Δομές Δεδομένων

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών
ΠΛΗΡΟΦΟΡΙΚΗ

Θεματική Ενότητα
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Τόμος Γ'

Δομές Δεδομένων

ΙΩΑΝΝΗΣ ΧΑΤΖΗΛΥΓΕΡΟΥΔΗΣ

Δρ. Πληροφορικής

Καθηγητής Β΄θμιας Εκπαίδευσης

ΠΑΤΡΑ 2000

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών
ΠΛΗΡΟΦΟΡΙΚΗ

Θεματική Ενότητα
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Τόμος Γ'
Δομές Δεδομένων

Συγγραφή
ΙΩΑΝΝΗΣ ΧΑΤΖΗΛΥΓΕΡΟΥΔΗΣ
Δρ. Πληροφορικής, Καθηγητής Β΄θμιας Εκπαίδευσης

Κριτική Ανάγνωση
ΕΛΠΙΔΑ ΚΕΡΑΥΝΟΥ
Καθηγήτρια Τμήματος Πληροφορικής Πανεπιστημίου Κύπρου

Ακαδημαϊκός Υπεύθυνος για την επιστημονική επιμέλεια του τόμου
ΠΑΝΑΓΙΩΤΗΣ ΠΙΝΤΕΛΑΣ
Καθηγητής Τμήματος Μαθηματικών Πανεπιστημίου Πατρών

Επιμέλεια στη μέθοδο της εκπαίδευσης από απόσταση
ΓΕΡΑΣΙΜΟΣ ΚΟΥΣΤΟΥΡΑΚΗΣ

Γλωσσική Επιμέλεια
ΙΩΑΝΝΗΣ ΘΕΟΦΙΛΑΣ

Τεχνική Επιμέλεια
ΕΣΠΙ ΕΚΔΟΤΙΚΗ Ε.Π.Ε.

Καλλιτεχνική Επιμέλεια
TYPORAMA

Σελιδοποίηση
ΛΥΔΙΑ ΑΠΟΣΤΟΛΑΚΗ

Συντονισμός ανάπτυξης εκπαιδευτικού υλικού και γενική επιμέλεια των εκδόσεων
ΟΜΑΔΑ ΕΚΤΕΛΕΣΗΣ ΕΡΓΟΥ ΕΑΠ / 1997–1999

ISBN: 960–538–064–1
Κωδικός Έκδοσης: ΠΛΗ 10/3

Περιεχόμενα

Πρόλογος

Ο τόμος αυτός γράφτηκε ως εκπαιδευτικό βοήθημα για τους φοιτητές της Θεματικής Ενότητας «Εισαγωγή στην Πληροφορική» του Προγράμματος Σπουδών «Πληροφορική» του Ε.Α.Π. Γι' αυτό, γράφτηκε ακολουθώντας κάποιο πρότυπο κατάλληλο για ανοικτή εξ αποστάσεως εκπαίδευση.

Ένα χαρακτηριστικό αυτού του τρόπου συγγραφής είναι το γεγονός ότι για το ίδιο τμήμα ύλης απαιτείται μεγαλύτερος συγγραφικός χώρος απ' ό,τι για το συμβατικό τρόπο. Αυτό συμβαίνει διότι η παρουσίαση της ύλης είναι πιο αναλυτική, μιας και πρέπει να εξασφαλίζει σ' ένα μέτρο την χωρίς δάσκαλο μάθηση.

Οι «Δομές Δεδομένων» είναι ένα από τα βασικότερα πεδία της Επιστήμης των Υπολογιστών. Η γνώση των διαθέσιμων δομών δεδομένων είναι απαραίτητη για την ανάπτυξη αποδοτικών προγραμμάτων στον Η/Υ. Το πεδίο αυτό ασχολείται, αφ' ενός μεν με τους τρόπους με τους οποίους μπορούμε να αποθηκεύουμε πληροφορίες στον Η/Υ, αφ' ετέρου δε με τους τρόπους διαχείρισης αυτών των πληροφοριών. Έτσι, η διαπραγμάτευση κάθε δομής δεδομένων στον τόμο αυτό γίνεται ως προς δύο άξονες: το *οργανωτικό σχήμα* της δομής, δηλαδή τον τρόπο απεικόνισης της πληροφορίας στη μνήμη του Η/Υ, και τις *πράξεις διαχείρισης* της δομής. Από τις πράξεις διαχείρισης, ιδιαίτερη έμφαση δίνεται στη «διάταξη», ώστε παρουσιάζεται σε ιδιαίτερο κεφάλαιο.

Στενά συνδεδεμένη με τις δομές δεδομένων είναι και η έννοια του αλγορίθμου, δεδομένου ότι αποτελεί το μέσο έκφρασης του τρόπου υλοποίησης των πράξεων διαχείρισης των δομών. Προς το σκοπό αυτό, χρησιμοποιείται όχι μια συγκεκριμένη γλώσσα προγραμματισμού, αλλά μια απλούστερη αλγοριθμική γλώσσα, η οποία ορίζεται στον τόμο αυτό. Έτσι, η *διαπραγμάτευση* γίνεται *ανεξάρτητη από γλώσσα προγραμματισμού*.

Υπάρχουν αρκετές κατηγορίες δομών δεδομένων και θέματα που σχετίζονται με αυτές. Στον τόμο αυτό αναφερόμαστε σε όλες τις κατηγορίες, εκτός από τους «γράφους» και τα «αρχεία». Αυτό έγινε διότι, αφ' ενός μεν κρίθηκε ότι αυτές οι κατηγορίες δομών δεν είναι απαραίτητες για μια εισαγωγική Θεματική Ενότητα, αφ' ετέρου δε δεν θα ήταν δυνατή η ανάπτυξή τους και λόγω στενότητας του διαθέσιμου συγ-

γραφικού χώρου. Γι' αυτό τον δεύτερο λόγο, επίσης, η διαπραγμάτευση των δομών και των αντίστοιχων θεμάτων δεν γίνεται παντού με το ίδιο βάθος και πλάτος. Τις ελλείψεις αυτές προσπαθεί να καλύψει, σ' ένα μέτρο, ο «Οδηγός για Περαιτέρω Μελέτη», προτείνοντας άλλες πηγές αναφοράς για διάφορα θέματα, στο τέλος κάθε κεφαλαίου. Επιπλέον, η στενότητα του χώρου, δεν επέτρεψε την παράθεση περισσότερων ασκήσεων και δραστηριοτήτων. Πιστεύουμε όμως ότι στον τόμο αυτό καλύπτονται όλα εκείνα που είναι απαραίτητα για ένα εισαγωγικού τύπου βοήθημα στις δομές δεδομένων, όπως είναι το παρόν.

Στο κεφάλαιο 1 παρουσιάζονται διάφορα γενικά και εισαγωγικά θέματα, καθώς και η «φιλοσοφία» με την οποία θεωρούνται οι δομές δεδομένων, σαν προετοιμασία του αναγνώστη για τα επόμενα κεφάλαια. Το κεφάλαιο 2 διαπραγματεύεται αναλυτικά τους πίνακες και λιγότερο αναλυτικά τις εγγραφές, τις δύο θεμελιώδεις δομές δεδομένων. Στο κεφάλαιο 3 παρουσιάζονται οι δύο βασικοί τύποι γενικών λιστών, οι συνεχόμενες και οι συνδεδεμένες λίστες, ενώ στο κεφάλαιο 4 οι δύο βασικοί τύποι ειδικών λιστών, οι στοίβες και οι ουρές. Το κεφάλαιο 5 διαπραγματεύεται κυρίως τα δυαδικά δένδρα, τη σπουδαιότερη κατηγορία δένδρων, και τις παραλλαγές τους. Τέλος, το κεφάλαιο 6 ασχολείται με την πράξη της διάταξης, παρουσιάζοντας τρεις αντιπροσωπευτικούς αλγορίθμους διάταξης.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον *Καθηγητή Παναγιώτη Πιντέλα* για την θεματική καθοδήγηση, την *Καθηγήτρια Ελπίδα Κεραυνού* για την εποικοδομητική κριτική ανάγνωση του κειμένου, τον *κ. Γεράσιμο Κουστουράκη* για την καθοδήγηση στην εφαρμογή των αρχών της μεθόδου της εκπαίδευσης από απόσταση και τον *κ. Χρήστο Παναγιωτακόπουλο* για την άψογη συνεργασία και βοήθεια για τη διεκπεραίωση του έργου της συγγραφής. Επίσης, τον *Καθηγητή Αθανάσιο Τσακαλίδη* για τη στήριξη στην ανάληψη του έργου. Τέλος, όλους όσοι βοήθησαν με κάποιο τρόπο στην παραγωγή αυτού του τόμου.

Ιωάννης Κ. Χατζηλυγερούδης
Ιούλιος 1999

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Σκοπός

Ο σκοπός του κεφαλαίου αυτού είναι να σας παρουσιάσει τα απαραίτητα στοιχεία υποδομής που θα χρειαστείτε για τη μελέτη των επόμενων κεφαλαίων, να δώσει τους γενικούς ορισμούς βασικών εννοιών, την αντίστοιχη ορολογία και τα όρια στα οποία θα κινηθεί η παρουσίαση των δομών δεδομένων στα επόμενα κεφάλαια.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό, θα είστε σε θέση να:

- *εξηγείτε τη διαφορά μεταξύ ενός ατομικού τύπου δεδομένων και μιας δομής δεδομένων και να αναφέρετε τουλάχιστον ένα παράδειγμα από το καθένα,*
- *κατατάσσετε τις βασικές δομές δεδομένων στις αντίστοιχες κατηγορίες,*
- *σχεδιάζετε απλούς αλγορίθμους στην αλγοριθμική μας γλώσσα από την περιγραφή τους σε φυσική γλώσσα,*
- *να βρίσκετε τη συνάρτηση πολυπλοκότητας απλών αλγορίθμων,*
- *βρίσκετε την τάξη μεγέθους μιας συνάρτησης πολυπλοκότητας από την αναλυτική της έκφραση.*

Έννοιες κλειδιά

- *Δεδομένα*
- *Μεταβλητή*
- *Τύπος δεδομένων*
- *Δομή δεδομένων*
- *Αφηρημένος τύπος δεδομένων*
- *Κατηγορίες δομών δεδομένων*
- *Υλοποίηση δομών δεδομένων*
- *Αναπαράσταση δομών δεδομένων*
- *Πράξεις στις δομές δεδομένων*
- *Αλγόριθμος*
- *Γλώσσα ψευδοκώδικα*

- *Αποδοτικότητα αλγορίθμου*
- *Πολυπλοκότητα αλγορίθμου*
- *Συνάρτηση πολυπλοκότητας.*

Ενότητες Κεφαλαίου 1

1.1 Ορισμοί και ορολογία

1.2 Αλγοριθμική γλώσσα

1.3 Πολυπλοκότητα αλγορίθμων

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό κατ' αρχήν εξηγούνται οι έννοιες 'δεδομένα', 'μεταβλητή', 'τύπος δεδομένων', 'δομή δεδομένων', 'αφηρημένη δομή δεδομένων' και δίνονται, όπου είναι απαραίτητο, γενικοί ορισμοί. Στη συνέχεια παρουσιάζονται οι διάφορες κατηγορίες δομών δεδομένων και οι πτυχές της υλοποίησης μιας δομής δεδομένων, καθώς και ο τρόπος αναπαράστασης της μνήμης ενός Η/Υ. Τα παραπάνω αποτελούν το περιεχόμενο της πρώτης ενότητας.

Στη δεύτερη ενότητα παρουσιάζεται κατά βάση η γλώσσα ψευδοκώδικα που χρησιμοποιούμε για την περιγραφή (υλοποίηση) των αλγορίθμων σε όλα τα επόμενα κεφάλαια. Σκοπός της ενότητας αυτής δεν είναι να μάθετε να σχεδιάζετε πολύπλοκους αλγορίθμους με τη γλώσσα αυτή, αλλά απλώς να εξοικειωθείτε με τη γλώσσα ώστε να μπορείτε να μελετάτε τους αλγορίθμους που παρουσιάζονται στα επόμενα κεφάλαια. Γι' αυτό και τα παραδείγματα και οι ασκήσεις αναφέρονται σε απλούς αλγορίθμους.

Τέλος, στην τρίτη ενότητα γίνεται μια συζήτηση για την έννοια της αποδοτικότητας (ή πολυπλοκότητας) ενός αλγορίθμου και ο τρόπος του ποσοτικού προσδιορισμού της. Στόχος της ενότητας αυτής δεν είναι να μάθετε να αναλύετε την πολυπλοκότητα δύσκολων αλγορίθμων, αλλά να εξοικειωθείτε με τους συμβολισμούς και τον τρόπο χειρισμού της πολυπλοκότητας απλών αλγορίθμων, ώστε να είστε σε θέση να κατανοήσετε τις διαπραγματεύσεις πολυπλοκότητας των αλγορίθμων που παρουσιάζονται στα επόμενα κεφάλαια.

Το παρόν κεφάλαιο περιλαμβάνει κάποια γενικά στοιχεία και έννοιες που ίσως να μη γίνουν κατ' αρχήν πλήρως κατανοητές. Αυτό δεν πρέπει να σας απογοητεύσει, διότι η πλήρης κατανόηση αυτών των σημεί-

ων θα γίνει αργότερα, καθώς θα προχωράτε στη μελέτη των επόμενων κεφαλαίων που αναφέρονται σε πιο συγκεκριμένα πράγματα. Θα χρειαστεί τότε να επιστρέψετε αρκετές φορές στο κεφάλαιο αυτό που θα λειτουργεί σαν σημείο αναφοράς.

Στο κεφάλαιο αυτό, αλλά και σε επόμενα, χρησιμοποιούμε μερικά παραδείγματα από τις γλώσσες προγραμματισμού Pascal και C. Δεν απαιτείται γνώση της Pascal ή της C για την κατανόηση αυτών των παραδειγμάτων ούτε τα παραδείγματα αυτά δίνονται για να μάθετε Pascal ή C. Αποτελούν απλώς επίδειξη του πως ορίζονται κάποιες δομές, μεταβλητές και πως υλοποιούνται κάποιοι αλγόριθμοι σε πραγματικές γλώσσες.

1.1 Ορισμοί και ορολογία

1.1.1 Δεδομένα και προγράμματα

Ο όρος 'δεδομένα' στην επιστήμη των υπολογιστών είναι στενά συνδεδεμένος με τα προγράμματα ηλεκτρονικών υπολογιστών (Η/Υ). Όπως είναι γνωστόν, ένα *πρόγραμμα* αποτελείται κατά βάση από μια ακολουθία εντολών, διατυπωμένων σε μια γλώσσα προγραμματισμού, για διαδοχική εκτέλεση από τον Η/Υ. Κατά την εκτέλεση ενός προγράμματος γίνεται *επεξεργασία* κάποιων αρχικών (ακατέργαστων) πληροφοριών σχετικών με κάποιο πρόβλημα, που ονομάζονται *δεδομένα*, και παράγονται τελικά πληροφορίες πιο χρήσιμες, που αποτελούν τη *λύση* για το συγκεκριμένο πρόβλημα, και ονομάζονται *αποτελέσματα*. Κατά την εκτέλεση ενός προγράμματος παράγονται επίσης κάποιες ενδιάμεσες πληροφορίες, που ονομάζονται *ενδιάμεσα αποτελέσματα* και χρειάζεται να αποθηκευθούν προσωρινά ώστε να χρησιμοποιηθούν για την παραγωγή των (τελικών) αποτελεσμάτων. Όλες αυτές οι «μορφές» πληροφορίας έχει επικρατήσει να αποδίδονται με τον όρο *δεδομένα* (*data*). ■

■ Τα **δεδομένα** είναι τιμές ή πλειάδες τιμών που σχετίζονται με κάποιες οντότητες.

Τα δεδομένα χρησιμοποιούνται σ' ένα πρόγραμμα κυρίως μέσω υπολογιστικών οντοτήτων που ονομάζονται *μεταβλητές* (*variables*). Εννοιολογικά, μια μεταβλητή αντιπροσωπεύει ένα σύνολο δεδομένων που σχετίζονται με μια οντότητα, συγκεκριμένη ή αφηρημένη. Σε φυσικό επίπεδο όμως αντιπροσωπεύει ένα χώρο στη μνήμη του Η/Υ. (Μια πιο συγκεκριμένη αναφορά στο ζήτημα αυτό γίνεται στην υποενότητα «1.1.5 Υλοποίηση Δομών Δεδομένων»).

