη για την κυκλική επανέναρξη

```
cardinality = last - first              ! όπως προηγουμένως
if ( cardinality < 0 ) cardinality = size(queue) + cardinality ! κύκλος
```

είτε μηδενίζοντας αρχικά την μεταβλητή *cardinality* και στην συνέχεια προσαυξά-

νοντάς την για κάθε ένθεση και μειώνοντάς την για κάθε απαλοιφή στοιχείου.

Η ουρά ελέγχεται εάν είναι κενή, όπως και προηγουμένως, δηλαδή

$$\boxed{is\_empty \ = \ (\ first \ == \ last\ )}$$

Εναλλακτικά, μπορεί βεβαίως να ελεγχθεί εξετάζοντας εάν η μεταβλητή *cardinality* έχει μηδενική τιμή.

Ο έλεγχος για το εάν η ουρά έχει φθάσει το μέγιστο επιτρεπτό μέγεθος και η περαιτέρω προσθήκες στοιχείων είναι αδύνατη γίνεται ως εξής

$$\boxed{is\_full \ = \ (\ first \ == \ last \ + \ 1\ )}$$

αφού λόγω των κυκλικών κινήσεων δεν αρκεί να εξετάσουμε απλώς εάν $last == \mathbf{size}(queue)$. Εναλλακτικά βεβαίως ο έλεγχος αυτός μπορεί να επιτευχθεί συγκρίνοντας την τιμή της μεταβλητής *cardinality* με την τιμή $\mathbf{size}(queue)$. Προσοχή όμως, μετά την έναρξη του πρώτου κύκλου, η ουρά μπορεί να διατηρεί το πολύ $\mathbf{size}(queue) - 1$ στοιχεία (Γιατί;).

Ένας δεύτερος ισοδύναμος τρόπος υλοποίησης της κυκλικής ουράς, πιο κομψός αλλά λιγότερο προφανής, είναι να υπολογίζονται οι τιμές των ψευδοδεικτών *first* και *last* με βάση το υπόλοιπο της ακέραιας διαίρεσης της τιμής τους με το $\mathbf{size}(queue)$. Αποφεύγονται έτσι τα δύο προηγούμενα **if then else endif** των αντίστοιχων προσαυξήσεων που αποδίδονται πια κομψά ως

$$\boxed{first \ = \ 1 \ + \ \mathbf{mod}(\ first,\ \mathbf{size}(queue)\ )}$$

και

$$\boxed{last \ = \ 1 \ + \ \mathbf{mod}(\ last,\ \mathbf{size}(queue)\ )}$$

όπου η συνάρτηση $\mathbf{mod}(k, n)$ επιστρέφει το ακέραιο υπόλοιπο της ακέραιας διαίρεσης k/n, το οποίο είναι k εφόσον $k < n$, 0 εφόσον $k = n$ ή $k = 0$ και 1 εφόσον $k = n + 1$.

1.3 Δομές Δένδρων

1.3.1 Διαζευκτικά Διακριτά Σύνολα

Ας είναι το $\mathcal{U}$ είναι καθολικό σύνολο με πληθύ $|\mathcal{U}| = n$ από το οποίο κατασκευάζονται διακριτά σύνολα $\mathcal{S}_k \subset \mathcal{U},\ k = 1, \ldots, K$ που είναι διαζευκτικά (ξένα μεταξύ τους), δηλαδή $\mathcal{S}_\ell \cap \mathcal{S}_m = \emptyset$ για όλα τα $\ell \neq m$. Θα υποθέσουμε ότι τα στοιχεία του καθολικού συνόλου $\mathcal{U}$ είναι οι ακέραιοι αριθμοί $\{1, 2, \ldots, n\}$. Η υπόθεση δεν είναι περιοριστική αφού ενδέχεται οι αριθμοί αυτοί ν' αποτελούν ψευδοδείκτες που καταδεικνύουν τις θέσεις άλλων πιο πολύπλοκων αντικειμένων με μεγάλο όγκο δεδομένων.

Θα χρησιμοποιήσουμε ένα ριζωμένο δένδρο για την παράσταση κάθε συνόλου $\mathcal{S}_k \subset \mathcal{U},\ k = 1, \ldots, K$ και συνεπώς ένα δάσος για την παράσταση όλων των K συνόλων. Οι κόμβοι των δένδρων αντιστοιχούν στα στοιχεία των συνόλων. Τα σύνολα ταυτοποιούνται από τις ρίζες των αντιστοίχων δένδρων. Θα ορίσουμε τρεις τελεστές:

δημιουργία μονοστοιχειακού συνόλου: Δεδομένου ενός στοιχείου $x \in \mathcal{U}$ δημιουργείται ένα σύνολο $\mathcal{S}_x = \{x\}$.

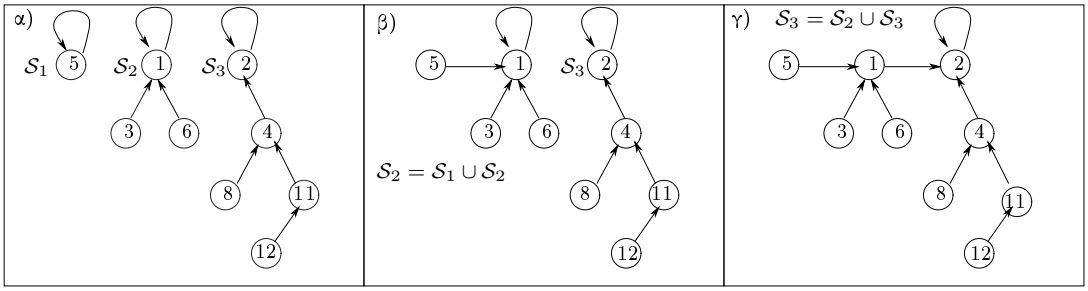
ταυτοποίηση συνόλου: Δεδομένου ενός στοιχείου $x \in \mathcal{U}$ ταυτοποιείται το σύνολο $\mathcal{S}_k$ που περιέχει το x.

ένωση συνόλων: Δεδομένων δύο συνόλων $\mathcal{S}_\ell$ και $\mathcal{S}_m$ δημιουργείται το σύνολο $\mathcal{S}_\ell \cup \mathcal{S}_m$.

Θα θεωρήσουμε ότι τα δένδρα είναι προσανατολισμένα προς την ρίζα. Κάθε κόμβος x ενός δένδρου είναι εφοδιασμένος συνεπώς μ᾽ έναν δείκτη $parent(x)$ που στο δένδρο αντιστοιχεί σε προσανατολισμένο τόξο από το x προς τον πατέρα του. Η ρίζα του δένδρου ξεχωρίζει από τους άλλους κόμβους διότι είναι πατέρας του εαυτού της. Εάν δηλαδή r είναι η ρίζα ενός δένδρου, τότε ο δείκτης $parent(r) = r$. Στην αναπαράσταση του δένδρου το γεγονός αυτό αποτυπώνεται μ᾽ έναν βρόχο στην ρίζα.

Παράδειγμα 1.1 Έστω το καθολικό σύνολο $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ και τα υποσύνολά του $S_1 = \{5\}$, $S_2 = \{1, 3, 6\}$ και $S_3 = \{2, 4, 8, 11, 12\}$. Ενδεχόμενες παραστάσεις αυτών των συνόλων ως δένδρα δίδονται στο Σχήμα 1.4(α). Το σύνολο S_1 είναι νεόδμητο μονοστοιχειακό σύνολο. Τα υπόλοιπα σύνολα έχουν συντεθεί από τις ενώσεις τέτοιων μονοστοιχειακών συνόλων. Η διαδικασία της ένωσης απεικονίζεται από τα υποσχήματα (β) και (γ) στο Σχήμα 1.4. Για την επίτευξη της πράξης $\mathcal{S}_1 \cup \mathcal{S}_2$, πολύ απλά, η ρίζα του $\mathcal{S}_1$, στην συγκεκριμένη περίπτωση ο κόμβος 5, γίνεται τέκνο της ρίζας του $\mathcal{S}_2$. Για να επιτευχθεί αυτό, ο δείκτης $parent(5)$ μεταβάλλει την τιμή του από 5 σε 1 και έτσι παύει ν᾽ αποτελεί ρίζα συνόλου. Με παρόμοια απλή διαδικασία το σύνολο $\mathcal{S}_2$ απορροφάται από το σύνολο $\mathcal{S}_3$ στο υποσχήμα (γ).

Κάθε σύνολο ταυτοποιείται από την ρίζα του. Έτσι το σύνολο $\mathcal{S}_1$ έχει ρίζα το 5, το $\mathcal{S}_2$ έχει ρίζα το 1 και το $\mathcal{S}_3$ έχει ρίζα το 2. Εάν πρέπει να γνωρίζουμε σε ποιό σύνολο ανήκει ένα στοιχείο x, αρκεί ν᾽ διατρέξουμε την διαδρομή από το x στην ρίζα του δένδρου. Π.χ. στο υπογράφημα (α), εκκινώντας από το 12 και ακολουθώντας την φορά των τόξων, διατρέχουμε τους κόμβους 11, 4 και 2, όπου αναγνωρίζουμε ότι το 2 είναι ρίζα δένδρου λόγω του βρόχου. Συμπεραίνουμε έτσι ότι το στοιχείο 12 ανήκει στο σύνολο με ρίζα το 2 και άρα στο σύνολο $\mathcal{S}_3$. ■



Σχήμα 1.4: Διακριτά σύνολα ως δένδρα

Η δημιουργία μονοστοιχειακού συνόλου για δεδομένο στοιχείο x μπορεί να επιτευχθεί με την ανάθεση

$$\boxed{parent(x) = x}$$

που είναι $O(1)$ πράξη.

Η ένωση δύο ξένων συνόλων με ρίζες x και r ($x \neq r$) επιτυγχάνεται επίσης με απλή ανάθεση. Η εντολή

$$\boxed{parent(x) = r}$$

καθιστά το πρώτο από τα δένδρα, αυτό με ρίζα x, υποδένδρο του δεύτερου, αυτού με ρίζα r. Και πάλι η πολυπλοκότητα είναι $O(1)$.

Η ταυτοποίηση του συνόλου στο οποίο ανήκει ένα στοιχείο x μπορεί να επιτευχθεί με την ταυτοποίηση της ρίζας του αντιστοίχου δένδρου. Η ταυτοποίηση της ρίζας γίνεται είτε επαναληπτικά

```
function find_set(x)
   set_root = x
    do while( parent(set_root) /= set_root ) ! εφόσον δεν είναι ρίζα
         set_root = parent(set_root) ! μετάβαση στον πατέρα
   enddo
   return( set_root )
end function
```

είτε, πιο εκλεπτυνσμένα, με αναδρομική συνάρτηση

```
recursive function find_set(x) result(set_root)
   set_root = x
   if ( set_root /= parent(set_root) ) then ! δεν είναι ρίζα
      set_root = find_set( parent(set_root) ) ! αναρρίχηση προς την ρίζα
   else        ! είναι ρίζα
       return( set_root )
   endif
end function
```

Ποιά είναι η πολυπλοκότητα αυτού του τελεστή; Προφανώς εξαρτάται από το μήκος της διαδρομής που πρέπει να διατρέξει από τον κόμβο x έως και την ρίζα του δένδρου. Στην χειρότερη των περιπτώσεων η διαδρομή αντιστοιχεί στο ύψος του δένδρου και είναι συνεπώς η μέγιστη δυνατή. Το x μπορεί ν' ανήκει σ' οποιδήποτε από τα δένδρα του δάσους. Ας είναι h το μέγιστο από τα ύψη του δάσους. Στην χείριστη των περιπτώσεων η πολυπλοκότητα του τελεστή είναι επομένως $O(h)$. Στην χείριστη των περιπτώσεων το δάσος αποτελείται απ' ένα μοναδικό δένδρο με n κόμβους συνδεδεμένους σε μία μοναδική διαδρομή Hamilton με τερματικό κόμβο την ρίζα. Αλλά τότε $h = n - 1$ και άρα η πολυπλοκότητα χείριστης περίπτωσης του τελεστή ταυτοποίησης του συνόλου είναι $O(n)$.

Για τις επιπτώσεις αυτής της γραμμικής πολυπλοκότητας ας εξετάσουμε την περίπτωση όπου αρχικά υπάρχει ένα δάσος από n μονοστοιχειακά σύνολα S_k, $k = 1, 2, \ldots, n$, από το καθολικό σύνολο $\mathcal{U}$, και στην συνέχεια εφαρμόζεται η διαδοχή πράξεων $\mathcal{S}_1 \cup \mathcal{S}_2, \text{find\_set}(1), \mathcal{S}_2 \cup \mathcal{S}_3, \text{find\_set}(1), \ldots, \mathcal{S}_{n-2} \cup \mathcal{S}_{n-1}, \text{find\_set}(1),$

$\mathcal{S}_{n-1} \cup \mathcal{S}_n$. Επειδή το $\mathcal{S}_1$ γίνεται υποδένδρο του $\mathcal{S}_2$ που με την σειρά του γίνεται υποδένδρο του $\mathcal{S}_3$ κ.ο.κ, το τελικό δένδρο που έτσι διαμορφώνεται αποτελεί πράγματι μία διαδρομή Hamilton με αρχικό κόμβο το 1 και τερματικό την ρίζα n. Οι $n-1$ ενώσεις για την δημιουργία του δένδρου απαίτησαν χρόνο $O(n)$. Ενδιάμεσα όμως εκτελέσθηκαν και $n-2$ πράξεις ταυτοποίησης, find_set(1). Ο συνολικός χρόνος εκτέλεσης αυτών των ταυτοποιήσεων είναι $1+2+\cdots+k+\cdots+n-2 = O(n^2)$ αφού η πράξη find_set(1) για δένδρο με k κόμβους σε σχηματισμό διαδρομής Hamilton απαιτεί χρόνο $O(k)$. Για τον χειρισμό n στοιχείων απ᾽ ένα κοινό σύνολο πληθύος n απαιτήθηκε δηλαδή συνολικά χρόνος $O(n^2)$. Το ίδιο αποτέλεσμα θα είχε προκύψει εάν η ακολουθία των πράξεων αποτελείτο από τις $n-1$ διαδοχικές εφαρμογές της ένωσης, $\mathcal{S}_1 \cup \mathcal{S}_2, \cdots, \mathcal{S}_{n-1} \cup \mathcal{S}_n$, να προηγούνται και τις n εφαρμογές ταυτοποίησης, find_set(1), find_set(2), find_set(n), να έπονται. Γενικότερα για κάθε λογική ακολουθία m τέτοιων πράξεων απαιτείται χρόνος $O(mn)$ στην χείριστη περίπτωση. Το αποτέλεσμα αυτό δεν είναι ικανοποιητικό.

Η χείριστη πολυπλοκότητα του τελεστή ταυτοποίησης συνόλου μπορεί να βελτιωθεί δραματικά εάν οι πράξεις επί των συνόλων γίνουν με μεγαλύτερη προσοχή έτσι ώστε ν᾽ αποτραπεί η παραγωγή δένδρων με μεγάλο ύψος. Θα εξετάσουμε βελτιωμένες εναλλακτικές τόσο στην απλή ένωση όσο και στην απλή ταυτοποίηση. Οι εναλλακτικές λύσεις βασίζονται σε δύο τεχνικές βελτίωσης.

Η πρώτη τεχνική ονομάζεται **σταθμισμένη ένωση** (weighted union). Υπάρχουν διάφορες εκδοχές της. Θα εξετάσουμε πρώτα την **ένωση κατά μέγεθος** (union by size). Στην περίπτωση αυτή διατηρείται για κάθε δένδρο το μέγεθός του, το πλήθος δηλαδή των κόμβων του. Κατά την ένωση δύο συνόλων, γίνεται σύγκριση των μεγεθών και το δένδρο με το μικρότερο πλήθος κόμβων γίνεται υποδένδρο του άλλου. Στο Σχήμα 1.4 ακολουθείται αυτη ακριβώς η διαδικασία κατά την ένωση των συνόλων.

Δεν απαιτείται πρόσθετη μνήμη για την αποθήκευση των μεγεθών αλλά μία απλή τροποποίηση στην αναγνώριση της ρίζας του δένδρου: Ο δείκτης $parent(r)$ αντί να καταδεικνύει την ρίζα r με την μορφή βρόχου καταδεικνύει το πλήθος των κόμβων που ανήκουν στο δένδρο με ρίζα r. Για ν᾽ αποφευχθεί σύγχιση, το πλήθος διατηρείται ως αρνητικός αριθμός.

Η δημιουργία μονοστοιχειακού συνόλου για δεδομένο στοιχείο x αντιστοιχεί τώρα στην ανάθεση

$$\boxed{parent(x) \;=\; -1 \text{ ! δένδρο ενός κόμβου}}$$

Η πολυπλοκότητα της δημιουργίας παραμένει $O(1)$.

Για την ένωση δύο συνόλων των οποίων τ᾽ αντίστοιχα δένδρα έχουν ρίζες $root_1$ και $root_2$ ($root_1 \neq root_2$), σύμφωνα με όσα ελέχθησαν πιο πάνω, ακολουθείται τώρα η εξής διαδικασία

```
total_nodes = parent(root1) + parent(root2) ! Νέο πλήθος κόμβων
if ( parent(root1) <= parent(root2) ) then
    parent(root1) = root2 ! Η ρίζα root2 ανήκει στο μεγαλύτερο σύνολο
    parent(root2) = total_nodes ! και είναι η νέα ολική ρίζα
else
    parent(root2) = root1 ! Η ρίζα root1 ανήκει στο μεγαλύτερο σύνολο
    parent(root1) = total_nodes ! και είναι η νέα ολική ρίζα
endif
```

Το πλήθος πράξεων έχει αυξηθεί περίπου στο διπλάσιο, η χείριστη πολυπλοκότητα του νέου τελεστή ένωσης παραμένει όμως $O(1)$.

Η διαδικασία ταυτοποίησης του συνόλου στο οποίο ανήκει ένα στοιχείο x παραμένει η ίδια, απλώς η σύγκριση αναγνώρισης της ρίζας μεταβάλλεται τώρα από $parent(x) == x$ σε $parent(x) < 0$.

Το ακόλουθο θεώρημα αποδεικνύει ότι η τεχνική της ένωσης κατά μέγεθος είναι αποδοτική.

Θεώρημα 1.1 *Ας είναι $\mathcal{U} = \{1, 2, \ldots, n\}$ ένα καθολικό σύνολο από το οποίο δημιουργούνται μονοστοιχειακά σύνολα και $\mathcal{F} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ το αρχικό δάσος από n μονοστοιχειακά δένδρα. Στο δένδρο $\mathcal{T}$ με n κόμβους που είναι το αποτέλεσμα μιας διαδοχικής εφαρμογής του τελεστή ένωσης κατά μέγεθος δεν υφίσταται κόμβος με βάθος μεγαλύτερο του $\log_2(n)$.* ◀

Ότι πράγματι είναι έτσι είναι εύκολο να διαπιστωθεί: Κάθε κόμβος μονοστοιχειακού δένδρου έχει βάθος 0. Όταν το βάθος του αυξάνει ως συνέπεια της ένωσης κατά μέγεθος, ο κόμβος τοποθετείται σ᾽ ένα δένδρο του οποίου το μέγεθος είναι τουλάχιστον δύο φορές μεγαλύτερο απ᾽ ό,τι προηγουμένως. Επομένως το βάθος του κόμβου μπορεί ν᾽ αυξηθεί το πολύ $\log_2(n)$ φορές. Μία πιο αυστηρή απόδειξη μπορεί να δοθεί μ᾽ επαγωγή στο n.

Επακόλουθο του θεωρήματος είναι ότι η πολυπλοκότητα χείριστης περίπτωσης του τελεστή ταυτοποίησης είναι $O(\log_2(n))$ για δένδρα με n κόμβους. Ένας συνδυασμός εφαρμογής $n-1$ ενώσεων και $n-2$ ταυτοποιήσεων απαιτεί τώρα χρόνο $O(n \log_2(n))$ αντί της προηγούμενης απαίτησης $O(n^2)$. Γενικότερα, μία ακολουθία m διαδοχικών πράξεων απαιτεί το πολύ $O(m \log_2(n))$ χρόνο.

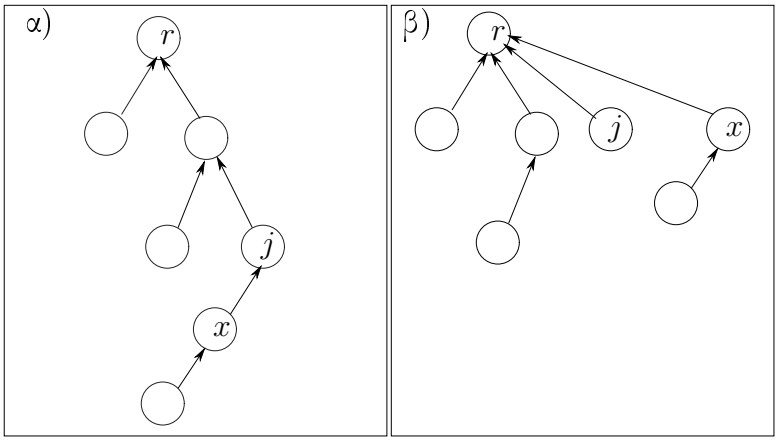
Παρόμοιο αποτέλεσμα ισχύει και για μια δεύτερη εκδοχή της σταθμισμένης ένωσης. Στην περίπτωση αυτή, η οποία είναι γνωστή ως **ένωση καθ᾽ ύψος** (union by height), χρησιμοποιείται το ύψος, $h(\mathcal{S})$, αντί του πλήθους των κόμβων του δένδρου $\mathcal{S}$. Βέβαια επειδή το ύψος ενός μονοστοιχειακού δένδρου είναι 0, αυτό αποτυπώνεται συνήθως με τον αρνητικό αριθμό $-1 = -(h(\mathcal{S}) + 1) = -(0 + 1)$, για να διατηρηθεί έτσι ο τρόπος αναγνώριση της ρίζα που εισήχθη προηγουμένως. Ο αριθμός $h(\mathcal{S})+1$ αντιστοιχεί στο πλήθος κόμβων στην μεγίστου μήκους διαδρομή με τερματικό κόμβο την ρίζα του δένδρου.