Παράδειγμα 1.1 *Παραδείγματα μεταβλητών-δεδομένων*

Σ' ένα πρόγραμμα, που ασχολείται με την κίνηση αντικειμένων, χρειάζεται να παραστήσουμε δεδομένα που αφορούν, μεταξύ άλλων, διανυόμενα διαστήματα και θέσεις κινητών ως προς σύστημα δύο ορθογωνίων αξόνων. Για το σκοπό αυτό χρησιμοποιούμε δύο μεταβλητές, SPACE και LOCATION. Η πρώτη αντιπροσωπεύει κάθε φορά μια (διαφορετική) τιμή της οντότητας 'διάστημα'. Λέμε ότι η μεταβλητή SPACE αντιπροσωπεύει δεδομένα (ή παίρνει τιμές) που εκφράζουν διαστήματα. Οι τιμές αυτές είναι απλές τιμές. Η μεταβλητή LOCATION αντιπροσωπεύει κάθε φορά μια τιμή της οντότητας 'θέση κινητού', που προσδιορίζεται από τις δύο συντεταγμένες της, x και y. Επομένως, η μεταβλητή αυτή αντιπροσωπεύει δεδομένα (τιμές) που εκφράζουν θέσεις κινητού, και είναι δυάδες τιμών.

1.1.2 Τύποι και δομές δεδομένων

Τα δεδομένα, δηλαδή οι μεταβλητές που τα αντιπροσωπεύουν σ' ένα πρόγραμμα, χαρακτηρίζονται από κάποιους τύπους δεδομένων. Οι τύποι δεδομένων αποτελούν τρόπους παράστασης και χρήσης των δεδομένων. Πιο συγκεκριμένα, ένας *τύπος δεδομένων* (*data type*) καθορίζει τις τιμές που επιτρέπεται να πάρει μια μεταβλητή, δηλαδή το *πεδίο τιμών* (*domain*) της, καθώς και τους δυνατούς τρόπους επεξεργασίας ή διαχείρισης των τιμών αυτών, που τους ονομάζουμε *πράξεις* (*operations*)[1]. ■

■ Ένας **τύπος δεδομένων** ορίζεται από α) ένα σύνολο (πεδίο) τιμών και β) ένα σύνολο τρόπων επεξεργασίας ή διαχείρισης των τιμών (πράξεων).

Οι τύποι δεδομένων διακρίνονται σε ατομικούς ή πρωτογενείς τύπους και σε δομημένους τύπους. Οι *ατομικοί τύποι* (*atomic types*) είναι αυτοί που οι τιμές τους είναι απλές, δηλαδή δεν αποτελούνται από άλλες επί μέρους τιμές και επομένως δεν μπορούν να αναλυθούν περαιτέρω. Μια απλή τιμή θεωρείται και χρησιμοποιείται ως ένα όλον. Τέτοιες είναι οι τιμές της μεταβλητής SPACE στο Παράδειγμα 1.1. Ατομικοί τύποι είναι οι βασικοί τύποι δεδομένων που υπάρχουν σε όλες τις γλώσσες υψηλού επιπέδου: ακέραιος (integer), πραγματικός (real), λογικός (boolean), χαρακτήρας (character).

[1] Στην ελληνική βιβλιογραφία χρησιμοποιείται πολλές φορές ο όρος «λειτουργίες» αντί «πράξεις».

Παράδειγμα 1.2

Παραδείγματα τύπων δεδομένων στη γλώσσα Pascal

Η γλώσσα Pascal διαθέτει όλους τους παραπάνω τύπους δεδομένων. Δύο από αυτούς παρουσιάζονται στον παρακάτω πίνακα με τα αντίστοιχα πεδία τιμών τους και τις επιτρεπόμενες πράξεις (με αντίστοιχους τελεστές σε παρένθεση) σαν παραδείγματα στον παραπάνω ορισμό.

Τύπος Δεδομένων	Πεδίο Τιμών	Πράξεις
integer (ακέραιος)	Το διάστημα ακεραίων [-maxint, maxint], όπου το maxint εξαρτάται από την υλοποίηση της γλώσσας[2].	Καταχώρηση[3] (:=) Αριθμητικές (+, -, *, div,…) Σύγκρισης (=, <>, <, >,...)
boolean (λογικός)	(false, true)	Καταχώρηση (:=) Λογικές (and, or, not)

Ο τύπος των μεταβλητών στην Pascal δηλώνεται ρητά στο πρόγραμμα. Π.χ. για τη μεταβλητή SPACE του Παραδείγματος 1.1 χρησιμοποιούμε την εξής έκφραση:

var SPACE **:** real **;**

όπου δηλώνεται μια μεταβλητή[4] (var) με όνομα SPACE, η οποία είναι (:) πραγματικού τύπου (real). Το σύμβολο ‘;’ αποτελεί διαχωριστικό των εκφράσεων.

Οι *δομημένοι τύποι* (*structured types*) είναι αυτοί των οποίων οι τιμές είναι σύνθετες, δηλαδή αποτελούνται από επί μέρους τιμές, που καλούνται *στοιχεία* (*elements*) ή *κόμβοι* (*nodes*), και σχετίζονται μεταξύ τους με βάση κάποιο σχήμα, έχουν δηλαδή κάποια ‘δομή’, στην οποία θα αναφερόμαστε με τον όρο *οργανωτικό σχήμα*. Το οργανωτικό σχήμα μιας δομής έχει σχέση με τον τρόπο παράστασης της δομής

[2] Όταν λέμε υλοποίηση μιας γλώσσας εννοούμε την συγκεκριμένη έκδοση της γλώσσας, που σχετίζεται με τον τύπο του Η/Υ που απευθύνεται (π.χ. προσωπικός Η/Υ, μεγάλος Η/Υ). Π.χ. η Turbo Pascal είναι μια έκδοση της Pascal για προσωπικούς Η/Υ.

[3] Στην ελληνική βιβλιογραφία συναντάται και ως ‘Ανάθεση’ ή ‘Εκχώρηση’.

[4] Στις εντολές της Pascal η έντονη γραφή αφορά λέξεις ή σύμβολα της Pascal που παραμένουν αμετάβλητα.

■ Μια **δομή δεδομένων** είναι ένας τύπος δεδομένων που α) το πεδίο τιμών του αποτελείται από σύνθετες τιμές, δηλαδή τιμές που συντίθενται από άλλες επί μέρους τιμές (στοιχεία ή κόμβοι), που είναι είτε απλές είτε σύνθετες και β) έχει επί πλέον ένα σύνολο σχέσεων (οργανωτικό σχήμα) μεταξύ των στοιχείων/ κόμβων κάθε τιμής.

στη μνήμη του Η/Υ. (Στο ζήτημα της αναπαράστασης της μνήμης ενός Η/Υ αναφερόμαστε στην υποενότητα «1.1.5 Υλοποίηση Δομών Δεδομένων»). Κάθε στοιχείο/κόμβος μιας δομής χαρακτηρίζεται από κάποιον τύπο δεδομένων. Οι δομημένοι τύποι δεδομένων συνήθως αποκαλούνται *δομές δεδομένων* (*data structures*). Το πλήθος των στοιχείων/κόμβων μιας δομής ονομάζεται *μέγεθος* (*size*) της δομής. ■

Μια δομή δεδομένων δηλαδή είναι ένα οργανωμένο σύνολο ατομικών τύπων δεδομένων ή άλλων δομών δεδομένων.

Παράδειγμα 1.3

Παράδειγμα απλής δομής δεδομένων στην Pascal

Ο *μονοδιάστατος πίνακας* (*array*) είναι η πιο απλή δομή δεδομένων και υπάρχει διαθέσιμη σε όλες σχεδόν τις γλώσσες υψηλού επιπέδου. Ένας πίνακας[5] είναι μια συλλογή ενός πεπερασμένου αριθμού στοιχείων που βρίσκονται σε μια σειρά. Η μεταβλητή LOCATION στο Παράδειγμα 1.1 θα μπορούσε να είναι τέτοιας δομής, πιο συγκεκριμένα ένας πίνακας με δύο στοιχεία πραγματικού τύπου, όπου το πρώτο αντιπροσωπεύει την τετμημένη (x) και το δεύτερο την τεταγμένη (y) της θέσης κινητού.

Στη γλώσσα Pascal αυτό υλοποιείται ως εξής:

```
type vec = array [1..2] of real ;
var LOCATION : vec ;
```

Με την πρώτη έκφραση ορίζουμε ένα τύπο (type) δεδομένων με όνομα vec, που είναι (=) ένας πίνακας (array) δύο στοιχείων ([1..2]) πραγματικού τύπου (real)[6]. Στη δεύτερη δηλώνεται ότι η μεταβλητή LOCATION είναι τύπου vec, που σημαίνει ότι κάθε τιμή της είναι μια δυάδα πραγματικών αριθμών.

---

[5] Με τον όρο ‘πίνακας’ εφεξής θα εννοούμε ‘μονοδιάστατος πίνακας’. Οι πίνακες παρουσιάζονται αναλυτικά στο επόμενο κεφάλαιο.

[6] Μ’ αυτό τον τρόπο μπορούμε να ορίσουμε (θεωρητικά) άπειρους τύπους με δομή πίνακα, οι οποίοι διαφέρουν ως προς τον τύπο και τον αριθμό των στοιχείων τους. Αυτοί οι συγκεκριμένοι τύποι αποτελούν *στιγμιότυπα* (*instances*) της *γενικής δομής* πίνακας. Με τους όρους ‘πίνακας’, ‘πίνακες’, ‘λίστα’, ‘λίστες’ κλπ αναφερόμαστε άλλοτε στα στιγμιότυπα και άλλοτε στις γενικές δομές, ανάλογα με την περίπτωση.

Το *πεδίο τιμών* της δομής vec, δηλαδή της μεταβλητής LOCATION, είναι ένα σύνολο που περιέχει όλες τις δυνατές δυάδες πραγματικών αριθμών (όπου η σειρά στη δυάδα παίζει ρόλο), άπειρες θεωρητικά, πεπερασμένες στην πραγματικότητα, λόγω φυσικών περιορισμών του υλικού του Η/Υ.

Οι κυριότερες *πράξεις* που επιτρέπει η Pascal σ' ένα πίνακα είναι οι εξής:

- *προσπέλαση* (*access*) *στοιχείου* (π.χ. LOCATION[2]): Γίνεται προσπέλαση της τιμής ενός στοιχείου (του 2^{ου} εδώ) μιας μεταβλητής-πίνακα (LOCATION), η οποία μπορεί να αποδοθεί στη συνέχεια σε μια μεταβλητή ίδιου τύπου με το στοιχείο (π.χ. SPACE**:=** LOCATION[2]). Η πράξη αυτή συνήθως καλείται και *ανάκτηση* (*retrieval*) *στοιχείου*. Το '2' είναι ένας *δείκτης* που δείχνει σε πιο κατά σειρά στοιχείο αναφερόμαστε. Εν γένει το A[K], όπου A μεταβλητή τύπου πίνακα και K δείκτης, λειτουργεί σαν μεταβλητή.
- *καταχώρηση* (*assignment*) *στοιχείου* (π.χ. LOCATION[2] **:=** SPACE, LOCATION[1] **:=** 2.5): Γίνεται καταχώρηση (ή απόδοση) της τιμής μιας μεταβλητής (SPACE) ή μιας σταθερής τιμής (2.5) σ' ένα στοιχείο του πίνακα. Όταν το στοιχείο έχει ήδη κάποια τιμή και του την αλλάζουμε, η πράξη αυτή ονομάζεται τότε *ενημέρωση* (*update*) *στοιχείου*.

1.1.3 Αφηρημένοι τύποι δεδομένων

Στη βιβλιογραφία, μαζί με τους όρους 'τύπος δεδομένων' και 'δομή δεδομένων' συναντάται και ο όρος 'αφηρημένος τύπος δεδομένων'. Για λόγους πληρότητας, θα αναφερθούμε, πολύ συνοπτικά όμως, στον όρο αυτό.

Για κάθε τύπο δεδομένων (μπορεί να) υπάρχει ένας αντίστοιχος μαθηματικός ορισμός που προσδιορίζει με αυστηρό τρόπο τα χαρακτηριστικά του. Το μαθηματικό αυτό μοντέλο χρησιμοποιεί αφηρημένους όρους και έννοιες για την περιγραφή του αντίστοιχου τύπου δεδομένων και δεν αναφέρεται καθόλου σε θέματα υλοποίησης. Ο μαθηματικός αυτός ορισμός ονομάζεται *αφηρημένος τύπος δεδομένων* (*abstract data type*) ή συντομογραφικά ΑΤΔ (ADT).

Η υλοποίηση ενός ΑΤΔ σε συγκεκριμένο λογισμικό (γλώσσα προγραμματισμού) συνιστά ένα τύπο δεδομένων. Ένας ΑΤΔ όμως είναι

δυνατόν να μη μπορεί να υλοποιηθεί σε όλη του τη γενικότητα ή και καθόλου σε κάποιο συγκεκριμένο λογισμικό ή υλικό Η/Υ. Π.χ. ο ΑΤΔ 'ακέραιος' δεν μπορεί να υλοποιηθεί σε όλη του τη γενικότητα λόγω των περιορισμών του υλικού και του λογισμικού των Η/Υ. Θυμηθείτε (Παράδειγμα 1.2) ότι το πεδίο τιμών του στη γλώσσα Pascal είναι το [-maxint, maxint], όπου το maxint εξαρτάται από τη συγκεκριμένη υλοποίηση της γλώσσας, δηλαδή τον μεταφραστή και τον τύπο Η/Υ που απευθύνεται, και όχι το (-∞, ∞) όπως απαιτεί ο αντίστοιχος ΑΤΔ.

Οι ΑΤΔ αποτελούν αυστηρές περιγραφές των ιδιοτήτων των τύπων δεδομένων, χωρίς να εμπλέκουν θέματα υλοποίησης ή αποδοτικότητας. (Το ζήτημα της υλοποίησης των δομών δεδομένων διαπραγματευόμαστε αναλυτικά στην υποενότητα «1.1.5 Υλοποίηση Δομών Δεδομένων», ενώ αυτό της αποδοτικότητας στην ενότητα «1.3 Πολυπλοκότητα Αλγορίθμων»). Είναι δυνατόν να υπάρχουν ΑΤΔ που να μη μπορεί να υλοποιηθούν σε κάποιο λογισμικό ή υλικό, αλλά να είναι χρήσιμοι για την κατανόηση κάποιων εννοιών. Η περιγραφή των ΑΤΔ γίνεται με κάποια «γλώσσα» ή «μέθοδο» και συνήθως συγκροτείται από δύο μέρη, το πρώτο αναφέρεται στον ορισμό του πεδίου τιμών και το δεύτερο στους ορισμούς των πράξεων. Έτσι, οι ΑΤΔ αποτελούν χρήσιμο οδηγό και για υλοποιητές γλωσσών προγραμματισμού, ώστε να γίνεται έλεγχος σωστής υλοποίησης, και για προγραμματιστές, ώστε να γίνεται σωστή χρήση και εκμετάλλευση των δυνατοτήτων των τύπων δεδομένων.

Δεδομένου ότι κύριος στόχος μας στα επόμενα κεφάλαια είναι η παρουσίαση των τρόπων υλοποίησης των δομών δεδομένων και όχι η σχεδίασή τους, δεν θα χρησιμοποιήσουμε ΑΤΔ, με την αυστηρή έννοια. Όμως, για κάθε δομή στα επόμενα κεφάλαια δίνουμε κάποιο μαθηματικό ορισμό ή περιγραφή.

Δραστηριότητα 1.1

Με βάση τον ορισμό της δομής δεδομένων στην υποενότητα «1.1.2 Τύποι και Δομές Δεδομένων», προσδιορίστε πιο καθαρά τα τρία στοιχεία που ορίζουν μια δομή δεδομένων και εξηγήστε τα με δικά σας λόγια χρησιμοποιώντας ενδεχομένως και παραδείγματα. Ποιο από τα στοιχεία αυτά δεν είναι απαραίτητο για τον ορισμό ενός ατομικού τύπου δεδομένων και γιατί; Δώστε την απάντησή σας σε μισή περίπου σελίδα. Για την απάντησή σας μπορεί να βοηθηθείτε από τη σύνοψη στο τέλος αυτού του κεφαλαίου.

1.1.4 Κατηγορίες δομών

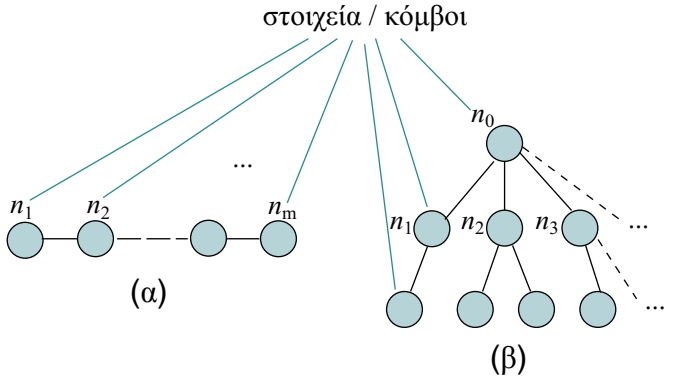
Οι δομές δεδομένων με τις οποίες θα ασχοληθούμε στα επόμενα κεφάλαια είναι οι εξής: *πίνακες*, *εγγραφές*, *γενικές λίστες* (*συνεχόμενες*, *συνδεδεμένες*), *ειδικές λίστες* (*στοίβες*, *ουρές*) και *δέντρα*. Προς το παρόν, μας είναι αρκετό να τις γνωρίσουμε μόνο ονομαστικά και να αναφερθούμε σε ορισμένα γενικά χαρακτηριστικά τους με βάση τα οποία τις κατατάσσουμε σε διάφορες κατηγορίες. Δεν θα ασχοληθούμε με *γραφήματα (ή γράφους)* και *αρχεία*, δύο άλλες δομές δεδομένων, διότι αποτελούν αντικείμενα άλλων θεματικών ενοτήτων.