Κατά την ένωση δύο δένδρων, το χαμηλότερο δένδρο γίνεται υποδένδρο του υψηλότερου. Το ύψος του υψηλότερου δένδρου δεν μεταβάλλεται. Μόνον η ένωση δύο

δένδρων με το αυτό ύψος συνεπάγεται μεταβολή του ύψου με μία μονάδα. Προφανώς ούτε ο τελεστής δημιουργίας μονοστοιχειακού δένδρου ούτε η διαδικασία ταυτοποίησης μεταβάλλονται και μόνον η διαδικασία της ένωσης αλλάζει ελάχιστα σε σχέση με την πρώτη εκδοχή της σταθμισμένης ένωσης και διατυπώνεται ως εξής:

if ($parent(root_1) \ < \ parent(root_2)$) **then**
  $parent(root_1) \ = \ root_2$! Η ρίζα $root_2$ ανήκει στο υψηλότερο δένδρο
else
  $parent(root_2) \ = \ root_1$! Η ρίζα $root_1$ ανήκει στο υψηλότερο δένδρο
  ! ή τα δένδρα του αυτού ύψους και άρα αύξηση ύψους στο νέο δένδρο
  if ($parent(root_1) \ == \ parent(root_2)$) $parent(root_1) = parent(root_1) \ + \ 1$
endif

Ο χρόνος εκτέλεσης μιας μεικτής ακολουθίας πράξεων μπορεί να μειωθεί ακόμη περισσότερο με την εισαγωγή μιας πρόσθετης τεχνικής βελτίωσης που μεταβάλλει το μήκη των διαδρομών από συγκεκριμένους κόμβους προς την ρίζα. Και πάλι υπάρχουν διαφορετικές εκδοχές. Θα εξετάσουμε αναλυτικώτερα αυτήν που φέρει το όνομα **συμπίεση διαδρομής** (path compression). Η τεχνική αυτή επιβάλλεται με τον μετασχηματισμό της διαδικασίας find_set: Αφού η διαδρομή από τον κόμβο x στην ρίζα r διανυθεί μία φορά για την ταυτοποίηση του συνόλου που αντιστοιχεί στο συγκεκριμένο δένδρο, διανύεται ακόμη μία φορά και οι δείκτες $parent(j)$ για κάθε κόμβο j της διαδρομής, συμπεριλαμβανομένου και του x αλλά όχι της r και του άμεσου υιού της, μεταβάλλουν την οποιαδήποτε αναφορά τους σε απ' ευθείας αναφορά στο r, επιβάλλεται δηλαδή η ανάθεση $parent(j) = r$. Αυτό έχει ως αποτέλεσμα την σμίκρυνση του βάθους του δένδρου. Η διαδικασία απεικονίζεται με τα δένδρα στο Σχήμα 1.5, όπου το δένδρο (α) μετά την εφαρμογή της πράξης find_set(x) σε συνδυασμό με συμπίεση διαδρομής μετασχηματίζεται στο δένδρο (β), το οποίο έχει σαφώς μικρότερο ύψος από το αρχικό.



Σχήμα 1.5: Συμπίεση διαδρομής

Η νέα συνάρτηση που συνδυάζει find_set και συμπίεση διαδρομής έχει τώρα την ακόλουθη επαναληπτική μορφή

```
function collapsing_find_set(x)
   ! Πρώτη φάση: Ταυτοποίηση συνόλου
   set_root = x
   do while( parent(set_root) > 0 ) ! εφόσον δεν είναι ρίζα
         set_root = parent(set_root) ! μετάβαση στον πατέρα
   enddo
   ! Δεύτερη φάση: Συμπίεση διαδρομής
   node = x ! Έναρξη με τον κόμβο x
   do while ( node > 0 ) ! εφόσον ο κόμβος δεν είναι ρίζα
      next_node = parent(node) ! ο πατέρας του είναι επόμενος
      parent(node) = set_root ! νέος γονιός του είναι η ρίζα
      node = next_node ! μετάβαση στον επόμενο κόμβο
   enddo
end function
```

ή την, πιο κομψή, αναδρομική μορφή

```
recursive function collapsing_find_set(x) result(set_root)
   set_root = x
   if (parent(set_root) > 0 ) then ! δεν είναι ρίζα
      ! Φάση Πρώτη και Δεύτερη
      parent(set_root) = collapsing_find_set( parent(set_root) )
   else        ! είναι ρίζα
      return( set_root )
   endif
end function
```

Η διαδικασία της αναδρομικής συνάρτησης collapsing_find_set είναι ενδιαφέρουσα: η ρίζα καταχωρείται στο $parent(x)$ και επιστρέφεται. Επειδή όμως αυτό συμβαίνει αναδρομικά, όλοι οι κόμβοι στην διαδρομή από το x στην ρίζα θα υποστούν αυτήν την μεταβολή.

Εξετάζοντας τον συνδυασμό των τελεστών της ένωσης κατά μέγεθος και της ταυτοποίησης με συμπίεση διαδρομής, παρατηρούμε ότι έχουμε αυξήσει το πλήθος των πράξεων και είναι φυσικό ν' αναρωτηθούμε πως είναι δυνατόν ν' αναμένουμε καλύτερη συμπεριφορά απ' ό,τι προηγουμένως στα πλαίσια μιας ακολουθίας εφαρμογών αυτού του συνδυασμού. Και όμως, είναι δυνατόν ν' αποδειχθεί ότι ο συνδυασμός αυτός οδηγεί σε σχεδόν γραμμική πολυπλοκότητα χείριστης περίπτωσης και άρα σε σχεδόν $O(1)$ πολυπλοκότητα ανά τελεστή. Ο λόγος είναι ότι το αυξημένο πλήθος πράξεων διαμοιράζεται. Έχομε δηλαδή παράδειγμα **χρεολυτικής πολυπλοκότητας** (amortized complexity), όπου μέρος του κόστους των ακριβών εφαρμογών «απλώνεται» στις φθηνότερες εφαρμογές των τελεστών, οπότε η χρεολυτική πολυπλοκότητα κάθε εφραμογής γίνεται σχεδόν $O(1)$. Πιο συγκεκριμένα, ο χρόνος εκτέλεσης μιας ακολουθίας $n-1$ ενώσεων και m ταυτοποιήσεων, $m \geq n$, έχει τώρα πολυπλοκότητα $O(m \cdot \alpha(m,n))$, όπου $\alpha(m,n)$ είναι η αντίστροφη της συνάρτησης Ackerman.

Ορισμός 1.1 *Συνάρτηση* **Ackerman**
Ο ανδρομικός τύπος

$$A(1,j) = 2^j \tag{1.1}$$

$$A(i,j) = \begin{cases} A(i-1,2) & \text{για } i \geq 2, \\ A(i-1, A(i,j-1)) & \text{για } i,j \geq 2, \end{cases} \tag{1.2}$$

ορίζει την συνάρτηση Ackerman.

Ο τύπος

$$\alpha(m,n) = \min\{i \geq 1 | A(i, \lfloor m/n \rfloor) \geq \log_2(n)\} \text{ για } m \geq n \geq 1 \tag{1.3}$$

ορίζει την αντίστροφη της συνάρτησης Ackerman.

Η συνάρτηση $\alpha(n) = A(2,n)$ ονομάζεται συνάρτηση Ackerman μιας μεταβλητής. Η αντίστροφή της

$$\log_2^*(n) = \min\{i | log_2^{(i)}(n) \leq 1\}, \tag{1.4}$$

όπου $\log_2^{(0)}(n) = n$ και $\log_2^{(i)}(n) = \log_2(\log_2^{(i-1)}(n)$, δηλώνει πόσες φορές θα πρέπει να εφαρμοσθεί αναδρομικά ο λογάρισθμος στο n έως ότου επιτευχθεί τιμή ≤ 1. ◀

Η συνάρτηση $A(i,j)$ αυξάνει ταχύτητα καθώς τα i και j αυξάνουν. Π.χ. $A(2,j) = A(1, A(2,j-1)) = 2^{A(2,j-1)} = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}_{j \text{ φορές}}$. Συνεπώς η αντίστροφή της $\alpha(m,n)$ αυξάνει απειροελάχιστα καθώς τα m και n αυξάνουν. Η $\alpha(m,n)$ αυξάνει με μικρότερο ρυθμό από την $\log_2^*(n)$. Π.χ. για $2^{16} = 65536$ και επειδή $\log_2 \log_2 \log_2 \log_2(65536) = 1$ έχομε $log_2^*(65536) = 4$. Από την άλλη πλευρά $A(3,1) = A(2,2) = 2^{2^2} = 16$ και $\alpha(m,n) \leq 3$ για $n < 2^{16}$ και $m \geq n$. Ο αριθμός $2^{2^{16}} = 2^{65536}$ έχει είκοσι χιλιάδες (20000) ψηφία, όμως $\log_2^*(2^{16}) = 5$. Έτσι, επειδή $A(4,1) = A(3,2) = A(2, A(3,1)) = A(2,16) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\Big\}_{16 \text{ φορές}}$, μπορούμε στην πράξη να υποθέσουμε ότι σχεδόν πάντοτε $\alpha(m,n) \leq 4$.

Θεώρημα 1.2 *Ας είναι $\mathcal{U} = \{1, 2, \ldots, n\}$ ένα καθολικό σύνολο από το οποίο δημιουργούνται μονοστοιχειακά σύνολα και $\mathcal{F} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ το αρχικό δάσος από n μονοστοιχειακά δένδρα. Ο μέγιστος χρόνος για την εκτέλεση μιας μεικτής ακολουθίας $k \leq n-1$ ενώσεων κατά μέγεθος και $m = \Omega(n)$ ταυτοποιήσεων με συμπίεση διαδρομών είναι $\Theta(m \cdot \alpha(m,n))$.* ◀

Προκύπτει από το θεώρημα ότι το $m \cdot \alpha(m,n)$ αποτελεί τόσο ένα άνω όριο $O(m \cdot \alpha(m,n))$ όσο και ένα κάτω όριο $\Omega(m \cdot \alpha(m,n)$ για την ακολουθία πράξεων και ότι επομένως υπάρχουν ακολουθίες πράξεων για τις οποίες το όριο αυτό είναι επιτεύξιμο. Αν και η $\alpha(m,n)$ αυξάνει με πολύ μικρό ρυθμό, δεν μπορούμε να πούμε ότι είναι μία σταθερά και άρα ο χρόνος εκτέλεσης δεν είναι γραμμικός αλλά σχεδόν γραμμικός. Επειδή η $\alpha(m,n)$ αυξάνει με μικρότερο ρυθμό από την $\log_2^*(n)$, μπορούμε να γράψουμε ότι η πολυπλοκότητα είναι $O(m \log_2^*(n))$ για $m = \Omega(n)$.

Ο τελεστής ταυτοποίησης με συμπίεση διαδρομής θα μπορούσε να χρησιμοποιηθεί σε συνδυασμό με τον τελεστή της απλής ένωσης χωρίς όμως τις εγγυήσεις του θεωρήματος. Θα μπορούσε να συνδυασθεί με την ένωση καθ' ύψος; Η απάντηση είναι όχι ακριβώς. Η συμπίεση διαδρομής έχει επιπτώσεις στα ύψη των δένδρων και δεν γνωρίζουμε πως να επαναϋπολογίσουμε τα ύψη αποδοτικά μετά από κάθε συμπίεση. Αυτό όμως δεν είναι ανυπέρβλητο εμπόδιο. Τα «ύψη» που προκύπτουν αποτελούν υπερεκτιμήσεις των πραγματικών υψών και αποκαλούνται **βαθμοί** (ranks) των δένδρων. Ο τελεστής της ένωσης καθ' ύψος καθίσταται τελεστής **ένωσης κατά βαθμό** (union by rank) χωρίς ν' αλλάξει κάτι στην διατύπωσή του. Ο συνδυασμός του τελεστή ταυτοποίησης με συμπίεση διαδρομής και του τελεστή ένωσης κατά βαθμό υπόκειται στις εγγυήσεις του θεωρήματος.

Επειδή ο τελεστής ταυτοποίησης με συμπίεση διαδρομής απαιτεί να διανυθεί η διαδρομή από τον κόμβο στην ρίζα δύο φορές, έχουν προταθεί δύο εναλλακτικοί τελεστές που επιτυγχάνουν την μείωση του ύψους του δένδρου διατρέχοντας όμως την διαδρομή μία μόνον φορά:

Στον τελεστή ταυτοποίησης με **διάσπαση της διαδρομής** (path splitting) αλλάζει η αναφορά του $parent(j)$ για όλους τους κόμβους j της διαδρομής, πλην της ρίζας και του άμεσου υιού της, σε αναφορά στον παππού του j.

Στο τελεστή ταυτοποίησης με **διχοτόμηση της διαδρομής** (path halving) αλλάζει η αναφορά του δείκτη $parent(j)$ για κάθε δεύτερο κόμβο j στην διαδρομή, πλην της ρίζας και του άμεσου υιού της, σε αναφορά στον παππού του j.

Και οι δύο νέοι τελεστές ταυτοποίησης σε συνδυασμό με την ένωση κατά μέγεθος υπόκεινται στις εγγυήσεις του θεωρήματος.

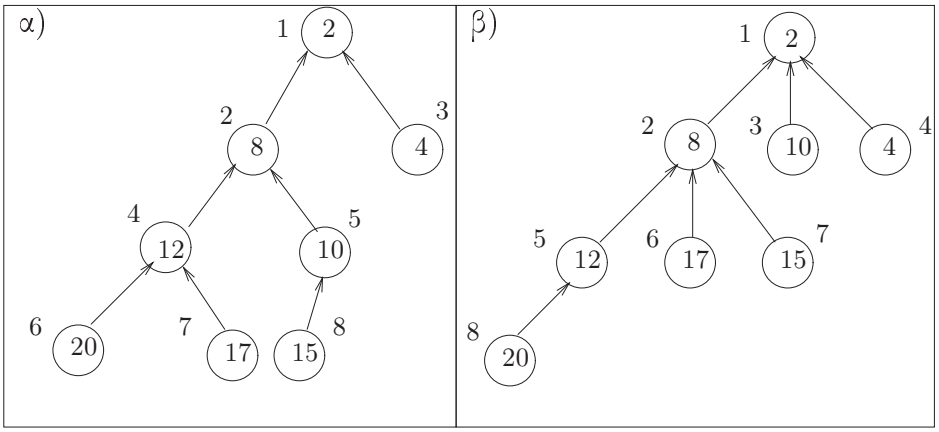
1.3.2 Ουρά Προτεραιότητας και Σωρός

Αναφέραμε στην § 1.1.4 ότι η ουρά προτεραιότητας είναι μία δομή δεδομένων που επιτρέπει την πρόσβαση μόνον στο στοιχείο της με την μέγιστη (ή, εναλλακτικά, ελάχιστη) τιμή ή προτεραιότητα. Αναφέραμε επίσης ότι θα μπορούσε ενδεχομένως να προσομοιηθεί με την συνήθη ουρά εφόσον εφοδιασθεί με τελεστή ανεύρεσης της μέγιστης (ή ελάχιστης) τιμής ή, εναλλακτικά, με τελεστή διάταξης των στοιχείων της σε φθίνουσα (ή αύξουσα) διάταξη τιμών. Στην πρώτη περίπτωση η ένθεση στοιχείων στην ουρά είναι $O(1)$ πολυπλοκότητας, ο εντοπισμός όμως και η ενδεχόμενη απαλοιφή του στοιχείου της με την μέγιστη (ελάχιστη) τιμή είναι γραμμικής πολυπλοκότητας $O(n)$. Αντίθετα, στην δεύτερη περίπτωση, η ένθεση στοιχείων είναι γραμμικής πολυπλοκότητας για να διατηρηθεί η διάταξη τιμών, ενώ η ανεύρεση και η απαλοιφή του στοιχείου με την μέγιστη (ελάχιστη) τιμή είναι $O(1)$. Θα εξετάσουμε εδώ μία νέα αφηρημένη δομή δεδομένων, τον σωρό που επιτρέπει την ένθεση στοιχείων σε λογαριθμικό χρόνο και την ανεύρεση του στοιχείου με την μέγιστη (ή ελάχιστη) τιμή σε χρόνο $O(1)$. Ανάλογα με το εάν είναι επιθυμητή η πρόσβαση στο στοιχείο της ουράς προτεραιότητας με την ελάχιστη ή μέγιστη τιμή έχομε σωρό ελαχίστου (min heap) και σωρό μεγίστου (max heap) αντίστοιχα. Θα εξετάσουμε τον σωρό ελαχίστου αφού η περιγραφή σωρού μεγίστου είναι ακριβώς παράλληλη.

Ο **σωρός** (heap) είναι μία αφηρημένη δομή δεδομένων που αποτελείται από μία συλλογή στοιχείων οργανωμένων σε μορφή δένδρου, όπου κάθε στοιχείο x έχει κατανεμηθεί μία τιμή ή προτεραιότητα $value(x)$ και όπου το δένδρο είναι σε **διάταξη σωρού** (heap order) ενώ οι κόμβοι του είναι **αριθμημένοι πρώτα κατά πλάτος** (breadth first order). Η διάταξη σωρού συνεπάγεται ότι το στοιχείο της συλλογής με την μικρότερη τιμή είναι ρίζα του δένδρου και ότι η ιδιότητα αυτή ισχύει αναδρομικά για την ρίζα κάθε υποδένδρου. Πιο συγκεκριμένα, εάν x ένα στοιχείο του δένδρου και $parent(x)$ ο γονίο του, τότε η διάταξη σωρού συνεπάγεται ότι

$$value(\ parent(x)\ ) \ \leq \ value(\ x\ ). \tag{1.5}$$

Παράδειγμα 1.2 Έστω ένα σύνολο 8 στοιχείων με τιμές $\{2, 4, 8, 10, 12, 15, 17, 20\}$. Στο Σχήμα 1.6 δίδονται δύο διαφορετικές παραστάσεις του συνόλου με μορφή δένδρων τα οποία ικανοποιούν την διάταξη σωρού. Δεν υπάρχει δηλαδή μοναδικότητα στην παράσταση ενός συνόλου ως σωρού. Τα τόξα των δένδρων είναι από υιό προς πατέρα και αντιστοιχούν στους δείκτες *parent*. Η ρίζα r κάθε δένδρου δεν έχει πατέρα οπότε μπορούμε να θεώρήσουμε ότι $parent(r) = -\infty$. Οι τιμές εμφανίζονται στο εσωτερικό των κόμβων ενώ οι αριθμοί δίπλα στους κόμβους αντιστοιχούν στην αρίθμησή τους πρώτα κατά πλάτος. Η ρίζα κάθε δένδρου περιέχει την μικρότερη μεταξύ όλων των τιμών. Το ίδιο ισχύει και για τις ρίζες των υποδένδρων. Π.χ. στο Σχήμα 1.6(β), ο κόμβος 2 περιέχει την τιμή 8 που είναι η μικρότερη μεταξύ των τιμών $\{8, 12, 17, 15, 20\}$ που περιέχουν οι κόμβοι του υποδένδρου που ριζώνει σ' αυτόν τον κόμβο. ■



Σχήμα 1.6: Δένδρα σε διάταξη σωρού

Για τον σωρό ορίζονται οι ακόλουθοι στοιχειώδεις τελεστές:

ένθεση ενός νέου στοιχείου στον σωρό.

δημιουργία σωρού από τα στοιχεία ενός δεδομένου συνόλου.

πρόσβαση ελαχίστου για την επιστροφή του στοιχείου με την ελάχιστη τιμή. Εάν ο σωρός είναι κενός επιστρέφει κάποια προκαθορισμένη τιμή *null*.

απαλοιφή ελαχίστου για την απαλοιφή του στοιχείου με την ελάχιστη τιμή από τον σωρό.

Στους τελεστές του σωρού προστίθενται συνήθως τουλάχιστον δύο ακόμη τελεστές:

απαλοιφή αυθαίρετου στοιχείου για την απαλοιφή ενός οποιουδήποτε στοιχείου από τον σωρό.

αλλαγή τιμής στοιχείου για την ανάθεση νέας τιμής σε ένα οποιοδήποτε στοιχείο του σωρού.

Θα εξετάσουμε τον σωρό σε μια τέτοια εκτεταμένη μορφή.