Οι *πίνακες* και οι *εγγραφές* είναι οι πιο απλές δομές δεδομένων και υπάρχουν συνήθως ενσωματωμένες στις γλώσσες υψηλού επιπέδου. Οι δομές αυτές ονομάζονται *θεμελιώδεις δομές* (*fundamental structures*). Οι θεμελιώδεις δομές δεδομένων συνήθως προσδιορίζονται στατικά μέσα σ' ένα πρόγραμμα, δηλαδή το μέγεθός τους και το οργανωτικό σχήμα τους παραμένουν αμετάβλητα στη διάρκεια εκτέλεσης ενός προγράμματος, γι' αυτό ονομάζονται και *στατικές δομές* (*static structures*).

Όμως, για τη λύση πολλών προβλημάτων απαιτούνται πιο πολύπλοκες δομές, που θεωρούνται *ανώτερες δομές* (*higher-level structures*). Οι ανώτερες δομές δεδομένων χρησιμοποιούν τις θεμελιώδεις δομές για την υλοποίησή τους και συνήθως προσδιορίζονται δυναμικά σ' ένα πρόγραμμα, δηλαδή το μέγεθός τους μεταβάλλεται κατά την εκτέλεση του προγράμματος, γι' αυτό αποκαλούνται και *δυναμικές δομές* (*dynamic structures*). Τέτοιες δομές δεδομένων είναι οι διάφορες λίστες (συνεχόμενες, συνδεδεμένες, στοίβες και ουρές) και τα δέντρα.

Μια άλλη διάκριση που αφορά τις δομές δεδομένων είναι σε *γραμμικές* (*linear*) και *μη γραμμικές* (*non-linear*). Μια δομή είναι γραμμική αν το οργανωτικό της σχήμα μπορεί να απεικονισθεί σαν μια γραμμή. Αυτό είναι δυνατόν όταν κάθε στοιχείο σχετίζεται μόνο με ένα άλλο στοιχείο της, υπάρχει δηλαδή μια σχέση ένα-προς-ένα μεταξύ των στοιχείων της. Παρατηρήστε το Σχήμα 1.1α όπου παρουσιάζεται η γενική μορφή μιας γραμμικής δομής, όπου κάθε στοιχείο/κόμβος της δομής σχετίζεται μόνο με ένα άλλο στοιχείο της δομής π.χ. με τη σχέση 'επόμενο'. Αντίθετα, σε μια μη γραμμική δομή υπάρχουν σχέσεις ένα-προς-πολλά ή πολλά-προς-πολλά μετα-

ξύ των στοιχείων της. Στο Σχήμα 1.1β παρουσιάζεται η γενική μορφή μιας κατηγορίας μη γραμμικών δομών, των δέντρων. Παρατηρήστε ότι κάθε κόμβος δυνατόν να σχετίζεται με περισσότερους του ενός άλλους κόμβους (σχέση ένα-προς-πολλά) π.χ. με τη σχέση 'παιδιά'. Γραμμικές δομές είναι οι πίνακες και οι λίστες, ενώ μη γραμμικές οι εγγραφές και τα δέντρα.



Σχήμα 1.1

Οργανωτικό σχήμα (α) γραμμικής (β) μη γραμμικής (δεντρικής) δομής.

1.1.5 Υλοποίηση δομών δεδομένων

Οι ανώτερες δομές δεν υπάρχουν ενσωματωμένες στις γλώσσες προγραμματισμού, αλλά υλοποιούνται χρησιμοποιώντας τις θεμελιώδεις δομές τους. Η υλοποίηση μιας δομής έχει δύο πτυχές. Η μία αφορά στην *αναπαράσταση της δομής* στον Η/Υ, ενώ η άλλη στην *υλοποίηση των πράξεων* στη δομή. Αναπαράσταση μιας δομής σημαίνει την απεικόνιση της δομής στην μνήμη του Η/Υ, δηλαδή τον τρόπο αποθήκευσης των στοιχείων/κόμβων της δομής στη μνήμη του Η/Υ ώστε να ανταποκρίνεται στο οργανωτικό σχήμα της. Συνήθως υπάρχουν περισσότεροι του ενός τρόποι αναπαράστασης μιας δομής δεδομένων.

Όταν μιλάμε για μνήμη Η/Υ, αναφερόμαστε στην *κύρια μνήμη* (*main memory*) του. Η κύρια μνήμη ενός Η/Υ θεωρείται σαν ένα σύνολο από *θέσεις μνήμης* (*memory cells*). Κάθε θέση μνήμης προσδιορίζεται από μια *διεύθυνση* (*address*), που παριστάνεται με ένα θετικό ακέραιο αριθμό. Επίσης οι θέσεις μνήμης χαρακτηρίζονται από το μέγεθός τους, δηλαδή το πλήθος των δυαδικών ψηφίων πληροφορίας που μπορεί να αποθηκεύσει η κάθε μια. Το μέγεθος μιας θέσης μνήμης ονομάζεται *λέξη* (*word*) του Η/Υ. Στα επόμενα, για λόγους απλότητας, θεωρούμε Η/Υ με μέγεθος θέσης μνήμης (λέξη) 1 byte, δηλαδή πληροφορίας 8

δυαδικών ψηφίων (bits) (1 byte = 8 bits). Μία μεταβλητή αντιπροσωπεύει ένα σύνολο θέσεων στη μνήμη, που το πλήθος του εξαρτάται από τον τύπο της μεταβλητής.

Η κύρια μνήμη ενός Η/Υ είναι *μνήμη τυχαίας προσπέλασης* (*RAM: Random Access Memory*). Ο όρος 'τυχαίας προσπέλασης' αναφέρεται στο ότι ο χρόνος που απαιτείται για πρόσβαση σε μια τυχαία επιλεγείσα θέση μνήμης είναι ο ίδιος για όλες τις θέσεις, υπόθεση που δεν ισχύει για δευτερεύουσες μνήμες (π.χ. μαγνητικό δίσκο, μαγνητική ταινία). Στο κεφάλαιο αυτό θα ασχοληθούμε με δομές δεδομένων που έχουν σχέση με την κύρια μνήμη του Η/Υ και όχι με κάποια δευτερεύουσα (όπως π.χ. τα αρχεία).

Υλοποίηση μιας πράξης σε μια δομή δεδομένων σημαίνει την εύρεση του τρόπου πραγματοποίησής της στον Η/Υ με βάση την αναπαράσταση της δομής και τις διαθέσιμες πράξεις των θεμελιωδών δομών και των ατομικών τύπων δεδομένων. Από τις πράξεις στις δομές δεδομένων αυτές που κυρίως μας ενδιαφέρουν αναφέρονται στο παρακάτω πλαίσιο.

Κυριότερες Πράξεις στις Δομές Δεδομένων
(*α*) *διαπέραση* (*traversal*): προσπέλαση και επεξεργασία κάθε στοιχείου ή κόμβου.
(*β*) *αναζήτηση* (*search*): εύρεση στοιχείου ή κόμβου με κάποια δεδομένη τιμή.
(*γ*) *εισαγωγή* (*insertion*): πρόσθεση ενός νέου στοιχείου ή κόμβου.
(*δ*) *διαγραφή* (*deletion*): αφαίρεση ενός υπάρχοντος στοιχείου ή κόμβου.
(*ε*) *διάταξη* (*sorting*): τακτοποίηση των στοιχείων ή κόμβων σε κάποια σειρά.

Από τις πράξεις αυτές οι δύο πρώτες δεν επιφέρουν μεταβολή στις δομές, σε αντίθεση με τις υπόλοιπες. Τις τέσσερις πρώτες πράξεις, που θεωρούνται πιο βασικές, τις εξετάζουμε μαζί με τις αντίστοιχες δομές, με την πράξη της διάταξης όμως ασχολούμεθα σε ιδιαίτερο κεφάλαιο.

Η υλοποίηση των δομών δεδομένων είναι το κύριο αντικείμενο των επόμενων κεφαλαίων. Για την υλοποίηση αυτή δεν θα χρησιμοποιήσουμε κάποια από τις υπάρχουσες γλώσσες υψηλού επιπέδου

(Pascal, C κλπ), αλλά μια πιο απλή αλγοριθμική γλώσσα, που περιγράφεται στην επόμενη ενότητα.

Άσκηση Αυτοαξιολόγησης 1.1

Αντιστοιχήστε τις δομές δεδομένων στο παρακάτω πλαίσιο με όλες τις κατηγορίες στις οποίες ανήκουν.

δέντρα	ανώτερες
εγγραφές	γραμμικές
λίστες	δυναμικές
πίνακες	θεμελιώδεις
	μη γραμμικές
	στατικές

1.2 Αλγοριθμική γλώσσα

1.2.1 Εισαγωγικές έννοιες

Η υλοποίηση μιας πράξης σε μια δομή δεδομένων, δηλαδή η πραγματοποίησή της στον Η/Υ, περιγράφεται από μια υπολογιστική διαδικασία που ονομάζεται *αλγόριθμος*. ■

■ **Αλγόριθμος** είναι μια υπολογιστική διαδικασία αποτελούμενη από μια πεπερασμένη ακολουθία αυστηρά καθορισμένων υπολογιστικών βημάτων με σκοπό τη λύση ενός (γενικού) προβλήματος. Κάθε αλγόριθμος δέχεται κάποια δεδομένα σαν είσοδο και παράγει κάποια αποτελέσματα (λύση) σαν έξοδο.

Η περιγραφή ενός αλγορίθμου γίνεται με κάποια αλγοριθμική γλώσσα. Σαν τέτοια μπορεί να χρησιμοποιηθεί είτε η φυσική μας γλώσσα ή μια γλώσσα προγραμματισμού ή και μια διαγραμματική γλώσσα (όπως π.χ. το γνωστό διάγραμμα ροής). Το προαπαιτούμενο μιας αλγοριθμικής γλώσσας είναι η καθαρή σημασιολογία της σημειογραφίας και των συμβάσεων που χρησιμοποιεί, ώστε να μπορούν να περιγραφούν τα βήματα του αλγορίθμου με αυστηρό τρόπο, δηλαδή με τρόπο που επιδέχεται μόνο μια ερμηνεία.

Για την περιγραφή των αλγορίθμων χρησιμοποιούμε μια *γλώσσα ψευδοκώδικα*. Αυτό που μας ενδιαφέρει σ' ένα ψευδοκώδικα είναι η όσο το δυνατόν πιο κατανοητή και ταυτόχρονα συνοπτική περιγραφή των βασικών βημάτων ενός αλγορίθμου. ■

■ Μια **γλώσσα ψευδοκώδικα** είναι μια αυστηρά καθορισμένη γλώσσα που μας απαλλάσσει από τις λεπτομέρειες μιας γλώσσας προγραμματισμού.

Η γλώσσα ψευδοκώδικα που θα χρησιμοποιήσουμε διαθέτει στοιχεία που βασίζονται στις αρχές του δομημένου προγραμματισμού και είναι

παρόμοια με στοιχεία των γλωσσών προγραμματισμού Pascal (κυρίως) και C. Η μεταφορά ενός αλγορίθμου από ψευδοκώδικα σε πραγματικό κώδικα είναι σχετικά εύκολη. Η έκφραση ενός αλγορίθμου σε μια γλώσσα προγραμματισμού συνιστά ένα πρόγραμμα.

1.2.2 Περιγραφή της γλώσσας

Η περιγραφή της αλγοριθμικής μας γλώσσας βασίζεται στο παρακάτω παράδειγμα.

Παράδειγμα 1.4

Σχεδίαση αλγορίθμου από την περιγραφή του σε φυσική γλώσσα

Τίθεται το *πρόβλημα*: «Να γραφεί αλγόριθμος που να βρίσκει τη θέση και την τιμή του μικρότερου στοιχείου ενός πίνακα που περιέχει *n* αριθμούς.»

Προτείνεται η εξής διαδικασία *λύσης* σε φυσική γλώσσα: «Θεωρούμε το πρώτο στοιχείο του πίνακα σαν το μικρότερο και κρατούμε στη μνήμη την τιμή του και τη θέση του. Στη συνέχεια συγκρίνουμε την τιμή που κρατήσαμε με την τιμή του δεύτερου στοιχείου του πίνακα. Αν η τιμή του δεύτερου στοιχείου είναι μικρότερη, τότε κρατούμε στη μνήμη αυτή και τη θέση του, αλλιώς προχωρούμε και κάνουμε το ίδιο με το τρίτο στοιχείο κ.ο.κ. ως το τελευταίο. Η τελευταία τιμή και θέση που κρατήσαμε στη μνήμη είναι τα ζητούμενα.»

Ζητείται ο αντίστοιχος *αλγόριθμος*, γραμμένος στην αλγοριθμική μας γλώσσα.

Ο Αλγόριθμος αυτός παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 1.1.

Κάθε αλγόριθμος παρουσιάζεται μέσα σ' ένα πλαίσιο, όπως και ο Αλγόριθμος 1.1. Μέσα στο πλαίσιο υπάρχουν σύντομες περιγραφές της εισόδου και εξόδου του αλγορίθμου, όπου δίδονται πληροφορίες για τα δεδομένα εισόδου και εξόδου, καθώς και από ποιες μεταβλητές αντιπροσωπεύονται. Ακολουθεί η κυρίως περιγραφή του αλγορίθμου σαν μια υπολογιστική διαδικασία.

ΑΛΓΟΡΙΘΜΟΣ 1.1: ΕΥΡΕΣΗ ΜΙΚΡΟΤΕΡΟΥ ΣΤΟΙΧΕΙΟΥ ΠΙΝΑΚΑ

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A) και το πλήθος (N) των στοιχείων του, που είναι αριθμοί.

Έξοδος: Η θέση (X) και η τιμή (MIN) του μικρότερου στοιχείου του πίνακα.

```
MINSTOIXEIO(A, N)
1   K ← 1, X ← 1, MIN ← A[1]          {Αρχικοποίηση μεταβλητών.}
2   while K ≤ N                       {Έλεγχος τέλους επανάληψης}
3       if MIN > A[K]                 {Σύγκριση με τη μικρότερη τιμή}
4           then X ← K, MIN ← A[K]    {Ενημέρωση των X και MIN}
        endif
5       K ← K + 1                     {Ενημέρωση μετρητή}
    endwhile
6   print X, MIN                      {Επιστροφή αποτελεσμάτων}
```

Η υπολογιστική διαδικασία έχει μια *κεφαλίδα* που περιλαμβάνει το *όνομα* (MINSTOIXEIO) του αλγορίθμου ακολουθούμενο από τις *παραμέτρους εισόδου* (A, N), που λειτουργούν σαν μεταβλητές, σε παρένθεση. Το *σώμα* της διαδικασίας περιγράφεται σαν μια ακολουθία (αριθμημένων) υπολογιστικών βημάτων (1-6) κωδικοποιημένων στην αλγοριθμική μας γλώσσα, που περιγράφουμε αμέσως παρακάτω. Λέξεις ή σύμβολα που παραμένουν αμετάβλητα εμφανίζονται με έντονη γραφή.

Βήματα και Εκτέλεση

- Κάθε βήμα αποτελεί ένα *στοιχειώδες εκτελέσιμο τμήμα* αλγορίθμου και περιλαμβάνει, είτε μια έκφραση καταχώρησης (βήμα 5) είτε μια συνθήκη μιας διάταξης ελέγχου (βήματα 2, 3) είτε μια έκφραση εξόδου (βήμα 6).
- Σε ένα βήμα επιτρέπεται να υπάρχουν περισσότερες της μιας εκφράσεις καταχώρησης χωριζόμενες μεταξύ τους με κόμμα, οπότε θεωρούμε ότι εκτελούνται διαδοχικά από αριστερά προς τα δεξιά (π.χ. βήματα 1, 4). Τα βήματα αυτά θεωρούνται ως *σύνθετα βήματα*, και χρησιμοποιούνται απλώς και μόνο σαν μέσα συντόμευσης της περιγραφής ενός αλγορίθμου.
- Τα βήματα εκτελούνται διαδοχικά, σύμφωνα με τη σειρά αναγρα-

φής. Ο έλεγχος ροής της εκτέλεσης μπορεί να μεταφέρεται από ένα βήμα σε κάποιο άλλο που δεν είναι το επόμενο μόνο σαν αποτέλεσμα της ενέργειας κάποιας διάταξης ελέγχου ροής. (Οι διατάξεις ελέγχου ροής περιγράφονται στη συνέχεια της υποενότητας αυτής κάτω από τον ομώνυμο τίτλο).