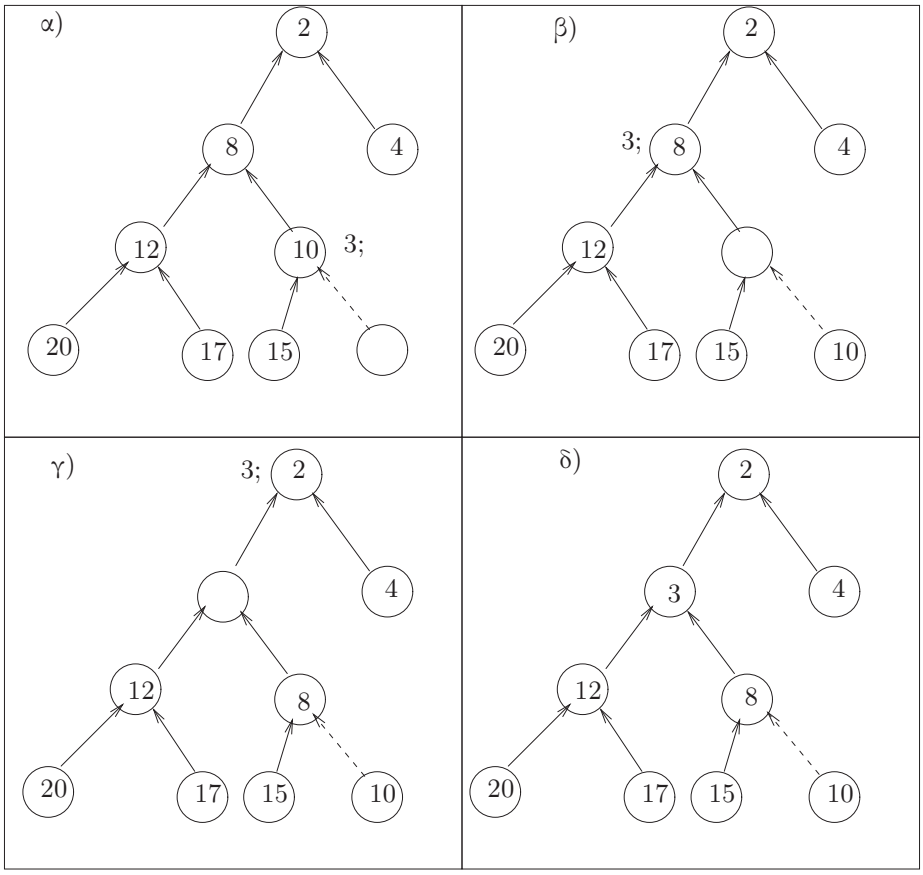
Ας είναι $1, 2, \ldots, n$ η πρώτα κατά πλάτος αρίθμηση των κόμβων του δένδρου του σωρού. Τότε ο κόμβος 1 αντιστοιχεί στην ρίζα του δένδρου και επομένως η πρόσβαση ελαχίστου είναι άμεση και άρα πολυπλοκότητας $O(1)$:

$$\boxed{min\_value \ = \ value(root)}$$

Η ένθεση ενός νέου στοιχείου τιμής $p = new\_value$ στον σωρό επιτυγχάνεται με την ακόλουθη διαδικασία: Δημιουργείται ένας νέος κενός κόμβος $n+1$ που αποτελεί φύλλο του δένδρου. Ας είναι x ο πατέρας του νέου κόμβου, $x = parent(n+1)$, και $p_x = value(x)$ η τιμή του. Εάν η ένθεση της τιμής p στον νέο κόμβο $n+1$ δεν παραβιάζει την διάταξη του σωρού, εάν δηλαδή $p \geq p_x$, τότε η διαδικασία έχει ολοκληρωθεί. Εάν όμως η ένθεση της τιμής p στον νέο κόμβο $n+1$ παραβιάζει την διάταξη του σωρού, εάν δηλαδή $p < p_x$, τότε η τιμή του πατέρα, p_x, καταλαμβάνει τον νέο κόμβο και η νέα τιμή p εφόσον δεν παραβιάζει την διάταξη του σωρού καταλαμβάνει τον κόμβο του πατέρα, x, και η διαδικασία τερματίζεται. Εάν όμως η διάταξη του σωρού παραβιάζεται, η διαδικασία σύγκρισης τιμών επαναλαμβάνεται για την κατάληψη του κενού κόμβου x είτε από την τιμή p είτε από την τιμή $value(parent(x))$. Η διαδικασία αυτή επναλαμβάνεται εφόσον ο πατέρας, $parent(x)$, είναι ορισμένος και η τιμή p υπερβαίνει την $value(parent(x))$: Η τιμή $value(parent(x))$ καταλαμβάνει τον κενό κόμβο και ο κενός κόμβος «μεταφέρεται» στον κόμβο $parent(x)$. Ο ακόλουθος ψευδοκώδικας αποδίδει την διαδικασία:

```
son = n+1 ! Δημιουργία ενός νέου φύλλου
father = parent(son) ! με συγκεκριμένο πατέρα
do while( son /= root and value(father) > new_value )
   value(son) = value(father) ! ανάθεση τιμής στον κενό κόμβο
   son = father ! κίνηση κενού κόμβου ...
   father = parent(son) ! ...προς την ρίζα
enddo
value(son) = new_value ! προσθήκη της νέας τιμής στον σωρό
```

Ο κενός κόμβος που αρχικά δημιουργήθηκε ομοιάζει προς φυσαλίδα η οποία αναδύεται από το $(n+1)-$φύλλο του δένδρου προς την ρίζα του ακολουθώντας την μοναδική διαδρομή σύνδεσής τους. Στην χειρότερη των περιπτώσεων, η φυσαλίδα θ' αναδυθεί στην ρίζα και θα έχει διανύσει μία διαδρομή ίση με το βάθος, d, του



Σχήμα 1.7: Ένθεση νέου στοιχείου στον σωρό

δένδου. Συνεπώς, η χείριστη πολυπλοκότητα του τελεστή ένθεσης νέου στοιχείου είναι $O(d)$.

Παράδειγμα 1.3 Στο Σχήμα 1.7(α) για την ένθεση ενός νέου στοιχείου με τιμή 3 δημιουργείται ένα κενό φύλλο με πατέρα τον κόμβο που περιέχει την τιμή 10. Επειδή $3 < 10$, στο Σχήμα 1.7(β), η τιμή 10 μεταφέρεται στο φύλλο και ο κόμβος που κατελάμβανε μένει κενός. Για την κατάληψη του κενού κόμβου, η τιμή 3 συγκρίνεται με την τιμή 8 του πατέρα του. Επειδή $3 < 8$, στο Σχήμα 1.7(γ), η τιμή 8 μεταφέρεται στον κενό κόμβο και ο πατέρας του μένει κενός. Για την κατάληψη του νέου κενού κόμβου, η τιμή 3 συγκρίνεται με την τιμή 2 του πατέρα του που εδώ τυγχάνει να είναι η ρίζα του δένδρου. Επειδή $3 > 2$, στο Σχήμα 1.7(δ), η τιμή 3 καταλαμβάνει τον κενό κόμβο και η διαδικασία της ένθεσης τερματίζεται. Το νέο δένδρο πληροί την διάταξη σωρού. ■

Η απαλοιφή ενός αυθαίρετου στοιχείου με τιμή p_X από τον σωρό είναι κάπως πιο περίπλοκη από την διαδικασία της ένθεσης διότι η αφαίρεση ενός στοιχείου συνεπάγεται σμίκρυνση του πλήθους των κόμβων κατά έναν. Ένα φύλλο του δένδρου, π.χ. ο nστός κόμβος σύμφωνα με την πρώτα κατά πλάτος αρίθμηση, απαλοίφεται από το δένδρο. Ας είναι $p_n = value(n)$ η τιμή που περιείχε. Εάν το φύλλο αυτό αντιστοιχεί στο στοιχείο που έπρεπε ν' απαλειφθεί η διαδικασία τερματίζεται. Δια-

φορετικά, ο κόμβος X που αντιστοιχεί στο προς απαλοιφή στοιχείο εκκενώνεται δημιουργούντας έτσι μία κενή θέση. Όμως η τιμή p_n από το απαλειφθέν φύλλο θα πρέπει να επανατοποθετηθεί στο δένδρο του σωρού. Η ερώτηση είναι βεβαίως εάν μπορεί να γίνει ένθεσή της στον κενό κόμβο χωρίς να διαταραχθεί η διάταξη σωρού. Υπάρχουν δύο περιπτώσεις:

(α) Εάν $p_n \leq p_X$, η ένθεση ακολουθεί την διαδικασία της αναδυόμενης φυσαλίδας που ήδη περιγράψαμε:

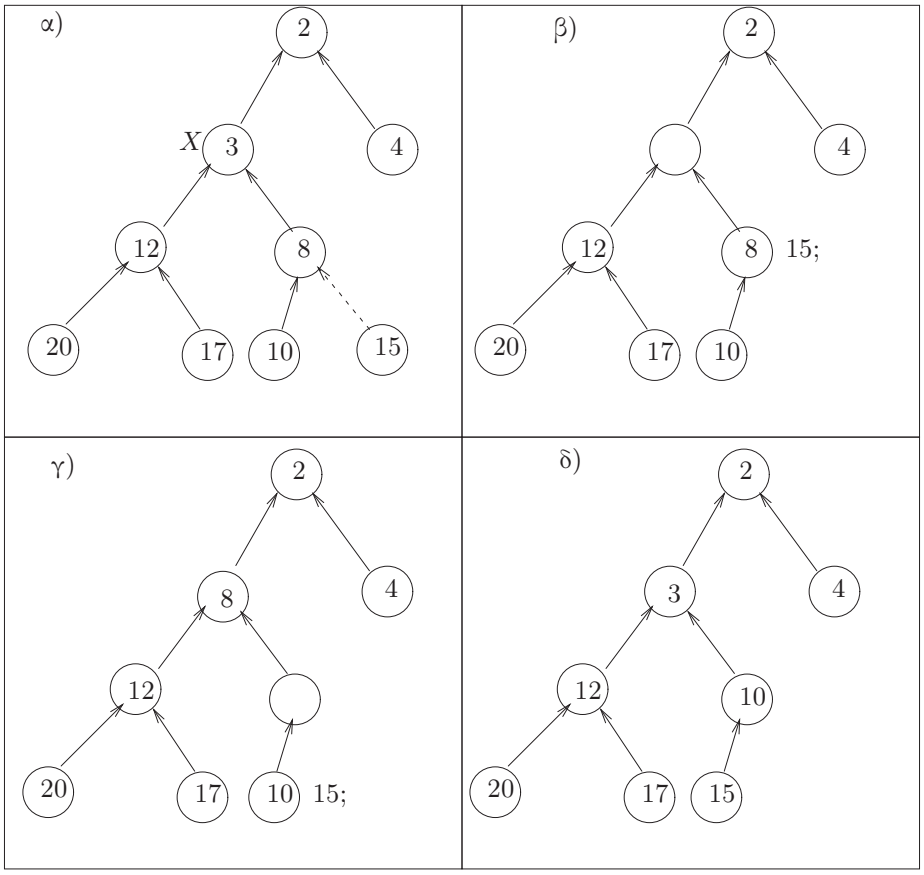
```
p_n = value(n) ! Η τιμή πρέπει να επανατοποθετηθεί ...
n = n - 1 ! ...αφού ο κόμβος n απαλοίφεται
son = X ! Ο κόμβος X θεωρείται κενός ...
father = parent(son) ! Ο πατέρας του X
do while( son /= root and value(father) > p_n )
   value(son) = value(father) ! ανάθεση τιμής στον κενό κόμβο
   son = father ! κίνηση κενού κόμβου ...
   father = parent(son) ! ...προς την ρίζα
enddo
value(son) = p_n ! επανατοποθέτηση τη τιμής p_n στον σωρό
```

(β) Εάν $p_n > p_X$, και εάν ο κόμβος Y είναι ο υιός του X με την μικρότερη τιμή μεταξύ όλων των υιών του, τότε εάν $value(Y) > p_n$, ο κενός κόμβος X καταλαμβάνεται από την τιμή p_n και η διαδικασία τερματίζεται. Διαφορετικά, εφόσον $value(Y) < p_n$, η τιμή του $value(Y)$ καταλαμβάνει τον κενό κόμβο X και ο κόμβος Y μένει κενός. Η διαδικασία αυτή επαναλαμβάνεται για τον νέο κενό κόμβο Y, «μεταφέροντας» έτσι τον κενό κόμβο προς τα φύλλα του δένδρου έως ότου επιτευχθεί η ένθεση της τιμής p_n. Για την περίπτωση (β) ο ψευδοκώδικας διατυπώνεται ως εξής:

```
p_n = value(n) ! Η τιμή p_n πρέπει να επανατοποθετηθεί ...
n = n - 1 ! ...αφού ο κόμβος n απαλοίφεται
father = X ! Ο κόμβος X θεωρείται κενός
son = min_value_son(father) ! Ο υιός με την μικρότερη τιμή
do while( son /= null and p_n > value(son) )
   value(father) = value(son) ! ανάθεση τιμής στον κενό κόμβο
   father = son ! κίνηση κενού κόμβου ...
   son = min_value_son(father) ! ...προς τα φύλλα
enddo
value(father) = p_n ! επανατοποθέτηση της τιμής p_n στον σωρό
```

Σε αντίθεση προς την αναδυόμενη φυσαλίδα, στην διαδικασία της περίπτωσης (β), ο κενός κόμβος «μεταφέρεται» προς τα κάτω απομακρυνόμενος από την ρίζα του δένδρου και θα μπορούσε να παρομοιασθεί με μεταλλική σφαίρα που βυθίζεται προς τα φύλλα του δένδρου. Στην χείριστη περίπτωση, η σφαίρα εκκινεί από την ρίζα του δένδρου και βυθιζόμενη ακολουθεί την μέγιστη διαδρομή προς κάποιο από τα φύλλα του διανύοντας έτσι μία διαδρομή ίση με το βάθος d του δένδρου. Ας είναι το μέγιστο πλήθος υιών που δύναται να έχει ένας κόμβος μία σταθερά c. Η χείριστη

πολυπλοκότητα της διαδικασίας (β) είναι τότε $O(c \cdot d)$. Η χείριστη πολυπλοκότητα της διαδικασίας (α) έχει ήδη χαρακτηρισθεί ως $O(d)$. Με τον τερματισμό της διαδικασίας είτε της περίπτωσης (α) είτε της (β) το δένδρο είναι σε διάταξη σωρού και ο σωρός αποτελείται από $n-1$ στοιχεία.



Σχήμα 1.8: Απαλοιφή στοιχείου από τον σωρό

Παράδειγμα 1.4 Στο Σχήμα 1.8(α), για την απάλειψη του στοιχείου που αντιστοιχεί στον κόμβο X, απαλείφεται η τιμή 3 δημιουργώντας μία κενή θέση στον κόμβο X. Αλλά επιπρόσθετα εξαλείφεται και ένα φύλλο του δένδρου, αυτό που περιέχει την τιμή 15. Το αποτέλεσμα αυτών των απαλοιφών είναι το δένδρο του Σχήματος 1.8(β). Από τους υιούς του κενού κόμβου ο ένας περιέχει την τιμή 12 και ο άλλος την τιμή 8 που είναι και η μικρότερη των δύο. Η τιμή 8 και η τιμή 15 από το απαλειφθέν φύλλο ανταγωνίζονται για την κατάληψη του κενού κόμβου. Επειδή $8 < 15$, ο κενός κόμβος καταλαμβάνεται από την τιμή 8 δημιουργώντας έτσι έναν νέο κενό κόμβο στο δένδρο του Σχήματος 1.8(γ). Ο νέος κενός κόμβος έχει έναν μόνον υιό με τιμή 10. Επειδή $10 < 15$, η τιμή 10 καταλαμβάνει τον κενό κόμβο δημιουργώντας έναν νέο κενό κόμβο που στην περίπτωση αυτή είναι φύλλο και άρα δεν έχει υιούς. Έτσι, η τιμή 15 καταλαμβάνει το κενό φύλλο και το αποτέλεσμα είναι το δένδρο του Σχήματος 1.8(δ), το οποίο πληροί την διάταξη σωρού και είναι κατά ένα στοιχείο μικρότερο από το αρχικό δένδρο του Σχήματος 1.8(α). ■

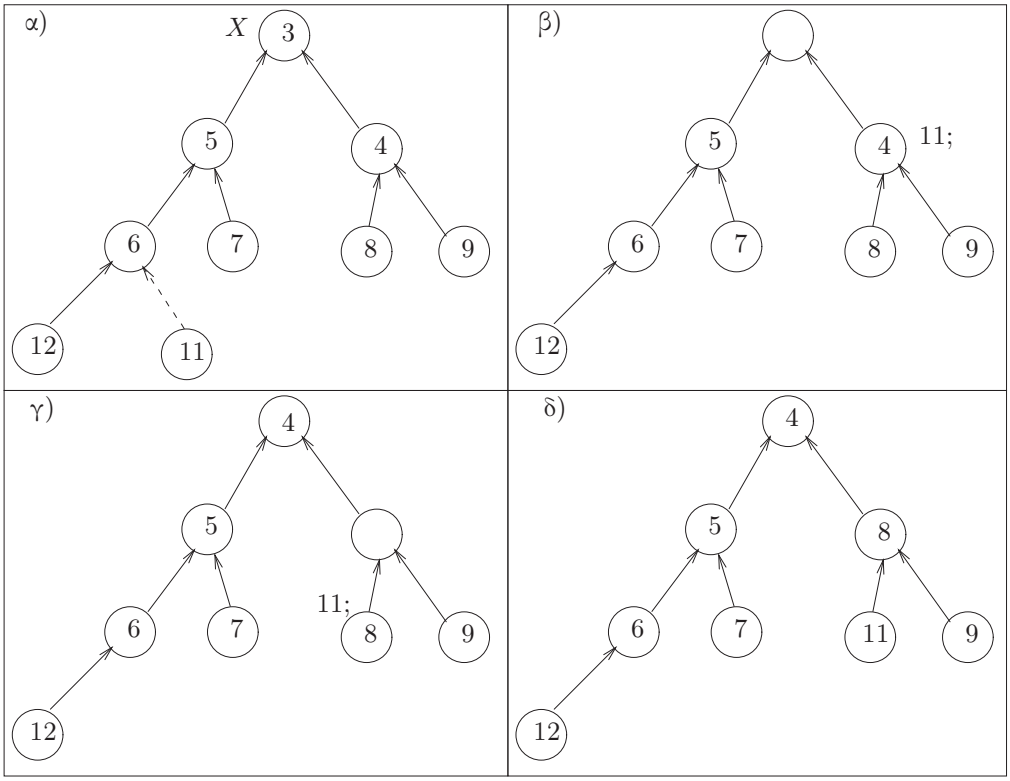
Η διαδικασία (α) όπου ο κενός κόμβος κινείται προς την ρίζα μπορεί ν' ονομασθεί

«ανάδυση» και είναι γνωστή στην διεθνή βιβλιογραφία ως sift-up, ενώ η διαδικασία (β) όπου ο κενός κόμβος κινείται προς τα φύλλα θα μπορούσε ν' ονομασθεί «κατάδυση» και γνωστή στην διεθνή βιβλιογραφία ως sift-down.

Η απαλοιφή ελαχίστου αντιστοιχεί στην απαλοιφή της ρίζας του σωρού και είναι μία ιδιαίτερη περίπτωση της απαλοιφής αυθαίρετου στοιχείου του που αντιστοιχεί στην διαδικασία της περίπτωσης (β). Επιτυγχάνεται δηλαδή με κατάδυση:

$min\_value = value(root)$! Η τιμή της ρίζας είναι η ελάχιστη του σωρού
$p_n = value(n)$! Η τιμή p_n πρέπει να επανατοποθετηθεί ...
$n = n - 1$! ...αφού ο κόμβος n απαλοίφεται
$father = root$! Ο κόμβος $root$ θεωρείται κενός
$son = min\_value\_son(father)$! Ο υιός με την μικρότερη τιμή
do while($son /= null$ **and** $p_n > value(son)$)
  $value(father) = value(son)$! ανάθεση τιμής στον κενό κόμβο
  $father = son$! κίνηση κενού κόμβου ...
  $son = min\_value\_son(father)$! ...προς τα φύλλα
enddo
$value(father) = p_n$! επανατοποθέτηση της τιμής p_n στον σωρό

Στο τέλος αυτής της $O(c{\cdot}d)$ διαδικασίας, η ρίζα του αρχικού δένδρου έχει απαλειφθεί και ένα νέο δένδρο με $n-1$ κόμβους που πληροί την διάταξη του σωρού έχει παραχθεί.



Σχήμα 1.9: Απαλοιφή ρίζας σωρού

Παράδειγμα 1.5 Για να επιτευχθεί η απαλοιφή της ρίζας του σωρού του Σχήματος 1.9(α), εξαλείφεται το τελευταίο, σύμφωνα με την πρώτα κατά πλάτος αρίθμηση, φύλλο του δένδρου και η ρίζα εκκενώνεται. Το αποτέλεσμα είναι το δένδρο του Σχήματος 1.9(β). Η τιμή 11 όμως, η οποία προέρχεται από το εξαλειφθέν φύλλο θα πρέπει να επανατοποθετηθεί στον σωρό. Υποψήφια θέση για την επανατοποθέτηση είναι φυσικά η κενή ρίζα του δένδρου. Επειδή όμως τον σωρό διέπει μία συγκεκριμένη διάταξη, η τμή 11 συγκρίνεται με την μικρότερη τιμή των υιών της ρίζας, εδώ το 4. Επειδή $4 < 11$, το 4 καταλαμβάνει την ρίζα και έτσι δημιουργείται ένας νέος κενός κόμβος (Σχήμα 1.9(γ)). Για την κατάληψη του κενού κόμβου ανταγωνίζονται από την μία η τιμή 11 και από την άλλη ο υιός του με την χαμηλότερη τιμή, εδώ 8. Επειδή $8 < 11$, η τιμή 8 καταλαμβάνει τον κενό κόμβο και έτσι δημιουργείται νέος κενός χώρος στην προηγούμενη θέση της τιμής. Ο νέος κενός κόμβος είναι ένα φύλλο. Δεν έχει επομένως υιούς για ν' ανταγωνισθούν την τιμή 11 και έτσι η τελευταία επανατοποθετείται στο κενό φύλλο. Το αποτέλεσμα είναι το δένδρο του Σχήματος 1.9(δ), το οποίο βρίσκεται σε διάταξη σωρού. ■

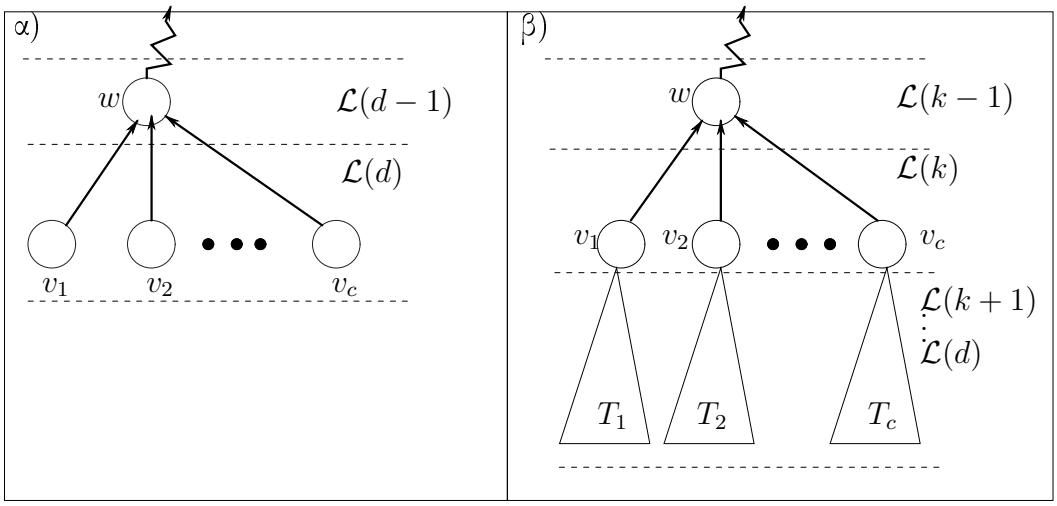
Η δημιουργία σωρού από τα στοιχεία ενός συνόλου $\mathcal{S}$ έχει τις ακόλουθες περιπτώσεις: Εάν το σύνολο είναι κενό, τότε αντιστοιχεί σε κενό δένδρο ($n = 0$) που ασφαλώς ικανοποιεί την διάταξη σωρού. Εάν το σύνολο είναι μονοστοιχειακό, τότε αντιστοιχεί σε δένδρο ενός κόμβου ($n = 1$) που φυσικά ικανοποιεί την διάταξη σωρού. Το μονοστοιχειακό αυτό δένδρο δύναται να θεωρηθεί ότι προέρχεται από το δένδρο του κενού σωρού με την προσθήκη ενός φύλλου που ταυτοχρόνως είναι και ρίζα του.