- Σε κάθε βήμα μπορεί να υπάρχουν *σχόλια* μέσα σε άγκιστρα (π.χ. {Αρχικοποίηση μεταβλητών} στο βήμα 1 του Αλγορίθμου 1.1).

Είσοδος και Έξοδος

- Τα δεδομένα εισόδου εισάγονται με τις παραμέτρους εισόδου της κεφαλίδας. Δεν απαιτείται είσοδος δεδομένων στο σώμα του αλγορίθμου.
- Η έξοδος των αποτελεσμάτων γίνεται με μια *έκφραση εξόδου*, που αποτελείται από τη λέξη **print** ακολουθούμενη από καμία, μια ή περισσότερες μεταβλητές ή σταθερές, που αντιπροσωπεύουν τα (τελικά) αποτελέσματα (βήμα 6).

Μεταβλητές

- Για ονόματα μεταβλητών χρησιμοποιούνται συνδυασμοί κεφαλαίων γραμμάτων (MIN, K, X).
- Κάθε μεταβλητή έχει κάποιο ατομικό τύπο δεδομένων που δεν αναφέρεται ρητά στο σώμα του αλγορίθμου, αλλά υπονοείται από τα στοιχεία του προβλήματος.
- Η προσπέλαση σ' ένα στοιχείο μιας μεταβλητής τύπου πίνακα γίνεται με τον αντίστοιχο δείκτη, που μπορεί να είναι ακέραιος αριθμός ή μεταβλητή, σε τετραγωνικές παρενθέσεις, π.χ. A[1] (βήμα 1), A[K] (βήματα 3, 4).
- Όλες οι μεταβλητές είναι τοπικές, δηλαδή έχουν νόημα μόνο μέσα στην υπολογιστική διαδικασία του αλγορίθμου. Οι παράμετροι εισόδου μπορεί να παίζουν το ρόλο καθολικών μεταβλητών, δηλαδή να έχουν νόημα και εκτός του αλγορίθμου.

Σταθερές

- Σταθερές είναι οι αριθμοί (οποιουδήποτε τύπου), οι λογικές σταθερές (TRUE, FALSE) και χαρακτήρες ή λέξεις σε απλά εισαγωγικά (π.χ. 'ΛΑΘΟΣ').

Τελεστές και Παραστάσεις

- Για πράξεις μεταξύ υπολογιστικών οντοτήτων (μεταβλητών, στα-

θερών) χρησιμοποιούνται οι γνωστοί από τα μαθηματικά τελεστές: *αριθμητικοί* (+, -, *, /, ύψωση σε δύναμη, κλπ), *σύγκρισης* (<, ≤, =, ≠, ≥ , >) και *λογικοί* **(and**, **or**, **not**).

- Με τη χρήση των τελεστών αυτών δημιουργούνται αντίστοιχα *αριθμητικές παραστάσεις* (βήμα 5 δεξί μέλος), *παραστάσεις σύγκρισης* (βήματα 2, 3) και *λογικές παραστάσεις* (π.χ. (A > 1) **and** (B ≠ 0)) κατά τους γνωστούς από τα βασικά μαθηματικά τρόπους. Το αποτέλεσμα μιας αριθμητικής παράστασης είναι ένας αριθμός, ενώ των άλλων δύο τύπων παραστάσεων μια λογική σταθερά.

Έκφραση Καταχώρησης

- Χρησιμοποιούμε ένα βέλος προς τα αριστερά (←) για να υποδηλώσουμε την καταχώρηση (ή απόδοση) τιμής σε μια μεταβλητή:

  <μεταβλητή> ← <τιμή>

  όπου η <τιμή> μπορεί να είναι είτε μια σταθερά είτε μια μεταβλητή είτε μια οποιαδήποτε παράσταση (π.χ. στο βήμα 5 είναι μια αριθμητική παράσταση) ή μια συνάρτηση. (Η έννοια της συνάρτησης εξηγείται στο τελευταίο τμήμα αυτής της υποενότητας κάτω από τον τίτλο «Μερικοί Αλγόριθμοι»).

Διατάξεις Ελέγχου Ροής

- Οι *διατάξεις ελέγχου ροής* είναι υπολογιστικά σχήματα που επιτρέπουν την διαφοροποίηση από την ακολουθιακή εκτέλεση ενός αλγορίθμου. Χρησιμοποιούμε δύο τέτοιες διατάξεις: *διάταξη ελέγχου με συνθήκη* (ή *επιλογής*) και *διάταξη ελέγχου επανάληψης*.
- Στο σώμα του αλγορίθμου, για ευκολότερη διάκριση των διατάξεων ελέγχου, χρησιμοποιούνται και *εσοχές* (βήμα 4, βήματα 3-5) και *ενδείξεις τέλους* (endif, endwhile). Γραμμές που περιέχουν ενδείξεις τέλους δεν αριθμούνται, διότι είναι μη εκτελέσιμα βήματα.
- *Διάταξη Ελέγχου με Συνθήκη*

  Η διάταξη ελέγχου με συνθήκη ή *διάταξη if* επιτρέπει την επιλογή εκτέλεσης κάποιου τμήματος αλγορίθμου με βάση την αλήθεια κάποιας ή κάποιων συνθηκών και έχει την ακόλουθη μορφή:

```
if <συνθήκη>
   then <τμήμα Α>
   [else <τμήμα Β>]
endif
```

Η <*συνθήκη*> μπορεί να είναι είτε μια παράσταση σύγκρισης (όπως π.χ. στο βήμα 3) είτε μια λογική παράσταση. Το καθένα από τα <τμήμα Α> και <τμήμα Β> μπορεί να περιλαμβάνει ένα ή περισσότερα αλγοριθμικά βήματα. Οι τετραγωνικές παρενθέσεις στην περίπτωση αυτή δηλώνουν το προαιρετικό της γραμμής **else** (π.χ. στη διάταξη if του Αλγορίθμου 1.1 δεν υπάρχει). Το **endif** είναι απλώς η ένδειξη τέλους της διάταξης ελέγχου if και δεν αποτελεί εκτελέσιμο βήμα.

Η λογική της διάταξης αυτής είναι η εξής: αν η συνθήκη είναι αληθής τότε εκτελείται το <τμήμα Α>, αλλιώς εκτελείται το <τμήμα Β>, αν υπάρχει. Κατόπιν εκτελείται το επόμενο της διάταξης βήμα. Στο παράδειγμα, αν η παράσταση «MIN > A[K]» (βήμα 3) είναι αληθής, τότε εκτελείται το βήμα 4 (καταχώρηση των K, A[K] στα X, MIN αντίστοιχα) και στη συνέχεια η εκτέλεση προχωρά στο επόμενο της διάταξης βήμα 5 (αύξηση του μετρητή K κατά ένα), αλλιώς εκτελείται κατ' ευθείαν το βήμα 5.

- Το <τμήμα Β> μπορεί να είναι μια άλλη διάταξη if, επίσης το αντίστοιχο τμήμα αυτής μπορεί να είναι μια άλλη διάταξη if κ.ο.κ., οπότε δημιουργείται μια *σύνθετη διάταξη if*, δηλαδή μια διάταξη που περιέχει περισσότερες της μιας συνθήκες και περισσότερα των δύο τμήματα, οπότε για να εκτελεστεί ένα τμήμα πρέπει να αληθεύει ένας ορισμένος συνδυασμός συνθηκών.
- *Διάταξη Ελέγχου Επανάληψης*

  Η διάταξη ελέγχου επανάληψης ή *διάταξη while* επιτρέπει την επανάληψη της εκτέλεσης ενός τμήματος αλγορίθμου εφόσον ισχύει μια συνθήκη και έχει τη μορφή:

```
while  <συνθήκη>
       <τμήμα Χ>
endwhile
```

Η λογική εδώ είναι η εξής: Εξετάζεται κάθε φορά η <συνθήκη> και εκτελείται το <τμήμα Χ>, που λέγεται *σώμα* της διάταξης, εφόσον η συνθήκη είναι αληθής. Η εκτέλεση σταματά όταν η συνθήκη γίνει ψευδής, και τότε ο έλεγχος μεταφέρεται στο επόμενο βήμα της διάταξης. Το <τμήμα Χ> μπορεί να είναι μια άλλη διάταξη ελέγχου, είτε με

συνθήκη ή επανάληψης. Το **endwhile** αποτελεί την ένδειξη τέλους και δεν είναι εκτελέσιμο. Π.χ. στον Αλγόριθμο 1.1, η <συνθήκη> είναι μια παράσταση σύγκρισης (βήμα 2) και το σώμα περιλαμβάνει μια διάταξη if (βήματα 3-4) και μια έκφραση καταχώρησης (βήμα 5). Παρατηρήστε ότι η τιμή του K, δηλ. της μεταβλητής που επηρεάζει το αποτέλεσμα της συνθήκης, μεταβάλλεται σε κάθε εκτέλεση (βήμα 5), ώστε κάποια φορά θα ξεπεράσει την τιμή του N και θα πάψει να εκτελείται το σώμα της διάταξης. Αν δεν προβλέψουμε να συμβεί αυτό τότε θα έχουμε εκτέλεση του σώματος της διάταξης άπειρες φορές.

Οι περισσότερες γλώσσες προγραμματισμού διαθέτουν και μια άλλη διάταξη ελέγχου επανάληψης, τη *διάταξη for*, που χρησιμοποιείται όταν είναι γνωστός (ή μπορεί να υπολογιστεί) ο αριθμός των επαναλήψεων. Η διάταξη αυτή ουσιαστικά είναι πλεονασμός, διότι η λειτουργία της μπορεί να περιγραφεί από τη διάταξη while. Όμως, δημιουργεί συνοπτικότερη και καλύτερα αναγνώσιμη περιγραφή των αλγορίθμων και οδηγεί σε αποδοτικότερη εκτέλεση. Γι' αυτό ορίζουμε μια αντίστοιχη διάταξη και στην αλγοριθμική μας γλώσσα, που έχει τη μορφή

for <μετρητής> ← <αρχική τιμή> **to** <τελική τιμή>
    <τμήμα Χ>
endfor

Το <μετρητής> είναι μια μεταβλητή. Τα <αρχική τιμή>, <τελική τιμή> είναι ακέραιοι αριθμοί ή ακέραιες μεταβλητές ή αριθμητικές παραστάσεις ακεραίων. Το **endfor** είναι η ένδειξη τέλους και δεν είναι εκτελέσιμο.

Η λογική της διάταξης είναι η εξής: Ο μετρητής παίρνει σαν τιμή την <αρχική τιμή> και την αυξάνει διαδοχικά κατά ένα (1). Πριν από κάθε αύξηση εκτελείται το <τμήμα Χ>, το *σώμα* της διάταξης. Αυτό επαναλαμβάνεται έως ότου η τιμή του μετρητή γίνει μεγαλύτερη της <τελική τιμή>, οπότε σταματά. Τότε ο έλεγχος μεταφέρεται στο επόμενο της διάταξης βήμα του αλγορίθμου.

Με χρήση της διάταξης for αντί της while ο Αλγόριθμος 1.1 του Παραδείγματος 1.4 γίνεται.

ΑΛΓΟΡΙΘΜΟΣ 1.2: ΕΥΡΕΣΗ ΜΙΚΡΟΤΕΡΟΥ ΣΤΟΙΧΕΙΟΥ ΠΙΝΑΚΑ

```
MINSTOIXEIO(A, N)
1    X ← 1, MIN ← A[1]                    {Αρχικοποίηση μεταβλητών}
2    for K ← 1 to N                       {Καθορισμός επαναλήψεων}
3        if MIN > A[K]                    {Σύγκριση με τη μικρότερη τιμή}
4          then X ← K, MIN ← A[K]         {Ενημέρωση των X και MIN}
         endif
     endfor
5    print X, MIN .                       {Επιστροφή αποτελεσμάτων}
```

Παρατηρήστε ότι ο μετρητής K είναι πλέον μέρος της διάταξης if και ότι τα βήματα του αλγορίθμου ελαττώθηκαν κατά ένα. Εφεξής, θα χρησιμοποιούμε τη διάταξη for σαν διάταξη επανάληψης όταν γνωρίζουμε τον αριθμό των επαναλήψεων και τη διάταξη while στις υπόλοιπες περιπτώσεις.

Μερικοί Αλγόριθμοι

- Ένας *μερικός αλγόριθμος* είναι ένας ανεξάρτητος αλγόριθμος που καλείται για εκτέλεση μέσα από ένα *κύριο αλγόριθμο* ή από άλλον μερικό αλγόριθμο. Η κλήση ενός μερικού αλγορίθμου γίνεται με την κεφαλίδα του. Διακρίνουμε δύο είδη (μερικών) αλγορίθμων, την *συνάρτηση* (*function*) και τη *διαδικασία* (*procedure*). Η πρώτη επιστρέφει πάντα μια τιμή που μπορεί να αποδοθεί σε μια μεταβλητή, ενώ η δεύτερη εκτελεί κάποιες υπολογιστικές πράξεις (π.χ. καταχωρήσεις, ενημερώσεις κλπ) και ενδεχομένως να επιστρέφει και κάποια ή κάποιες τιμές.

Άσκηση Αυτοαξιολόγησης 1.2

Τίθεται το πρόβλημα: «Να γραφτεί αλγόριθμος που να βρίσκει το μέσο όρο των στοιχείων ενός πίνακα που περιέχει *n* αριθμούς». Δίνεται η εξής διαδικασία λύσης σε φυσική γλώσσα: «Προσθέτουμε το πρώτο και το δεύτερο στοιχείο και κρατούμε το άθροισμα στη μνήμη. Κατόπιν προσθέτουμε το τρίτο στοιχείο στο προηγούμενο άθροισμα και κρατούμε το νέο άθροισμα στη μνήμη κ.ο.κ. μέχρι να προσθέσουμε και το τελευταίο στοιχείο του πίνακα. Διαιρούμε το τελικό άθροισμα με το πλήθος των στοιχείων του πίνακα. Το αποτέλεσμα της διαίρεσης

είναι ο μέσος όρος». Να εκφράσετε τη λύση αυτή στην αλγοριθμική μας γλώσσα. Γράψτε μόνο την κεφαλίδα και το σώμα του αλγορίθμου.

1.3 Πολυπλοκότητα αλγορίθμων

Η υλοποίηση μιας πράξης επεξεργασίας μπορεί να γίνει συνήθως με περισσότερους του ενός τρόπους, δηλαδή αλγορίθμους. Επομένως τίθεται θέμα επιλογής του «καλύτερου» αλγορίθμου και εν γένει θέμα μέτρησης της επίδοσης ενός αλγορίθμου. Όταν λέμε «καλύτερος» αλγόριθμος, συνήθως εννοούμε «πιο αποδοτικός». Η *αποδοτικότητα* είναι το βασικό μέτρο σύγκρισης της επίδοσης δύο αλγορίθμων.

1.3.1 Η έννοια της αποδοτικότητας

Η αποδοτικότητα ενός αλγορίθμου έχει δύο παραμέτρους, την *αποδοτικότητα χρόνου* και την *αποδοτικότητα χώρου*. Η αποδοτικότητα χρόνου σχετίζεται με τον χρόνο επεξεργασίας που απαιτεί η εκτέλεση ενός αλγορίθμου. Η αποδοτικότητα χώρου σχετίζεται με την ποσότητα μνήμης που απαιτείται κατά την εκτέλεση του αλγορίθμου. Από τις δύο παραμέτρους, αυτή που είναι δυσκολότερο να εκτιμηθεί και συνήθως σημαντικότερη είναι η αποδοτικότητα χρόνου. Η αποδοτικότητα χώρου είναι απλούστερη στην εκτίμηση και λιγότερο σημαντική συνήθως. Γι' αυτό, εφεξής θα επικεντρώσουμε την προσοχή μας στην αποδοτικότητα χρόνου.

Το θέμα που τίθεται τώρα είναι πως θα μετράμε την αποδοτικότητα χρόνου, ποιο μέτρο θα χρησιμοποιούμε για την μέτρησή της. Ένας τρόπος θα ήταν να υλοποιήσουμε τον αλγόριθμο σε μια γλώσσα προγραμματισμού, να τον εκτελέσουμε και να μετρήσουμε την απόδοσή του μετρώντας τον χρόνο εκτέλεσής του. Όμως μια τέτοια μέτρηση επηρεάζουν α) η γλώσσα στην οποία θα υλοποιηθεί, β) οι ικανότητες του προγραμματιστή και γ) οι δυνατότητες του συγκεκριμένου Η/Υ στον οποίο θα τρέξει. Εμείς όμως ενδιαφερόμαστε για κάποιο τρόπο μέτρησης ανεξάρτητο από αυτούς τους παράγοντες.