Εάν το σύνολο $\mathcal{S}$ είναι πολυστοιχειακό υπάρχουν δύο τουλάχιστον τρόποι δημιουργίας ενός αντίστοιχου σωρού. Σύμφωνα με την πρώτη εκδοχή, δημιουργείται ένας μονοστοιχειακός σωρός από κάποιο αυθαίρετο στοιχείο του συνόλου και στην συνέχεια τα υπόλοιπα στοιχεία του συνόλου προστίθενται στον σωρό ένα προς ένα με την διαδικασία ένθεσης που ήδη περιγράψαμε. Δηλαδή για κάθε νέο στοιχείο δημιουργείται ένα νέο κενό φύλλο που ως φυσαλίδα αναδύεται προς την ρίζα. Ας είναι n η πληθύς του συνόλου. Επειδή η πολυπλοκότητα κάθε ανάδυσης είναι $O(d_k)$, όπου $d_k,\ k = 1, 2, \ldots, n$, είναι το βάθος του εκάστοτε δένδρου με k κόμβους, εάν το τελικό αποτέλεσμα από την προσθήκη όλων των στοιχείων του συνόλου είναι ένα δένδρο $\mathcal{H}$ με n κόμβους σε διάταξη σωρού, η πολυπλοκότητα δημιουργίας του είναι $O(\sum_{k=1}^{n} d_k)$. Στην εκδοχή αυτή, η διάταξη σωρού αποκαθίσταται για κάθε νέο στοιχείο που προστίθεται στο δένδρο. Αυτό όμως που ουσιαστικά ενδιαφέρει είναι να έχει το τελικό, ολοκληρωμένο δένδρο διάταξη σωρού.

Η δεύτερη εκδοχή αποσκοπεί στην επίτευξη αυτού ακριβώς του στόχου χωρίς ν' ασχολείται με την επιβολή διάταξης σωρού στα ενδιάμεσα στάδια. Για τον λόγο αυτό, όλα τα στοιχεία του συνόλου οργανώνονται αρχικά σε μορφή δένδρου που δεν πληροί αναγκαστικά διάταξη σωρού και στην συνέχεια, με δεδομένο το ολοκληρωμένο δένδρο, η διάταξη σωρού αποκαθίσταται.

Ας είναι $\mathcal{T}$ το δένδρο με n κόμβους, έναν για κάθε στοιχείο του συνόλου $\mathcal{S}$, και ας είναι d το βάθος του δένδου. Κάθε κόμβος $v \in \mathcal{L}(d)$ είναι τότε φύλλο του δένδρου με πατέρα κάποιον κόμβο $w \in \mathcal{L}(d-1)$. Κάθε κόμβος στην στιβάδα

$\mathcal{L}(d-1)$ είναι επομένως είτε φύλλο του δένδρου είτε πατέρας c το πολύ υιών, $v_1, v_2, \ldots, v_c \in \mathcal{L}(d)$, στην στιβάδα $\mathcal{L}(d)$. Ας είναι v_k ο υιός με την μικρότερη τιμή, δηλαδή $value(v_k) = \min\limits_{v_i \in \{v_1, v_2, \ldots, v_c\}} value(v_i)$. Τότε στο υποδένδρο (Σχήμα 1.10α) με ρίζα τον πατέρα w είτε ισχύει $value(w) \leq value(v_k)$, οπότε πληροίται η διάταξη σωρού, είτε $value(w) > value(v_k)$, οπότε η διάταξη σωρού αποκαθίσταται με την εναλλαγή θέσης μεταξύ του πατέρα w και του υιού v_k. Η διαδικασία αυτή επαναλαμβάνεται για όλους του πατέρες στην στιβάδα $\mathcal{L}(d-1)$ και στην συνέχεια για όλους τους πατέρες στην στιβάδα $\mathcal{L}(d-2)$ κ.ο.κ. έως ότου η διάταξη σωρού επιτευχθεί για τον μοναδικό πατέρα, την ρίζα, του σωρού $\mathcal{L}(0)$. Το Σχήμα 1.10(β) αναπαριστά μία ενδιάμεση κατάσταση. Τα υποδένδρα $\mathcal{T}_1, \ldots, \mathcal{T}_c$ με ρίζες τους υιούς $v_1, \ldots, v_c$ του w είναι ήδη, καθένα χωριστά, σε διάταξη σωρού. Δεν είναι όμως γνωστό εάν το υποδένδρο με ρίζα το w βρίσκεται σε διάταξη σωρού. Ας είναι v_k ο υιός του w με την μικρότερη τιμή, δηλαδή $value(v_k) = \min\limits_{v_i \in \{v_1, v_2, \ldots, v_c\}} value(v_i)$. Τότε εάν $value(w) \leq value(v_k)$, το υποδένδρο με ρίζα το w πληροί την διάταξη σωρού. Εάν όμως $value(w) > value(v_k)$, τότε το v_k καταλαμβάνει την θέση του πατέρα του και η δική του θέση εκκενώνεται. Η κενή θέση, που αντιστοιχεί στην ρίζα του υποδένδρου $\mathcal{T}_k$, διεκδικείται από το w αλλά καταλαμβάνεται απ' αυτόν μόνον εφόσον η διάταξη σωρού διατηρείται για το υποδένδρο $\mathcal{T}_k$. Διαφορετικά η διαδικασία σύγκρισης τιμών επαναλαμβάνεται και η κενή θέση καταδύεται προς τα φύλλα του υποδένδρου $\mathcal{T}_k$ έως ότου γίνει εφικτή η επανατοποθέτηση της τιμής $value(w)$. Για κάθε w που εξετάζεται μ' αυτόν τον τρόπο, η διαδικασία αντιστοιχεί σ' αυτήν της κατάδυσης της φυσαλίδας και είναι επομένως πολυπλοκότητας $O(c \cdot h_w)$, όπου h_w είναι το ύψος του κόμβου w. Επειδή η διαδικασία επαναλαμβάνεται για όλους του κόμβους w πλην των φύλλων του δένδρου, η συνολική πολυπλοκότητα για την αποκατάσταση της διάταξης σωρού στο δένδρο ανέρχεται σε $O(c \cdot \sum\limits_{w \in \mathcal{I}(\mathcal{T})} h_w)$, όπου $\mathcal{I}(\mathcal{T})$ είναι οι κόμβοι του $\mathcal{T}$ που δεν είναι φύλλα.



Σχήμα 1.10: Αποκατάσταση διάταξης σωρού σε δένδρο

Η διαδικασία για την αποκατάσταση της διάταξης σωρού σ' ένα υποδένδρο με ρίζα

w δύναται ν' αποδοθεί με ψευδοκώδικα ανάλογο μ' αυτόν της απαλοιφής ρίζας. Έτσι για δεδομένο δένδρο $\mathcal{T}$ και κόμβο $w \in \mathcal{I}(\mathcal{T})$, ο ακόλουθος ψευδοκώδικας αποκαθιστά την διάταξη σωρού στο υποδένδρο με ρίζα το w:

```
p = value(w) ! Η τιμή του κόμβου w
father = w ! Ο κόμβος w είναι ρίζα υποδένδρου και θεωρείται κενός
son = min_value_son(father) ! Ο υιός με την μικρότερη τιμή
do while( son /= null and p > value(son) )
   value(father) = value(son) ! ανάθεση τιμής στον κενό κόμβο
   father = son ! κίνηση κενού κόμβου ...
   son = min_value_son(father) ! ...προς τα φύλλα
enddo
value(father) = p ! επανατοποθέτηση της τιμής του κόμβου w στον σωρό
```

Η αποκατάσταση της διάταξης σωρού σ' ολόκληρο το δένδρο $\mathcal{T}$ επιτυγχάνεται με την εφαρμογή της διαδικασίας αυτής πρώτα για όλους τους κόμβους $w \in \mathcal{L}(d-1)$, στην συνέχεια για όλους τους κόμβους $w \in \mathcal{L}(d-2)$, ..., $\mathcal{L}(1)$ και τέλος για την ρίζα του $\mathcal{T}$ που είναι ο μοναδικός κόμβος στην στιβάδα $\mathcal{L}(0)$.

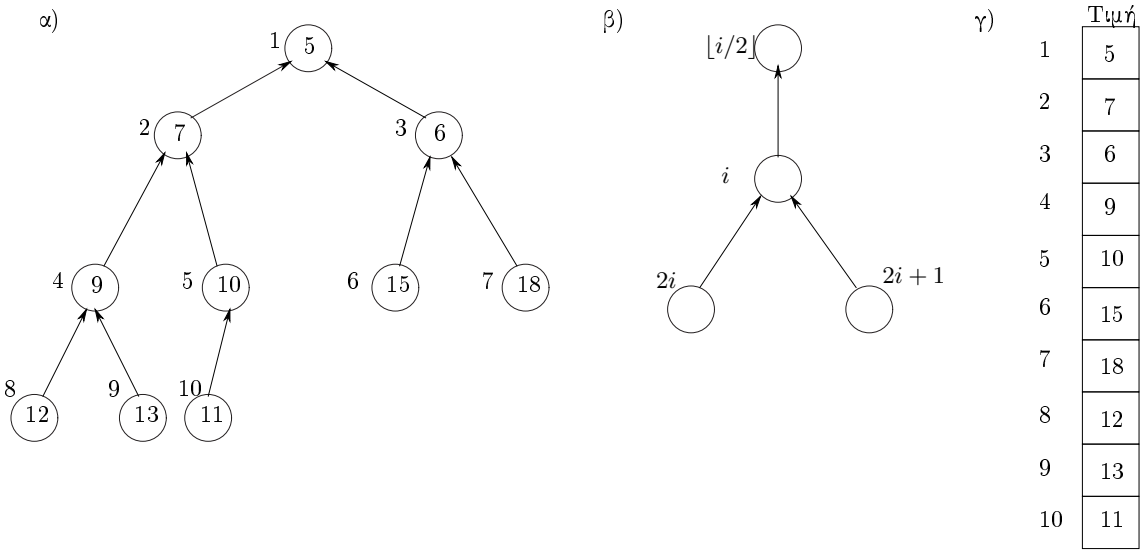
Η αλλαγή της τιμής (προτεραιότητας) ενός στοιχείου του σωρού ελαχίστου αφορά στην μείωση της τιμής του στοιχείου. Σε μία τέτοια περίπτωση, η αλλαγή τιμής δύναται να διαταράξει την διάταξη σωρού, η οποία τότε αποκαθίσταται με την πράξη της ανάδυσης του επηρεασμένου στοιχείου προς την ρίζα του σωρού. Συνεπώς η πολυπλοκότητα του τελεστή μείωσης της τιμής είναι $O(d_x)$, όπου x ο κόμβος και d_x το βάθος του.