Έτσι, για τον υπολογισμό της αποδοτικότητας ενός αλγορίθμου, στηριζόμαστε σ' αυτές που ονομάζουμε *βασικές υπολογιστικές πράξεις* (*basic operations*) του αλγορίθμου. Πιο συγκεκριμένα, η αποδοτικότητα εκτιμάται με τον αριθμό εκτελέσεων των βασικών υπολογιστι-

κών πράξεων, χωρίς να μας ενδιαφέρει ο πραγματικός χρόνος εκτέλεσής τους. Βασικές υπολογιστικές πράξεις είναι αυτές που χαρακτηρίζουν τη χρονική εκτέλεση ενός αλγορίθμου. Πολλές φορές επιλέγουμε μια σαν βασική υπολογιστική πράξη. Π.χ. στους αλγορίθμους αναζήτησης και διάταξης θεωρούμε συνήθως σαν βασική υπολογιστική πράξη την *σύγκριση* δύο στοιχείων, και επομένως η αποδοτικότητά τους εκτιμάται από τον αριθμό των συγκρίσεων που ενεργούνται κατά την εκτέλεση. Αντίθετα, σ' ένα αλγόριθμο διαγραφής θεωρούμε συνήθως σαν βασική πράξη την *μετακίνηση* ενός στοιχείου, οπότε η αποδοτικότητα εκτιμάται από τον αριθμό των μετακινήσεων που πραγματοποιούνται.

Είναι προφανές επίσης ότι η αποδοτικότητα εξαρτάται και από το μέγεθος των δεδομένων εισόδου. Π.χ. αν έχω να διαγράψω ένα στοιχείο από μια λίστα, ο αριθμός των μετακινήσεων που θα απαιτηθούν για να καλυφθεί το κενό εξαρτάται εν γένει από το πλήθος των στοιχείων της λίστας.

1.3.2 Συνάρτηση πολυπλοκότητας

Η εκτίμηση της αποδοτικότητας ενός αλγορίθμου αναφέρεται ως *πολυπλοκότητα* του αλγορίθμου και εκφράζεται με μια *συνάρτηση πολυπλοκότητας* $f(n)$, όπου n το μέγεθος/πλήθος των δεδομένων εισόδου. Η $f(n)$ εκφράζει την απαίτηση του αλγορίθμου σε χρόνο εκτέλεσης. Συνήθως μας ενδιαφέρει η εύρεση της τιμής της $f(n)$ στις εξής περιπτώσεις:

(α) *καλύτερη περίπτωση*: η ελάχιστη τιμή της $f(n)$

(β) *χειρότερη περίπτωση*: η μέγιστη τιμή της $f(n)$

(γ) *μέση περίπτωση*: η αναμενόμενη τιμή της $f(n)$.

Η εύρεση της τιμής της $f(n)$ στη μέση περίπτωση στηρίζεται κάθε φορά σε κάποια υπόθεση που κάνουμε για την κατανομή πιθανοτήτων των δεδομένων εισόδου, η οποία μπορεί και να μην είναι πρακτικά εφαρμόσιμη. Λόγω αυτού και του γεγονότος ότι η πολυπλοκότητα της μέσης περίπτωσης είναι πολύ πιο δύσκολο να βρεθεί, στα επόμενα κεφάλαια θα αναφερόμαστε συνήθως στις άλλες δύο περιπτώσεις, κυρίως δε σ' αυτή της χειρότερης περίπτωσης. Εξ' άλλου, η πολυπλοκότητα της μέσης περίπτωσης για πολλούς αλγορίθμους είναι κλάσμα της πολυπλοκότητας χειρότερης περίπτωσης.

Παράδειγμα 1.5 *Εύρεση συνάρτησης πολυπλοκότητας από τον αλγόριθμο*

Ας θεωρήσουμε τον Αλγόριθμο 1.1 και ας υπολογίσουμε τη συνάρτηση πολυπλοκότητας $f(n)$ θεωρώντας σαν κύρια υπολογιστική πράξη την *ενημέρωση*, δηλαδή την *καταχώρηση* (νέων) τιμών, των μεταβλητών X και MIN. Εδώ δεν θεωρούμε σαν βασική υπολογιστική πράξη και τη σύγκριση, διότι ο αριθμός των συγκρίσεων στον αλγόριθμο αυτό είναι σταθερός, και ίσος με n-1, σε όλες τις περιπτώσεις και επομένως δεν διαφοροποιεί την εκτίμηση της πολυπλοκότητας.

(α) Η καλύτερη περίπτωση συμβαίνει όταν το μικρότερο στοιχείο είναι πρώτο. Τότε γίνονται μόνο οι αρχικοποιήσεις των X και MIN (βήμα 1), και επομένως είναι $f(n) = 2$.

(β) Η χειρότερη περίπτωση συμβαίνει όταν τα στοιχεία του πίνακα είναι διατεταγμένα κατά φθίνουσα σειρά, οπότε κάθε σύγκριση στο βήμα 3, πλην της πρώτης, έχει σαν αποτέλεσμα την ενημέρωση των X και MIN. Δεδομένου ότι γίνονται συνολικά n συγκρίσεις, θα έχουμε n - 1 ενημερώσεις για καθένα από τα X και MIN, εκτός των αρχικοποιήσεων, δηλ. $f(n) = 2\ (n - 1) + 2 \Rightarrow f(n) = 2\ n$.

(γ) Για την ανάλυση της μέσης περίπτωσης θα θεωρήσουμε για απλότητα ότι ο πίνακας έχει μόνο τρία στοιχεία, και έστω ότι α_1, α_2 και α_3 είναι το μικρότερο, το αμέσως μεγαλύτερο και το μεγαλύτερο στοιχείο αντίστοιχα. Υπάρχουν τότε έξι δυνατοί τρόποι διάταξης με τους οποίους τα τρία στοιχεία μπορεί να υπάρχουν στον πίνακα, που αντιστοιχούν στους 3!=6 δυνατούς συνδυασμούς των τριών στοιχείων. Στη συνέχεια παραθέτουμε τους έξι συνδυασμούς και τις αντίστοιχες ενημερώσεις:

Συνδυασμοί:	$\alpha_1\ \alpha_2\ \alpha_3$	$\alpha_1\ \alpha_3\ \alpha_2$	$\alpha_2\ \alpha_1\ \alpha_3$	$\alpha_2\ \alpha_3\ \alpha_1$	$\alpha_3\ \alpha_1\ \alpha_2$	$\alpha_3\ \alpha_2\alpha_1$
Ενημερώσεις:	0	0	2	2	2	4

Υποθέτουμε ότι οι συνδυασμοί είναι ισοπίθανοι (αυτή είναι η υπόθεση κατανομής πιθανοτήτων δεδομένων εισόδου), οπότε η μέση τιμή είναι:

$$f(3) = \frac{0+0+2+2+2+4}{6} = \frac{10}{6}$$

Ο υπολογισμός της μέσης τιμής στη γενική περίπτωση των n στοιχείων είναι έξω από τους σκοπούς αυτού του κεφαλαίου. Ο σκοπός αυτού του παραδείγματος είναι να δώσει μια γεύση του τρόπου υπολογισμού

της μέσης περίπτωσης και να δείξει και το μέγεθος της δυσκολίας του υπολογισμού.

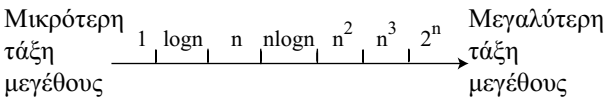
Άσκηση Αυτοαξιολόγησης 1.3

Για τον αλγόριθμο της Άσκησης Αυτοαξιολόγησης 1.2 υπολογίστε τη συνάρτηση πολυπλοκότητας θεωρώντας σαν βασικές πράξεις την πρόσθεση και την καταχώρηση.

Στην εύρεση της πολυπλοκότητας ενός αλγορίθμου δεν μας ενδιαφέρει τόσο η ακριβής τιμή της όσο η τάξη μεγέθους της σε σχέση με το πλήθος των δεδομένων εισόδου. Η *τάξη μεγέθους* της συνάρτησης πολυπλοκότητας δηλώνεται με το σύμβολο O, που διαβάζεται *κεφαλαίο ή μεγάλο όμικρον* (*big-O*), ως εξής:

$$f(n) = O(g(n)),$$

όπου η $g(n)$ ονομάζεται *συνάρτηση τάξης μεγέθους*, όπως π.χ. στις εκφράσεις $f(n) = O(1)$, $f(n) = O(n\log n)$, $f(n) = O(n^2)$. Οι σπουδαιότερες συναρτήσεις τάξης μεγέθους παρουσιάζονται στο Σχήμα 1.2, όπου $\log n$ ($\log^2 n$) παριστάνει τον δυαδικό λογάριθμο του n, και το βέλος υποδηλώνει τη σειρά τους κατά αύξουσα τάξη μεγέθους (για τις αντίστροφες των συναρτήσεων προφανώς ισχύει η αντίστροφη σειρά). Για να πάρουμε μια ιδέα της διαφοράς τους, ας θεωρήσουμε ότι $n = 1024$. Τότε $\log n = 10$, $n \log n = 1024 \times 10 = 10.240$, $n^2 \approx 1.000.000$ και $n^3 \approx 1.000.000.000$.



Σχήμα 1.2

Βασικές συναρτήσεις τάξης`μεγέθους.

Η τάξη μεγέθους της $f(n)$ βασικά εξαρτάται από τον επικρατούντα όρο της αναλυτικής της έκφρασης. Οι δευτερεύοντες όροι δεν μπορούν να αλλάξουν το μέγεθος της πολυπλοκότητας. Επικρατών όρος είναι αυτός που έχει μεγαλύτερη τάξη μεγέθους, δηλαδή δίνει μεγαλύτερες τιμές καθώς το n αυξάνει.

Παράδειγμα 1.6

(α) Ας υποθέσουμε ότι $f(n) = \frac{n(n+1)}{2}$

και ζητούμε να προσδιορίσουμε την τάξη μεγέθους της. Αναλύουμε την $f(n)$ ως εξής: $f(n) = (½)\, n^2 + (½)\, n$. Ο επικρατών όρος εδώ είναι ο $(½)\, n^2 = n^2/2$ και επομένως είναι: $f(n) = O(n^2)$.

(β) Έστω $f(n) = \frac{8 \log n + 4n}{3n^2 \log n}$, οπότε

$$f(n) = \frac{8 \log n}{3n^2 \log n} + \frac{4n}{3n^2 \log n} \text{ και τελικά}$$

$$f(n) = \frac{8 \log n}{3n^2 \log n} + \frac{4n}{3n^2 \log n}.$$

Εδώ ο επικρατών όρος είναι ο δεύτερος, διότι $n \log n < n^2$. Επομένως είναι: $f(n) = O(\frac{1}{n \log n})$.

Παρατηρήστε στο παράδειγμα ότι οι αριθμητικοί συντελεστές ουσιαστικά αγνοούνται.

Άσκηση Αυτοαξιολόγησης 1.4

Να βρεθεί η τάξη μεγέθους των συναρτήσεων:

α) $f(n) = 5\, n \log n + 0.5\, n^2 + 10$ β) $f(n) = \frac{3n + 5 \log n + 1}{2n}$

Σύνοψη

Ένας ***τύπος δεδομένων*** καθορίζει το είδος των τιμών που μπορεί να πάρει μια μεταβλητή, δηλαδή το ***πεδίο τιμών της***, και τις ***πράξεις*** που μπορούν να γίνουν στις τιμές αυτές. Σ' ένα ***ατομικό τύπο δεδομένων*** οι τιμές είναι απλές (π.χ. αριθμοί) και οι πράξεις στοιχειώδεις (π.χ. αριθμητικές). Μια ***δομή δεδομένων*** είναι ένας τύπος δεδομένων που προσδιορίζει μεταβλητές που αντιπροσωπεύουν σύνθετες τιμές. Οι σύνθετες τιμές περιλαμβάνουν επί μέρους τιμές που ονομάζονται ***στοιχεία*** ή ***κόμβοι*** και μπορεί να είναι είτε απλές τιμές είτε

άλλες σύνθετες τιμές. Μια δομή δεδομένων λοιπόν εκτός από το πεδίο τιμών και τις αντίστοιχες πράξεις προσδιορίζει και το ***οργανωτικό σχήμα*** των (σύνθετων) τιμών, δηλαδή τον τρόπο οργάνωσης ή συσχετισμού των στοιχείων της δομής. Αυτό το χαρακτηριστικό δεν υπάρχει σ' ένα ατομικό τύπο δεδομένων διότι στερείται επί μέρους τιμών και επομένως (εσωτερικής) οργάνωσης. Το πλήθος των στοιχείων (ή κόμβων) μιας δομής συνιστά το ***μέγεθος*** της δομής.

Με βάση την απλότητα της δομής και τη μεταβλητότητα του μεγέθους της κατά τη διάρκεια εκτέλεσης του προγράμματος, οι δομές δεδομένων διακρίνονται σε ***θεμελιώδεις-στατικές*** (πίνακες, εγγραφές) και ***ανώτερες-δυναμικές*** (λίστες, δένδρα). Επιπλέον, με βάση το είδος του οργανωτικού σχήματος διακρίνονται σε ***γραμμικές*** (πίνακες, λίστες) και ***μη γραμμικές*** (εγγραφές, δένδρα).

Ο **αλγόριθμος** είναι μια πεπερασμένη ακολουθία υπολογιστικών βημάτων που δίνει κάποιο επιδιωκόμενο αποτέλεσμα. Κάθε αλγόριθμος περιγράφεται με μια ***γλώσσα ψευδοκώδικα***, δηλαδή μια αυστηρά καθορισμένη γλώσσα που είναι όμως απλούστερη από μια γλώσσα προγραμματισμού υψηλού επιπέδου και μας αποδεσμεύει από μη αναγκαίες λεπτομέρειες.

Οι αλγόριθμοι χρησιμοποιούνται ως μέσα περιγραφής της υλοποίησης μιας δομής δεδομένων, η οποία κυρίως αναφέρεται στην υλοποίηση των πράξεων στη δομή. Είναι δυνατόν να υπάρχουν περισσότεροι του ενός αλγόριθμοι υλοποίησης μιας πράξης. Η σύγκρισή τους στηρίζεται κυρίως στην εκτίμηση της αποδοτικότητάς χρόνου που εκφράζεται με τη ***συνάρτηση πολυπλοκότητας***. Στον προσδιορισμό της πολυπλοκότητας ενός αλγορίθμου δεν μας ενδιαφέρει τόσο η ακριβής τιμή της όσο η τάξη μεγέθους της.

Οδηγός για περαιτέρω μελέτη

Μια περισσότερο αναλυτική παρουσίαση του θέματος της πολυπλοκότητας αλγορίθμων μπορείτε να βρείτε στην παρακάτω βιβλιογραφία:

1. M. Main and W. Savitch, *Data Structures and Other Objects*, Benjamin/Cummings, 1995.

   Στην ενότητα 1.2 γίνεται μια εισαγωγή στο θέμα της χρονικής

πολυπλοκότητας αλγορίθμων και του συμβολισμού *O*, ενώ στην 10.1 γίνεται μια αναλυτικότερη παρουσίαση του θέματος.

2. C. A. Shaffer, *A Practical Introduction to Data Structures and Algorithm Analysis*, Prentice Hall, 1997.

   Το κεφάλαιο 3 του βιβλίου αναφέρεται με αυστηρό, αναλυτικό και συστηματικό τρόπο στην ανάλυση αλγορίθμων. Γίνεται χρήση, πέραν του συμβολισμού *O*, και των συμβολισμών Ω και Θ. Παρουσιάζονται επίσης υπολογισμοί χρονικής πολυπλοκότητας συχνά χρησιμοποιούμενων τμημάτων αλγορίθμων. Τέλος, αναφέρεται και στην πολυπλοκότητα χώρου των αλγορίθμων.

3. T. A. Standish, *Data* Structures, *Algorithms and Software Principles*, Addison-Wesley, 1994.

   Στις ενότητες 6.1 ως και 6.4 το βιβλίο αυτό παρουσιάζει αναλυτικά το θέμα της χρονικής ανάλυσης αλγορίθμων με ιδιαίτερη έμφαση στην έννοια και χρήση του συμβολισμού *O* και την εύρεση της τάξης μεγέθους από την αναλυτική έκφραση της συνάρτησης πολυπλοκότητας.

4. M. A. Weiss, *Data Structures and Algorithm Analysis*, Benjamin/Cummings, 2nd Edition, 1995.

   Το κεφάλαιο 2 του βιβλίου αναπτύσσει αυστηρά, αναλυτικά και συστηματικά το θέμα της χρονικής ανάλυσης αλγορίθμων με τη χρήση όχι μόνο του συμβολισμού *O*, αλλά και των συμβολισμών Ω και Θ, που είναι δύο άλλοι συμβολισμοί που χρησιμοποιούνται στα πλαίσια της χρονικής ανάλυσης αλγορίθμων. Παρουσιάζονται κανόνες υπολογισμού της πολυπλοκότητας χρόνου καθώς και τρόποι μείωσής της.

ΚΕΦΑΛΑΙΟ 2

Πίνακες και εγγραφές

Σκοπός

Ο σκοπός του κεφαλαίου αυτού είναι να σας παρουσιάσει τις δύο θεμελιώδεις δομές δεδομένων, 'πίνακας' και 'εγγραφή'. Η παρουσίαση αφορά τους ορισμούς, τους τύπους, την αναπαράσταση και τις πράξεις στις δομές αυτές, κυρίως όσον αφορά στους πίνακες.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό, θα είστε σε θέση να:

- *εξηγείτε τα ιδιαίτερα χαρακτηριστικά της δομής του πίνακα,*
- *βρίσκετε τη συνάρτηση απεικόνισης ενός πίνακα,*
- *υπολογίζετε τη διεύθυνση ενός τυχαίου στοιχείου ενός πίνακα από τη συνάρτηση απεικόνισης,*
- *βρίσκετε τη συνάρτηση απεικόνισης και τον τρόπο υλοποίησης ενός ειδικού πίνακα όταν δίνεται ο τρόπος αναπαράστασής του,*
- *εξηγείτε την εφαρμογή των αλγορίθμων γραμμικής και δυαδικής αναζήτησης σε συγκεκριμένους πίνακες,*
- *σχεδιάζετε παραλλαγές των βασικών αλγορίθμων γραμμικής και δυαδικής αναζήτησης,*
- *σχεδιάζετε αλγορίθμους αναζήτησης που χρησιμοποιούν πίνακες εγγραφών.*

Έννοιες κλειδιά

- *Μονοδιάστατος πίνακας*
- *Σύνολο δεικτών*
- *Μεταβλητή με δείκτη*
- *Δομή Τυχαίας προσπέλασης*
- *Αναπαράσταση πίνακα*

- *Συνάρτηση απεικόνισης πίνακα*
- *Διεύθυνση βάσης*
- *Δισδιάστατος πίνακας*
- *Διάταξη κατά γραμμές*
- *Διάταξη κατά στήλες*
- *Πολυδιάστατος πίνακας*
- *Ειδικός πίνακας*
- *Συμμετρικός πίνακας*
- *Τριγωνικός πίνακας*
- *Τριδιαγώνιος πίνακας*
- *Γραμμική αναζήτηση*
- *Δυαδική αναζήτηση*
- *Εγγραφή*
- *Πεδία εγγραφής*
- *Πίνακες εγγραφών*

Ενότητες Κεφαλαίου 2

2.1 Πίνακες

2.2 Αναζήτηση σε πίνακες

2.3 Εγγραφές

Εισαγωγικές Παρατηρήσεις

Στην πρώτη ενότητα αυτού του κεφαλαίου εξετάζονται οι πίνακες, μονοδιάστατοι και δισδιάστατοι, όσον αφορά κυρίως την αναπαράστασή τους στην μνήμη του Η/Υ. Επίσης, γίνεται μια σύντομη αναφορά στους πολυδιάστατους και στους ειδικούς πίνακες. Οι πίνακες είναι από τις πιο βασικές δομές δεδομένων.

Στη δεύτερη ενότητα, αναφερόμαστε στο θέμα της αναζήτησης ενός στοιχείου σ' ένα (μονοδιάστατο) πίνακα. Εξετάζουμε δύο βασικές μεθόδους, τη γραμμική αναζήτηση (για μη διατεταγμένους πίνακες) και τη

δυαδική αναζήτηση (για διατεταγμένους πίνακες). Οι βασικοί αλγόριθμοι που παρουσιάζονται εδώ, μπορούν να χρησιμοποιηθούν και σε άλλες δομές, όπως οι λίστες, με τις απαιτούμενες συντακτικές κυρίως αλλαγές. Η λογική παραμένει η ίδια.

Τέλος, στην τρίτη ενότητα παρουσιάζουμε βασικά στοιχεία της δομής 'εγγραφή'. Οι εγγραφές, συχνά αποτελούν στοιχεία άλλων δομών. Στην περίπτωση του πίνακα, η παραγόμενη δομή ονομάζεται «πίνακας εγγραφών», στην οποία αναφερόμαστε εν συντομία.

2.1 Πίνακες

2.1.1 Μονοδιάστατος πίνακας

Ο μονοδιάστατος *πίνακας* (*array*)[1], όπως αναφέρθηκε και στην εισαγωγή του κεφαλαίου 1, είναι μια θεμελιώδης δομή δεδομένων που υπάρχει ενσωματωμένη σε όλες σχεδόν τις γλώσσες προγραμματισμού υψηλού επιπέδου. Πριν αναφερθούμε λοιπόν στις ανώτερες δομές, κρίνουμε σκόπιμο να αναφερθούμε στον πίνακα, δεδομένου ότι αποτελεί τη βάση υλοποίησης για πολλές από τις ανώτερες δομές.

Πίνακας είναι μια συλλογή ενός πεπερασμένου πλήθους στοιχείων του ίδιου τύπου, που ονομάζεται *τύπος βάσης* (*base type*). Λόγω της τελευταίας ιδιότητας, λέμε ότι ένας πίνακας αποτελεί μια *ομογενή δομή* (*homogeneous structure*). Κάθε πίνακας σχετίζεται με ένα *σύνολο δεικτών* (*index set*) *I*, που αποτελείται από τόσους δείκτες όσα και τα στοιχεία του πίνακα. Οι δείκτες είναι και αυτοί ίδιου τύπου δεδομένων, που ονομάζεται *τύπος δείκτη* (*index type*). Κάθε δείκτης αντιστοιχεί σ' ένα και μόνο ένα στοιχείο του πίνακα. ■

■ Ένας **μονοδιάστατος πίνακας** ορίζεται από ένα σύνολο Α με πεπερασμένο πλήθος Ν στοιχείων κοινού τύπου δεδομένων (τύπος βάσης) και από ένα σύνολο δεικτών Ι κοινού τύπου δεδομένων (τύπος δείκτη) τέτοιο, ώστε υπάρχει μια αμφιμονοσήμαντη αντιστοιχία μεταξύ των στοιχείων του Ι και των στοιχείων του Α.

Παρόλο που οι δείκτες δεν είναι απαραίτητα ακέραιοι αριθμοί, εφεξής, για λόγους απλότητας και δεδομένου ότι δε βλάπτεται η γενικότητα, θα θεωρούμε δείκτες μόνο ακεραίους και μάλιστα διαδοχικούς. Σ' αυτή την περίπτωση, οι δείκτες είναι συνήθως διατεταγμένοι.

Έστω, πίνακας (ή μεταβλητή τύπου πίνακα) A και το αντίστοιχο σύνολο δεικτών $I = \{1, 2, \ldots, n\}$. Θα αναφερόμαστε στα στοιχεία ενός τέτοιου πίνακα με τα σύμβολα $A_1, A_2, \ldots, A_n$. Το A_k, όπου $k \in I$, λει-

[1] Στην ξένη βιβλιογραφία χρησιμοποιείται και ο όρος 'table', ως γενικότερος όμως του 'array'.

τουργεί σαν μια μεταβλητή με τύπο τον τύπο βάσης του πίνακα και ονομάζεται *μεταβλητή με δείκτη* (*indexed variable*), ενώ το *k* ονομάζεται *δείκτης* (*index*) και είναι τύπος δείκτη. Θυμηθείτε ότι ο αντίστοιχος συμβολισμός στην αλγοριθμική μας γλώσσα είναι A[K].

Στη γενική περίπτωση, το σύνολο δεικτών ενός πίνακα είναι $I = \{n_l, \ldots, n_u\}$, όπου $n_l < n_u$. Τα n_l και n_u καλούνται αντίστοιχα *κάτω όριο* (*lower bound*) και *πάνω όριο* (*upper bound*) δείκτη. Το μέγεθος του πίνακα *N* μπορεί να υπολογιστεί, αν είναι γνωστά αυτά τα όρια, από τον παρακάτω τύπο:

$$N = n_u - n_l + 1. \tag{2.1}$$

Παρατηρήστε ότι αν $n_l = 1$, τότε $N = n_u = n$.

Το *πεδίο τιμών* ενός πίνακα μεγέθους *N* είναι ένα σύνολο που περιλαμβάνει πλειάδες μήκους *N*, που είναι όλοι οι δυνατοί συνδυασμοί τιμών των *N* στοιχείων του πίνακα. Οι *βασικές πράξεις* σ' ένα πίνακα είναι αυτές που αναφέραμε στο Παράδειγμα 1.3 στο κεφάλαιο 1, δηλαδή η *προσπέλαση* (*access*) ή *ανάκτηση* (*retrieval*) ενός στοιχείου και η *καταχώρηση* (*assignment*) ή *ενημέρωση* (*update*) ενός στοιχείου. Στην αλγοριθμική μας γλώσσα, αυτές οι πράξεις εκφράζονται ως εξής (όπου X είναι μια μεταβλητή):

X ← A[K] (προσπέλαση ή ανάκτηση του στοιχείου με δείκτη K)

A[K] ← X (καταχώρηση ή ενημέρωση του στοιχείου με δείκτη K).

Κάθε γλώσσα προγραμματισμού έχει το δικό της τρόπο για τη δήλωση (δημιουργία) ενός πίνακα. Πάντως, κάθε τέτοια δήλωση πρέπει να περιλαμβάνει τρία στοιχεία: (α) το όνομα του πίνακα, (β) τον τύπο βάσης του πίνακα και (γ) το σύνολο (ή τον τύπο) δεικτών του πίνακα. Στο παράδειγμα 3, στο προηγούμενο κεφάλαιο, είδαμε πώς δηλώνουμε ένα πίνακα στην Pascal με τα εξής στοιχεία: όνομα 'vec', τύπο στοιχείων 'real' και σύνολο δεικτών $I = \{1, 2\}$.

Το μέγεθος ενός πίνακα συνήθως παραμένει σταθερό κατά τη διάρκεια εκτέλεσης ενός προγράμματος, γι' αυτό και ο πίνακας αποτελεί *στατική δομή*. Θυμηθείτε ότι για το ίδιο ζήτημα αναφερθήκαμε στην υποενότητα «1.1.4 Κατηγορίες Δομών Δεδομένων». Στις περισσότερες γλώσσες υψηλού επιπέδου (π.χ., Pascal, C), το μέγεθος ενός πίνακα καθορίζεται στατικά, κατά τη μετάφραση του προγράμματος. Υπάρχουν όμως και γλώσσες που επιτρέπουν τον καθορισμό του μεγέ-

θους ενός πίνακα και δυναμικά, δηλαδή κατά την εκτέλεση του προγράμματος μέσω μιας μεταβλητής (π.χ. Lisp). Και στις δύο περιπτώσεις όμως, το μέγεθος ενός πίνακα παραμένει σταθερό μετά τον καθορισμό του.

Ένα από τα βασικά χαρακτηριστικά ενός πίνακα είναι ότι είναι μια *δομή τυχαίας προσπέλασης*, δηλαδή ο χρόνος εντοπισμού (προσπέλασης) ενός στοιχείου του είναι ανεξάρτητος από τη θέση του στοιχείου στον πίνακα. Αυτό είναι μια ιδιότητα μοναδική για τους πίνακες που δεν υπάρχει σε άλλες δομές.

2.1.2 Αναπαράσταση μονοδιάστατου πίνακα

Για να είναι δυνατό να επιτευχθεί τυχαία προσπέλαση στα στοιχεία ενός πίνακα πρέπει να υπάρχει ένας τρόπος υπολογισμού της διεύθυνσης της θέσης μνήμης κάθε στοιχείου με βάση μόνο το δείκτη του στοιχείου, ανεξάρτητα από τα υπόλοιπα στοιχεία. Ας θεωρήσουμε ένα πίνακα A με $I = \{n_l, \ldots, n_u\}$.

Θυμηθείτε, από την υποενότητα «1.1.5 Υλοποίηση Δομών Δεδομένων», ότι η μνήμη ενός Η/Υ θεωρείται σαν ένα σύνολο θέσεων μνήμης (bytes) με διαδοχικές διευθύνσεις. Τα στοιχεία ενός πίνακα αποθηκεύονται σε γειτονικές (διαδοχικές) θέσεις μνήμης. Κάθε στοιχείο ενός πίνακα καταλαμβάνει εν γένει L θέσεις μνήμης. Το L καλείται *μήκος στοιχείου* (*element length*) του πίνακα και εξαρτάται από τον τύπο των στοιχείων του πίνακα και τη γλώσσα υλοποίησης. Ας θεωρήσουμε τη διεύθυνση της πρώτης θέσης μνήμης του πρώτου στοιχείου του πίνακα, που καλείται *διεύθυνση βάσης* (*base address*) του πίνακα, και ας τη συμβολίσουμε με b_A. Τότε, το πρώτο στοιχείο του πίνακα θα αποθηκευτεί σε L θέσεις μνήμης με διευθύνσεις

$$b_A\ ,\ b_A + 1,\ b_A + 2,\ \ldots,\ b_A + L\text{-}1.$$

Επομένως, η διεύθυνση της πρώτης θέσης του δεύτερου στοιχείου, που θα είναι η αμέσως επόμενη από την τελευταία του πρώτου στοιχείου, θα είναι $b_A + L$, αυτής του τρίτου στοιχείου θα είναι $b_A + 2L$, του τέταρτου $b_A + 3L$ κ.ο.κ., όπου οι συντελεστές 1, 2, 3, … του L εκφράζουν τον αριθμό των στοιχείων του πίνακα που προηγούνται των στοιχείων με δείκτες $n_l + 1$, $n_l + 2$, $n_l + 3$, … αντίστοιχα. (δηλαδή $1 = ((n_l + 1) - n_l)$, $2 = ((n_l + 2) - n_l)$, $3 = ((n_l + 3) - n_l)$, …). Άρα, κατ' αντιστοιχία, η διεύθυνση της πρώτης θέσης μνήμης του στοιχείου με δείκτη i ($n_l \leq i \leq n_u$) θα είναι $b_A + (i - n_l)\ L$, αφού υπάρχουν $(i - n_l)$

στοιχεία πριν το στοιχείο με δείκτη i, δηλαδή έχουν καταληφθεί $(i - n_l)\ L$ θέσεις από αυτά ξεκινώντας από τη θέση b_A. Επομένως, η διεύθυνση $loc(A_i)$ του πρώτου byte του στοιχείου με δείκτη i του πίνακα A είναι:

$$loc\ (A_i) = b_A + (i - n_l)\ L \qquad (2.2)$$

που μπορεί να γραφεί και ως

$$loc\ (A_i) = c_0 + c_1 i \qquad (2.3)$$

όπου

$$c_0 = b_A - n_1 L,\ \ c_1 = L \qquad (2.4)$$

Από την σχέση (2.3) είναι φανερό ότι η διεύθυνση του στοιχείου με δείκτη i είναι μια γραμμική συνάρτηση του i. Η σχέση αυτή αποτελεί τη *συνάρτηση απεικόνισης πίνακα* (*array mapping function)* ή συντομογραφικά *ΣΑΠ* (*AMF*). Τα c_0, c_1 είναι οι *σταθερές ΣΑΠ* και υπολογίζονται εσωτερικά από τον μεταφραστή της γλώσσας υλοποίησης για κάθε πίνακα.

Παρατηρήστε ότι ο χρόνος υπολογισμού της $loc\ (A_i)$ είναι ο ίδιος για κάθε i, δεδομένου ότι μετά τον υπολογισμό των c_0 και c_1 απαιτούνται ένας πολλαπλασιασμός $(c_1 L)$ και μια πρόσθεση $(c_0 + c_1 L)$ για τον υπολογισμό της διεύθυνσης οποιουδήποτε στοιχείου του πίνακα. Επομένως, αν δοθεί ο δείκτης i μπορούμε να προσπελάσουμε το περιεχόμενο του A_i χωρίς να περάσουμε από κανένα άλλο στοιχείο του A.

Παράδειγμα 2.1 *Καθορισμός χαρακτηριστικών, εύρεση ΣΑΠ και διεύθυνσης στοιχείου μονοδιάστατου πίνακα.*

Ας υποθέσουμε ότι μια υπηρεσία του Υπουργείου Παιδείας για να καταχωρήσει τον αριθμό των μαθηματικών που διορίστηκαν στη μέση εκπαίδευση για κάθε έτος από το 1956 ως το 1998 χρησιμοποιεί ένα πίνακα *MATH*. Στην περίπτωση αυτή, είναι χρήσιμο να θεωρήσουμε σαν σύνολο δεικτών του πίνακα το $I = \{1956, 1957, \ldots, 1998\}$. Έτσι, το στοιχείο $MATH_{1965}$ παριστάνει το πλήθος των μαθηματικών που διορίστηκαν το έτος 1965. Το μέγεθος του πίνακα υπολογίζεται από τον τύπο 2.1 ως εξής:

$$N = n_u - n_l + 1 = 1998\text{-}1956 + 1 = 43 \text{ στοιχεία.}$$

Αν μας δίνεται ότι το μήκος στοιχείου του πίνακα είναι $L = 2$ θέσεις μνήμης (bytes) και ότι η διεύθυνση βάσης είναι $b_{MATH} = 500$, τότε αυτό σημαίνει ότι $loc(MATH_{1956}) = 500$, $loc(MATH_{1957}) = 502$, $loc(MATH_{1958}) = 504$ κ.ο.κ., δηλαδή η διεύθυνση του πρώτου byte μνήμης που καταλαμβάνουν τα στοιχεία $MATH_{1956}$, $MATH_{1957}$, $MATH_{1958}$ κ.ο.κ. είναι 500, 502, 504 κ.ο.κ. Η κατάσταση της περιοχής της μνήμης που χρησιμοποιεί ο πίνακας απεικονίζεται στο σχήμα 2.1α.