1.3.3 Υλοποίηση σωρού με απλή παράταξη ως πλήρες δυαδικό δένδρο

Επειδή το βάθος (και το ύψος) ενός πλήρους δυαδικού δένδρου με n κόμβους είναι $O(\log_2(n))$, η επιλογή παράστασης του σωρού με ένα πλήρες δυαδικό δένδρο οδηγεί σε πολυπλοκότητα $O(\log_2(n))$ τόσο για την ένθεση όσο και για την αλλαγή τιμής, σε πολυπλοκότητα $O(2\log_2(n)) = O(\log_2(n))$ για την απαλοιφή και σε πολυπλοκότητα $O(n)$ για την δημιουργία του σωρού από τα n στοιχεία ενός συνόλου. Η δημιουργία ενός κενού σωρού καθώς και η πρόσβαση στην ελάχιστη τιμή του σωρού παραμένουν $O(1)$ τελεστές.

Η παράσταση του σωρού από ένα πλήρες δυαδικό δένδρο συνεπάγεται ότι η τελική υλοποίηση του σωρού δύναται να επιτευχθεί με απλή γραμμική παράταξη. Όπως είναι δυνατόν να παρατηρηθεί από το Σχήμα 1.11, εφόσον οι κόμβοι του πλήρους δυαδικού δένδρου έχουν αρίθμηση πρώτα κατά πλάτος, ο κόμβος i έχει πατέρα τον κόμβο $\lfloor i/2 \rfloor$, αριστερό υιό τον κόμβο $2i$ και δεξιό υιό τον κόμβο $2i+1$. Η ρίζα, η οποία αντιστοιχεί στον κόμβο 1 δεν έχει πατέρα, τα φύλλα του δένδρου δεν έχουν υιούς, ενώ υπάρχει το ενδεχόμενο να υπάρχει κόμβος, όπως ο κόμβος 5 του σχήματος, με έναν μοναδικό αριστερό υιό. Είναι επομένως δυνατόν να υλοποιηθεί ένας σωρός n στοιχείων σε μορφή πλήρους δυαδικού δένδρου από μία γραμμική παράταξη (διάνυσμα) n στοιχείων, όπου η θέση i, $(1 \leq i \leq n)$, της παράταξης

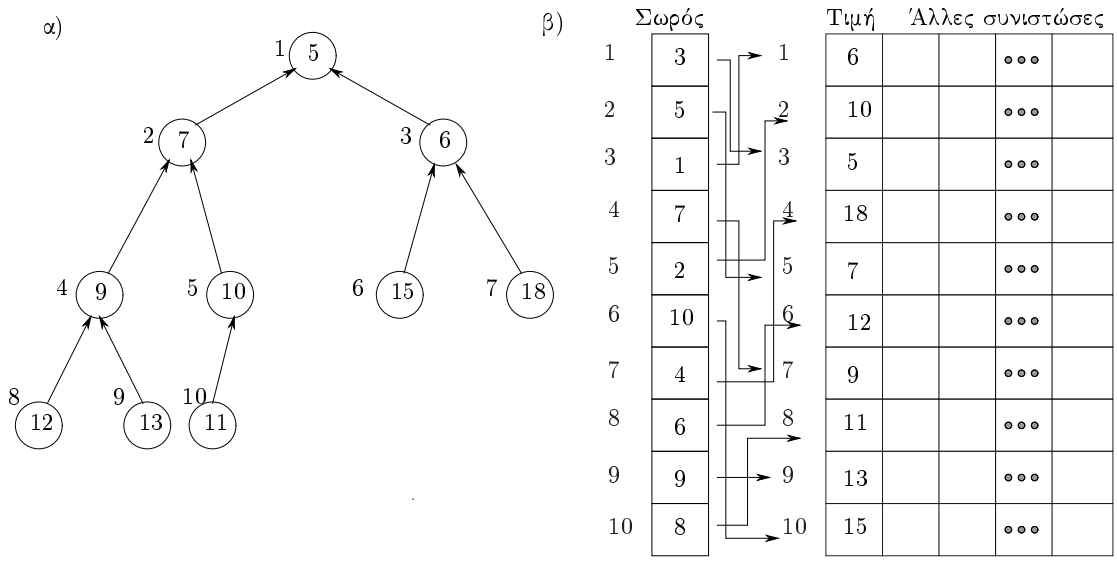
αντιστοιχεί στον κόμβο i του δυαδικού δένδρου. Η θέση $\lfloor i/2 \rfloor$, εφόσον $1 \leq \lfloor i/2 \rfloor \leq n$, αντιστοιχεί στον πατέρα του κόμβου i, η θέση $2i$, εφόσον $1 \leq 2i \leq n$, αντιστοιχεί στον αριστερό υιό του κόμβου i, ενώ η θέση $2i+1$, εφόσον $1 \leq 2i+1 \leq n$, αντιστοιχεί στον δεξιό υιό του κόμβου i



Σχήμα 1.11: Υλοποίηση με γραμμική παράταξη ενός σωρού σε μορφή πλήρους δυαδικού δένδρου

Επειδή η τιμή που καθορίζει την προτεραιότητα και την διάταξη σωρού ενδέχεται ν' αποτελεί συνιστώσα ενός σύνθετου αντικειμένου, οι πράξεις της ανάδυσης και της κατάδυσης των τελεστών του σωρού θα απέβαιναν μη-αποδοτικές εάν συνεπάγονταν εναλλαγές θέσεων σύνθετων αντικειμένων. Για την αποφυγή ενός τέτοιου εκφυλισμού της πολυπλοκότητας των τελεστών, η υλοποίηση του σωρού γίνεται με γραμμική παράταξη ακέραιων ψευδοδεικτών που καταδεικνύουν την θέση της τιμής προτεραιότητας σε μία άλλη γραμμική παράταξη σύνθετων αντικειμένων (Σχήμα 1.12). Σε μια τέτοια υλοποίηση, οι πράξεις της ανάδυσης και της κατάδυσης συνεπάγονται εναλλαγές θέσεων μεταξύ των ακεραίων ψευδοδεικτών ενώ τα σύνθετα αντικείμενα παραμένουν στις θέσεις που ήδη κατέχουν. Έτσι κάθε εναλλαγή θέσης παραμένει πολυπλοκότητας $O(1)$.

Η υλοποίηση του σωρού μπορεί να βασισθεί όχι μόνον σε πλήρη δυαδικά δένδρα αλλά σε πλήρη t-δικά δένδρα, τα οποία είναι δένδρα των οποίων οι κόμβοι έχουν $t \geq 2$ το πολύ υιούς. Και αυτά τα δένδρα δύνανται να παρασταθούν με γραμμική παράταξη διότι ο κόμβος i έχει τον πατέρα του στην θέση $\lceil (i-1)/t \rceil$ και τους υιούς του στις θέσεις $\{(i-1)t+2, \ldots, \min\{i \cdot t+1, n\}\}$. Π.χ. για τριαδικό δένδρο ($t=3$), ο κόμβος i έχει τον πατέρα του στην θέση $\lceil (i-1)/3 \rceil$ και τους υιούς του στις θέσεις $3(i-1)+2, 3(i-1)+3$ και $3(i-1)+4$, εφόσον βεβαίως δεν υπερβαίνουν το n. Οι πολυπλοκότητες των τελεστών λαμβάνονται με βάση το t. Έτσι η ένθεση, όπως και η αλλαγή τιμής, είναι πολυπλοκότητας $O(\log_t(n))$, και στην περίπτωση τριαδικού δένδρου είναι $O(\log_3(n))$, η απαλοιφή είναι πολυπλοκότητας $O(t \log_t(n))$, και στην περίπτωση τριαδικού δένδρου είναι $O(3 \log_3(n)) = O(\log_3(n))$, ενώ η δημιουργία



Σχήμα 1.12: Υλοποίηση με γραμμική παράταξη ψευδοδεικτών ενός σωρού σε μορφή πλήρους δυαδικού δένδρου

σωρού από τα n στοιχεία ενός συνόλου παραμένει πολυπλοκότητας $O(n)$.

1.3.4 Αύξουσα ή φθίνουσα διάταξη n τιμών (Heapsort)

Ο σωρός δύναται να χρησιμοποιηθεί στην αύξουσα ή φθίνουσα διάταξη n τιμών με την χρήση των τελεστών ένθεσης και απαλοιφής της ρίζας ως εξής:

1. Ο τελεστής ένθεσης εφαρμόζεται σε κάθε ένα από τις n τιμές με αποτέλεσμα έναν σωρό n στοιχείων.

2. Ο τελεστής απαλοιφής της ρίζας εφαρμόζεται στον σωρό έως ότου επιτευχθεί η απαλοιφή όλων των στοιχείων του. Το αποτέλεσμα είναι η διάταξη των τιμών.

Βέβαια, σύμφωνα με όσα ελέχθησαν προηγουμένως, μία πιο αποδοτική διαδικασία θα ήταν η εξής:

1. Δημιουργία ενός $t-$δικού δένδρου από τις n τιμές.

2. Αποκατάσταση της διάταξης σωρού στο $t-$δικό δένδρο.

3. Εφαρμογή του τελεστή απαλοιφής της ρίζας n φορές. Το αποτέλεσμα είναι η διάταξη των τιμών.

Το πρώτο και το δεύτερο βήμα είναι πολυπλοκότητας $O(n)$ αντίστοιχα. Το τρίτο βήμα είναι πολυπλοκότητας $O(n \log_t(n))$. Συνεπώς η διάταξη n τιμών είναι ολικής πολυπλοκότητας $O(n \log_t(n))$. Η διαδικασία δεν απαιτεί πρόσθετη μνήμη πέραν απ'

αυτήν που ήδη είναι διαθέσιμη από το διάνυσμα του σωρού: η εκάστοτε απαλοιφή ρίζας δημιουργεί το απαιτούμενο κενό στο διάνυσμα για την κατάληψή του από την τιμή της ρίζας. Στο τέλος της διαδικασίας οι τιμές του διανύσματος είναι σε σωστή διάταξη.