Για να βρούμε τη ΣΑΠ υπολογίζουμε τις σταθερές της από τους τύπους (2.4),

$$c_0 = 500 - 1956 \times 2 = -3412,\ c_1 = 2$$

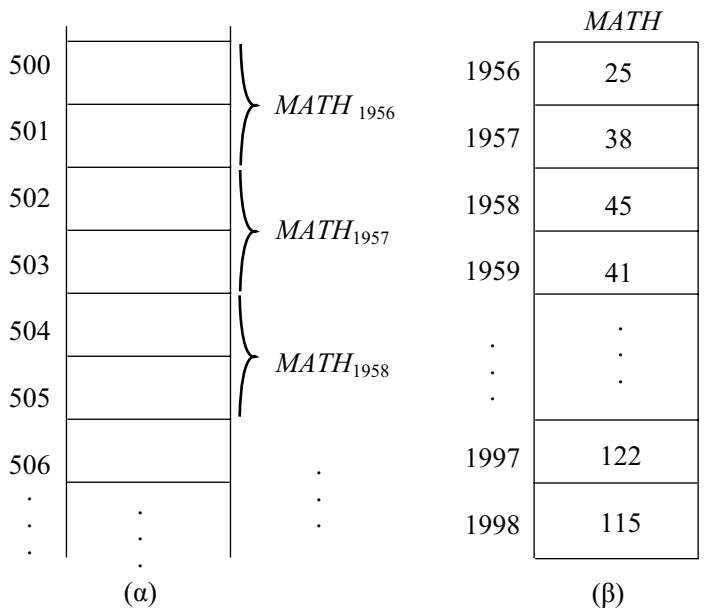
οπότε η ΣΑΠ (τύπος 2.3) είναι

$$loc(MATH_i) = -3412 + 2\,i\ .$$

Η διεύθυνση της πρώτης θέσης μνήμης του στοιχείου του πίνακα MATH που αναφέρεται στο έτος π.χ. 1985 υπολογίζεται από τη ΣΑΠ για $i = 1985$:

$$loc(MATH_{1985}) = -3412 + 2 \times 1985 = 558.$$

Στο σχήμα 2.1β παρουσιάζεται ένας τρόπος γραφικής απεικόνισης ενός μονοδιάστατου πίνακα, εδώ του *MATH* με κάποιες εικονικές τιμές στοιχείων.



Σχήμα 2.1

(α) Αναπαράσταση του πίνακα MATH στη μνήμη (β) Απεικόνιση του πίνακα MATH

Δραστηριότητα 2.1

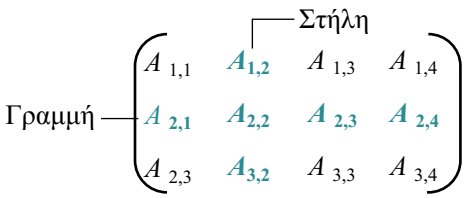
Με βάση το μέχρι τώρα περιεχόμενο αυτής της υποενότητας, εξηγήστε σε λιγότερο από μια σελίδα τους εξής χαρακτηρισμούς ενός πίνακα: α) ομογενής δομή β) δομή τυχαίας προσπέλασης, γ) στατική δομή και δ) γραμμική δομή. Για τα δύο τελευταία, ανατρέξτε για βοήθεια και στην υποενότητα «1.1.4 Κατηγορίες Δομών Δεδομένων». Τη δική μας συνοπτική απάντηση θα τη βρείτε στο τέλος του βιβλίου.

2.1.3 Δισδιάστατοι πίνακες

■ Ένας **δισδιάστατος πίνακας** ορίζεται από ένα σύνολο A με πεπερασμένο πλήθος N στοιχείων κοινού τύπου δεδομένων και από ένα σύνολο ζευγών δεικτών $I = I_1 \times I_2$ τέτοιο, ώστε υπάρχει μια αμφιμονοσήμαντη αντιστοιχία μεταξύ των στοιχείων του I και των στοιχείων του A.

■ Ένας *δισδιάστατος πίνακας* (*two-dimensional array*) είναι ένα σύνολο πεπερασμένου πλήθους στοιχείων του ίδιου τύπου, όπου κάθε στοιχείο προσδιορίζεται από ένα ζεύγος δεικτών. Δηλαδή, το σύνολο δεικτών I ενός δισδιάστατου πίνακα είναι ένα σύνολο διατεταγμένων ζευγών (θεωρούμε ακεραίων) δεικτών ή με άλλα λόγια το καρτεσιανό γινόμενο δύο συνόλων (θεωρούμε ακεραίων) δεικτών, $I = I_1 \times I_2$. Τα πλήθη των στοιχείων N_1 και N_2 των I_1 και I_2 αντίστοιχα καλούνται *διαστάσεις* (*dimensions*) του πίνακα, και λέμε ότι ο πίνακας είναι διαστάσεων $N_1 \times N_2$. Προφανώς, το μέγεθος του πίνακα είναι $N = N_1 \times N_2$.

Ας θεωρήσουμε ένα δισδιάστατο πίνακα A διαστάσεων $N_1 \times N_2$, με $I_1 = \{1, 2, \ldots, n_{u_1}\}$ και $I_2 = \{1, 2, \ldots, n_{u_2}\}$, οπότε $N_1 = n_{u_1}$ και $N_2 = n_{u_2}$ Τότε, θα αναφερόμαστε στα στοιχεία του πίνακα αυτού με τα σύμβολα $A_{1,1}, A_{1,2}, \ldots, A_{2,1}, A_{2,2}, \ldots$. Για καλύτερη εποπτεία και κατανόηση των δύο διαστάσεων, απεικονίζουμε ένα δισδιάστατο πίνακα διαστάσεων 3×4, όπως φαίνεται στο σχήμα 2.2.



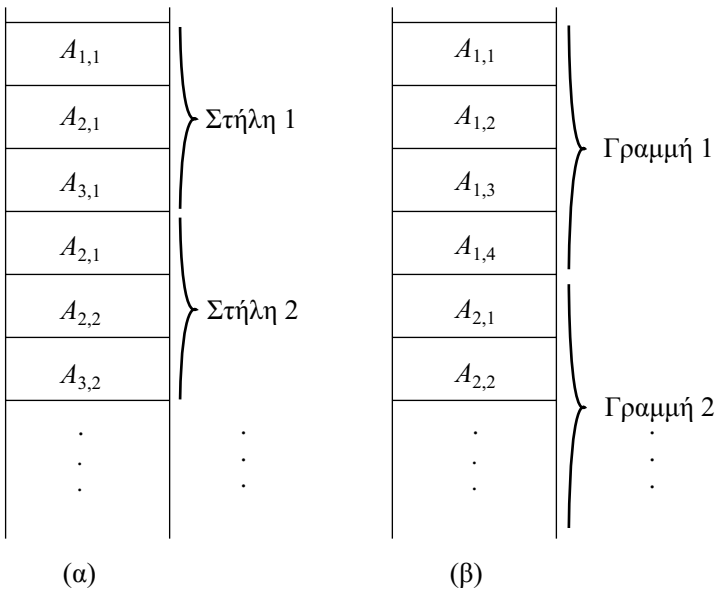
Σχήμα 2.2.
Δισδιάστατος πίνακας διαστάσεων 3 × 4.

Με βάση αυτή την απεικόνιση διακρίνουμε σ' ένα δισδιάστατο πίνακα *γραμμές* (*rows*), δηλαδή οριζόντιες συλλογές στοιχείων, και *στήλες* (*columns*), δηλαδή κατακόρυφες συλλογές στοιχείων. Ο παραπάνω πίνακας A έχει 3 γραμμές και 4 στήλες, που αντιστοιχούν στις διαστάσεις του πίνακα. Κάθε ζεύγος δεικτών προσδιορίζει ένα και μόνο

ένα στοιχείο του πίνακα. Γενικά το σύμβολο $A_{i,j}$, όπου $i \in I_1$ ($1 \leq i \leq N_1$) και $j \in I_2$ ($1 \leq j \leq N_2$), παριστάνει το στοιχείο που ορίζεται από την γραμμή i και την στήλη j ή, με άλλες λέξεις, το στοιχείο που βρίσκεται στη θέση j της γραμμής i ή, με άλλες λέξεις, το στοιχείο που βρίσκεται στη θέση i της στήλης j.

2.1.4 Αναπαράσταση δισδιάστατου πίνακα

Η αναπαράσταση ενός δισδιάστατου πίνακα διαστάσεων $N_1 \times N_2$ στη μνήμη του Η/Υ μπορεί να γίνει κατά δύο τρόπους: (α) με βάση τις στήλες του, οπότε ονομάζεται *διάταξη κατά στήλες* (*column-major order*), (β) με βάση τις γραμμές του, οπότε ονομάζεται *διάταξη κατά γραμμές* (*row-major order*). Οι δύο μέθοδοι απεικονίζονται στο σχήμα 2.3, όπου θεωρούμε ότι κάθε ορθογώνιο κελί αντιστοιχεί σε τόσες θέσεις μνήμης όσο και το μήκος στοιχείου του πίνακα.



Σχήμα 2.3.

Αναπαράσταση δισδιάστατου πίνακα στη μνήμη Η/Υ
(α)κατά στήλες, (β) κατά γραμμές

Ένας δισδιάστατος πίνακας, όπως και ένας μονοδιάστατος, αποθηκεύεται σε γειτονικές θέσεις μνήμης, ώστε ο Η/Υ δεν είναι υποχρεωμένος να κρατά τις διευθύνσεις όλων των στοιχείων του, αλλά μόνο του πρώτου στοιχείου του πίνακα, δηλαδή τη διεύθυνση της πρώτης θέσης μνήμης που καταλαμβάνει το στοιχείο $A_{1,1}$ που, όπως και στην

περίπτωση του μονοδιάστατου πίνακα, ονομάζεται διεύθυνση βάσης και τη συμβολίζουμε με b_A. Αν θεωρήσουμε ένα δισδιάστατο πίνακα A με $I_1 = \{ n_{i_1}, \ldots, n_{i_1} \}$ και $I_2 = \{ n_{l_2}, \ldots, n_{l_2} \}$, οι συναρτήσεις απεικόνισης για τους δύο τρόπους αναπαράστασής του αποδεικνύεται, με βάση παρόμοιο συλλογισμό με αυτόν στην περίπτωση του μονοδιάστατου πίνακα, ότι είναι οι εξής:

Διάταξη κατά στήλες

$$loc(A_{i,j}) = b_A + (N_1 (j - n_{l_2}) + (i - n_{l_1})) L \qquad (2.5)$$

η οποία μπορεί να γραφεί

$$loc(A_{i,j}) = c_0 + c_1 i + c_2 j \qquad (2.6)$$

όπου

$$c_1 = L, \; c_2 = N_1 L, \; c_0 = b_A - c_1 n_{l_1} - c_2 n_{l_2} \qquad (2.7)$$

οι σταθερές ΣΑΠ, και ως γνωστόν $N_1 = n_{u^I} - n_{l^I} + 1$.

Διάταξη κατά γραμμές

$$loc(A_{i,j}) = b_A + (N_2 (i - n_{l_1}) + (j - n_{l_2})) L \qquad (2.8)$$

η οποία μπορεί να γραφεί

$$loc(A_{i,j}) = c_0 + c_1 i + c_2 j \qquad (2.9)$$

όπου

$$c_1 = N_2 L, \; c_2 = L, \; c_0 = b_A - c_1 n_{l_1} - c_2 n_{l_2} \qquad (2.10)$$

και ως γνωστόν $N_2 = n_{l_2} - n_{l_2} + 1$.

Ας σημειωθεί ότι, η διεύθυνση ενός στοιχείου είναι γραμμική συνάρτηση των δεικτών i, j, σχέσεις (2.6) και (2.9). Επίσης, ότι ο χρόνος υπολογισμού είναι ανεξάρτητος από το συγκεκριμένο ζεύγος δεικτών.

Ένας άλλος τρόπος υπολογισμού της θέσης ενός στοιχείου πίνακα στη μνήμη του Η/Υ, εκτός της συνάρτησης απεικόνισης, είναι ο *πίνακας προσπέλασης* (*access table*), που είναι λιγότερο χρονοβόρος, αλλά απαιτεί περισσότερο χώρο μνήμης. Για περισσότερες λεπτομέρειες, αναφερθείτε στον οδηγό για περαιτέρω μελέτη, στο τέλος του κεφαλαίου.

Οι γλώσσες υψηλού επιπέδου δεν χρησιμοποιούν όλες τον ίδιο τρόπο αποθήκευσης των δυσδιάστατων πινάκων. Π.χ., η C και η πλειονότητα των γλωσσών υψηλού επιπέδου χρησιμοποιούν την αποθήκευση κατά γραμμές, ενώ η FORTRAN την κατά στήλες.

Παράδειγμα 2.2

Καθορισμός χαρακτηριστικών, εύρεση ΣΑΠ και διεύθυνσης στοιχείου δισδιάστατου πίνακα

Σ' ένα τμήμα 15 μαθητών δόθηκαν τέσσερα διαγωνίσματα, τα αποτελέσματα των οποίων (βαθμοί) παρουσιάζονται στον παρακάτω πίνακα.

Για την αποθήκευσή τους χρησιμοποιούμε ένα δισδιάστατο πίνακα *BATH* διαστάσεων 15×4 ($N_1 = 15$, $N_2 = 4$), με $I_1 = \{1, 2, \dots, 15\}$ και $I_2 = \{1, \dots, 4\}$. Κάθε ακέραιος στο I_1 αντιπροσωπεύει ένα μαθητή, ενώ κάθε ακέραιος στο I_2 ένα διαγώνισμα. Κάθε στοιχείο του πίνακα παριστάνει το βαθμό κάποιου μαθητή σε κάποιο διαγώνισμα. Π.χ., το $BATH_{5,3}$ παριστάνει το βαθμό του μαθητή 5 στο διαγώνισμα 3. Συνολικά, ο πίνακας έχει $15 \times 4 = 60$ στοιχεία.

		Διαγωνίσματα			
		1	2	3	4
Μαθητές	1	15	11	13	14
	2	12	14	15	12
	3	17	16	14	13
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.
	15	14	18	15	13

Βαθμοί

Αν μας δίνεται ότι η διεύθυνση βάσης του πίνακα είναι $b_{BATH} = 300$, το μήκος στοιχείου του πίνακα $L = 3$ και ότι η αποθήκευση γίνεται κατά γραμμές, τότε η ΣΑΠ προκύπτει από τον τύπο (2.9), αφού υπολογιστούν οι σταθερές από τους τύπους (2.10):

$$c_1 = N_2 L = 3 \times 3 = 9,$$

$$c_2 = L = 3,\ c_0 = b_A - c_1\ n_{l_1} - c_2\ n_{l_2} = 300 - 9 \times 1 - 3 \times 1 = 288$$

$$loc(BATH_{i,j}) = 288 + 9i + 3j$$

Αν θέλουμε τώρα να βρούμε τη διεύθυνση του πρώτου byte του στοιχείου $BATH_{12,3}$, υπολογίζουμε την τιμή της ΣΑΠ για $i = 12$ και $j = 3$:

$$loc(BATH_{12,3}) = 288 + 9 \times 12 + 3 \times 3 = 288 + 108 + 9 = 405.$$

2.1.5 Πολυδιάστατοι πίνακες

Γενικά, σ' έναν *πολυδιάστατο πίνακα* (*multidimensional array*) m διαστάσεων $N_1 \times N_2 \times \ldots \times N_m$, κάθε στοιχείο του συνόλου Α προσδιορίζεται από μια m-άδα δεικτών, δηλαδή είναι $I = I_1 \times I_2 \times \ldots \times I_m$. Η ΣΑΠ είναι κι εδώ γραμμική συνάρτηση των m δεικτών. Π.χ., στην περίπτωση της διάταξης κατά γραμμές αποδεικνύεται ότι είναι

$$loc(A_{i_1}, {}_{i_2}, \ldots, {}_{i_m}) = c_0 + c_1 i_1 + c_2 i_2 + \ldots + c_m i_m \qquad (2.11)$$

όπου

$$\begin{aligned} c_{\mathrm{m}} &= L \\ c_{l-1} &= N_l c_i \quad \text{για } i = 2, \ldots, m \\ c_0 &= b_A - c_1 n_{l_1} - c_2 n_{l_2} - \ldots - c_m n_{l_m}\ . \end{aligned} \qquad (2.12)$$

Οι διάφορες γλώσσες προγραμματισμού, στην καλύτερη περίπτωση, προσφέρουν πίνακες μέχρι και επτά διαστάσεις.

Οι υπολογισμοί των διευθύνσεων των στοιχείων ενός πίνακα, είτε μονοδιάστατου είτε δισδιάστατου είτε πολυδιάστατου, γίνονται από τον μεταφραστή της συγκεκριμένης γλώσσας με βάση τις αντίστοιχες ΣΑΠ, χωρίς να απαιτείται κάποια μέριμνα από τον προγραμματιστή. Οι παραπάνω συναρτήσεις είναι χρήσιμες σ' έναν προγραμματιστή, αν χρειαστεί να προγραμματίσει σε γλώσσα χαμηλού επιπέδου (π.χ. Assembly).

Για τον απαιτούμενο χειρισμό ενός πίνακα, ο μεταφραστής πρέπει να έχει ανά πάσα στιγμή διαθέσιμες όλες τις απαραίτητες πληροφορίες γι' αυτόν. Έτσι, για κάθε πίνακα δημιουργείται ένας *περιγραφέας πίνακα* (*array descriptor*), που κρατά τις ακόλουθες πληροφορίες: α) το όνομα του πίνακα, β) τον τύπο βάσης, γ) το μήκος στοιχείου, δ) τη διεύθυνση βάσης, ε) το κάτω και πάνω όριο δείκτη για κάθε διάσταση και στ) τις σταθερές της ΣΑΠ.

Άσκηση Αυτοαξιολόγησης 2.1

Μια αθλητική ομοσπονδία παρακολούθησε 5 αθλητές στίβου του ίδιου αθλήματος από το 1988 ως το 1997. Οι αθλητές συμμετείχαν κάθε χρόνο σε 3 συγκεκριμένους αγώνες. Οι επιδόσεις των αθλητών καταχωρήθηκαν σ' ένα πίνακα H.

α) Να ορίσετε τα χαρακτηριστικά του πίνακα (σύνολα δεικτών, διαστάσεις, πλήθος στοιχείων)

β) Να βρείτε τη συνάρτηση απεικόνισης του πίνακα, αν $b_H = 200$, $L = 2$ και η αποθήκευση στη γλώσσα υλοποίησης γίνεται κατά γραμμές.

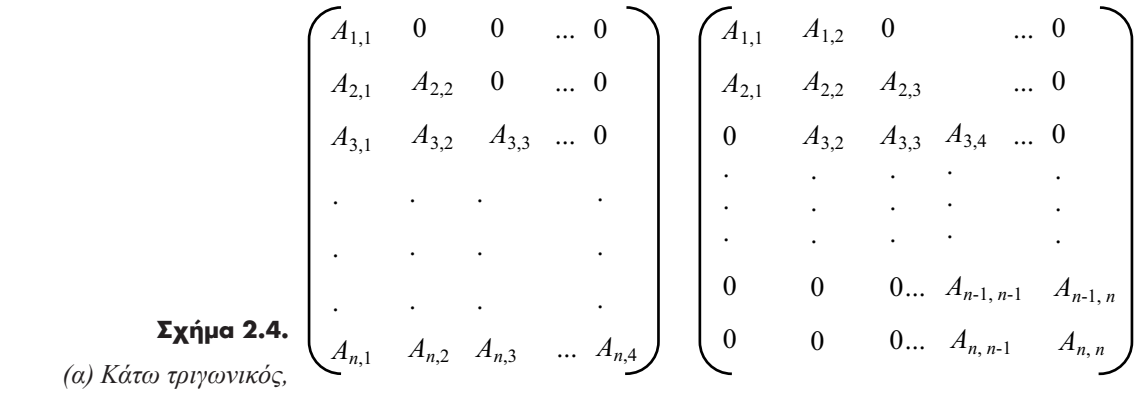
γ) Να υπολογίσετε τη διεύθυνση του στοιχείου που αντιπροσωπεύει την επίδοση του αθλητή 3, το έτος 1990 στον αγώνα 2.

2.1.6 Ειδικοί πίνακες

Υπάρχουν ορισμένοι πίνακες που χαρακτηρίζονται σαν ειδικοί, λόγω κάποιας χαρακτηριστικής ιδιότητας που έχουν. Αυτοί οι πίνακες είναι δυνατόν να αποθηκευθούν στη μνήμη του Η/Υ με ιδιαίτερο τρόπο, που έχει σαν αποτέλεσμα σημαντική μείωση του χώρου αποθήκευσης σε σχέση με τους κανονικούς πίνακες. Τέτοιες κατηγορίες πινάκων είναι οι συμμετρικοί, οι τριγωνικοί, οι τριδιαγώνιοι και οι αραιοί. Οι τρεις πρώτες κατηγορίες, έχουν το κοινό γνώρισμα, πέραν της ιδιαιτερότητας τους, ότι όλοι είναι *τετραγωνικοί πίνακες* (*rectangular arrays*), δηλαδή δισδιάστατοι πίνακες με $n_{l_1} = n_{l_2}$ και $n_{u_1} = n_{u_2}$ και, κατά συνέπεια, $N_1 = N_2 = N$. Οι αραιοί πίνακες μπορεί να είναι οποιωνδήποτε διαστάσεων.

Ένας πίνακας είναι *συμμετρικός* (*symmetric*) όταν $A_{i,j} = A_{j,i}$ για κάθε i, j.

Υπάρχουν δύο τύποι τριγωνικού πίνακα, ο *κάτω τριγωνικός πίνακας* (*lower triangular array*) και ο *πάνω τριγωνικός πίνακας* (*upper triangular array*). Στον κάτω τριγωνικό πίνακα, όλα τα στοιχεία πάνω από τη διαγώνιο είναι μηδενικά (δείτε το Σχήμα 2.4α), δηλαδή $A_{i,j} = 0$ για $i < j$, ενώ στον πάνω τριγωνικό, όλα τα στοιχεία κάτω από τη διαγώνιο είναι μηδενικά, δηλαδή $A_{i,j} = 0$ για $i > j$.



(α) (β)

Σχήμα 2.4.
(α) Κάτω τριγωνικός,
(β) Τριδιαγώνιος πίνακας

Στον *τριδιαγώνιο πίνακα* (*tridiagonal array*) όλα τα στοιχεία είναι μηδενικά, πλην αυτών της κύριας διαγωνίου ($i = j$), της υπερδιαγωνίου ($i = j$-1), δηλ. αυτής που βρίσκεται από πάνω της, και της υποδιαγωνίου ($i = j$+1), δηλ. αυτής που βρίσκεται από κάτω της (Σχήμα 2.4β).

Τέλος, *αραιός πίνακας* (*sparse array*) είναι ένας πίνακας στον οποίο ένα μεγάλο ποσοστό των στοιχείων του (π.χ. πάνω από 80%) είναι μηδενικά.

Στη συνέχεια, σαν παράδειγμα μελέτης ειδικού πίνακα, εξετάζουμε την περίπτωση του κάτω τριγωνικού πίνακα, που παρουσιάζεται στο Σχήμα 2.4α. Για εξοικονόμηση χώρου, δεν αποθηκεύουμε στη μνήμη τα μηδενικά στοιχεία του πίνακα. Παρατηρήστε ότι ένας τέτοιος πίνακας έχει 1 (μη μηδενικό) στοιχείο στην πρώτη γραμμή, 2 στη δεύτερη, 3 στη τρίτη, …, n στη n-οστή γραμμή. Επομένως, θα χρειαστεί να αποθηκεύσουμε [2]

$$\Sigma = 1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} \qquad (2.13)$$

στοιχεία, που είναι πολύ λιγότερα από τα $n \times n = n^2$, όσα θα ήταν στην περίπτωση που τα αποθηκεύαμε όλα. Π.χ., αν n =10, τότε, αντί για 100, θα αποθηκεύσουμε μόνο 55 στοιχεία.

---

[2] Πρόκειται περί γνωστής ταυτότητας από τα μαθηματικά, $\sum_{x=1}^{n} x = \frac{n(n+1)}{2}$

Τα μη μηδενικά στοιχεία τα αποθηκεύουμε σε διάταξη κατά γραμμές, δηλαδή με την εξής σειρά: $A_{1,1}\, A_{2,1}\, A_{2,2}\, A_{3,1}\, A_{3,2} \ldots A_{n,n-1}\, A_{n,n}$. Το ζητούμενο είναι να βρεθεί η ΣΑΠ ενός τέτοιου πίνακα, δηλαδή ο τύπος που δίνει τη διεύθυνση του πρώτου byte του (μη μηδενικού) στοιχείου $A_{i,j}$ σαν συνάρτηση των δεικτών i, j (όπου $i \geq j$).

Το στοιχείο $A_{i,j}$ βρίσκεται στη γραμμή i, επομένως, προηγούνται στη σειρά τα στοιχεία των i-1 προηγούμενων γραμμών, που είναι

$$\Sigma_i = 1 + 2 + 3 + \ldots + (i\text{-}1) = \frac{i(i-1)}{2}$$

(με εφαρμογή της ταυτότητας της υποσημείωσης). Το $A_{i,j}$ βρίσκεται στη θέση j στη γραμμή i, επομένως, προηγούνται άλλα $\Sigma_j = j$-1 στοιχεία. Άρα, συνολικά προηγούνται $(\Sigma_i + \Sigma_j)$ στοιχεία στη σειρά διάταξης στη μνήμη, που καταλαμβάνουν $(\Sigma_i + \Sigma_j)L$ θέσεις μνήμης. Επομένως, η διεύθυνση της πρώτης θέσης μνήμης του $A_{i,j}$ θα είναι:

$$loc(A_{i,j}) = b_{\mathrm{A}} + (\Sigma_i + \Sigma_j)L = b_{\mathrm{A}} + \frac{i(i\text{-}1)}{2}L + (j\text{-}1)L$$

που μπορεί να γραφεί

$$loc(A_{i,j}) = c_0 + c_1\, i(i\text{-}1) + c_2\, j \qquad (2.14)$$

όπου

$$c_0 = b_A\text{-}L,\ \ c_1 = L/2,\ \ c_2 = L \qquad (2.15)$$

Η ΣΑΠ του κάτω τριγωνικού πίνακα που βρήκαμε είναι αυτή που θα χρησιμοποιούσε ο μεταφραστής μιας γλώσσας εσωτερικά για τον υπολογισμό των διευθύνσεων των στοιχείων ενός τέτοιου πίνακα, με την προϋπόθεση ότι θα αποθήκευε τα στοιχεία με τον παραπάνω αναφερθέντα τρόπο. Όμως, δεν υπάρχει γλώσσα προγραμματισμού που προσφέρει ιδιαίτερο τύπο δεδομένων για τριγωνικούς πίνακες. Υπό τις συνθήκες αυτές, το ερώτημα είναι πώς μπορούμε να υλοποιήσουμε τα παραπάνω στις υπάρχουσες γλώσσες.

Η λύση είναι να υλοποιηθούν μέσω ενός μονοδιάστατου πίνακα DA με $I = \{1, \ldots, n_u\}$, όπου

$$n_u = N = \frac{n(n+1)}{2} \qquad (2.16)$$

δηλαδή όσο το πλήθος των μη μηδενικών στοιχείων. Δεδομένου ότι

υπάρχουν Σ_i στοιχεία στις προηγούμενες γραμμές και το $A_{i,j}$ βρίσκεται στην θέση j της γραμμής του, το $A_{i,j}$ αντιστοιχεί στο DA_k, όπου

$$k = \frac{i(i-1)}{2} + j. \tag{2.17}$$

Για να μπορεί να πραγματοποιηθεί αυτή η λύση χρειαζόμαστε επιπλέον α) μια συνάρτηση για την προσπέλαση/ανάκτηση των στοιχείων του DA με βάση τους δείκτες των στοιχείων του A και β) μια διαδικασία για την καταχώρηση/ενημέρωση των στοιχείων του A στα αντίστοιχα στοιχεία του DA. Παρακάτω, παρουσιάζονται οι αλγόριθμοι υλοποίησής τους.

Η συνάρτηση ANAKTHSH επιστρέφει μηδέν όταν οι δείκτες i, j αναφέρονται σε μηδενικό στοιχείο του πίνακα A ($i < j$), αλλιώς επιστρέφει το αντίστοιχο στοιχείο του DA, δηλαδή αυτό με δείκτη που υπολογίζεται από τον τύπο (2.17).

Η διαδικασία ENHMEROSH εισάγει την τιμή (X) του $A_{i,j}$ στο αντίστοιχο στοιχείο του DA, όταν οι δείκτες i, j αναφέρονται σε μη μηδενικό στοιχείο ($i \geq j$), αλλιώς αν η τιμή δεν είναι μηδενική, όπως θα όφειλε, επιστρέφει το μήνυμα 'ΜΗ ΕΓΚΥΡΗ ΤΙΜΗ'.

```
ANAKTHSH (I, J, DA)
1    if I < J
2       then X ← 0
3       else K ← ((I * (I-1)/2) + J)
4            X ← DA[K]
     endif
5    print X
```

```
ENHMEROSH (I, J, X, DA)
1    if I ≥ J
2       then K ← ((I ∗ (I-1)/2) + J)
3            DA[K] ← X
4       else if X ≠ 0
5               then print 'ΜΗ ΕΓΚΥΡΗ ΤΙΜΗ'
            endif
     endif
```

Με παρόμοιο τρόπο, μπορούμε να διαχειριστούμε και τις περιπτώσεις του πάνω τριγωνικού, του συμμετρικού και του τριδιαγώνιου πίνακα.

Άσκηση Αυτοαξιολόγησης 2.2

Δίνεται συμμετρικός πίνακας διαστάσεων $n \times n$. Να βρεθούν α) το κέρδος σε χώρο μνήμης που μπορούμε να έχουμε αποθηκεύοντας τα στοιχεία του με τρόπο παρόμοιο με αυτό του κάτω τριγωνικού πίνακα, β) η συνάρτηση απεικόνισης ενός τέτοιου πίνακα, γ) τα

χαρακτηριστικά του αντίστοιχου μονοδιάστατου πίνακα και δ) οι αλγόριθμοι για προσπέλαση/ανάκτηση και καταχώρηση/ενημέρωση του μονοδιάστατου πίνακα.

2.2 Αναζήτηση σε πίνακες

Η *αναζήτηση* (*search*) σ' έναν πίνακα αναφέρεται στο εξής πρόβλημα: Δοθείσης μιας τιμής να βρεθεί η θέση του στοιχείου στον πίνακα που έχει την ίδια τιμή με τη δοθείσα, αν υπάρχει, ή να τυπωθεί κάποιο μήνυμα αποτυχίας, αν δεν υπάρχει. Πολλές φορές, η αναζήτηση συνδυάζεται με τη διαγραφή ενός στοιχείου. Υπάρχουν διάφοροι μέθοδοι αναζήτησης σ' έναν πίνακα. Εδώ, θα αναφερθούμε σε δύο βασικές μεθόδους, τη *σειριακή* ή *γραμμική αναζήτηση* (*sequential* or *linear search*) και τη *δυαδική αναζήτηση* (*binary search*).

2.2.1 Γραμμική αναζήτηση

Η λύση που δίνει στο παραπάνω πρόβλημα η γραμμική αναζήτηση, είναι η προφανής. Συγκρίνουμε, δηλαδή, τη δοθείσα τιμή με την τιμή κάθε στοιχείου του πίνακα, ξεκινώντας από το πρώτο στοιχείο. Αν βρούμε κάποιο στοιχείο, με τιμή ίση με τη δοθείσα (επιτυχημένη αναζήτηση), τότε σταματάμε και επιστρέφουμε τη θέση του στοιχείου, αλλιώς (αποτυχημένη αναζήτηση) επιστρέφουμε ένα μήνυμα αποτυχίας.

ΑΛΓΟΡΙΘΜΟΣ 2.1: ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ

Είσοδος: Ένας πίνακας (A), το μέγεθος του πίνακα (N) και η προς αναζήτηση τιμή (X).

Έξοδος: Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα σε περίπτωση επιτυχίας ή μήνυμα σε περίπτωση αποτυχίας. Εσωτερικά, δεν συμβαίνει τίποτα.

```
GRAM-ANAZHTHSH (A, N, X)
1    I ← 1                                  {Αρχικοποίηση δείκτη θέσης}
2    while (I ≤ N) and (A[I] ≠ X)           {Έλεγχος τέλους επανάληψης}
3       I ← I + 1                           {Ενημέρωση δείκτη θέσης}
     endwhile
4    if A[I] = X                            {Αναγνώριση αποτελέσματος}
5       then print I                        {Επιστροφή θέσης στοιχείου}
6       else print 'ΑΠΟΤΥΧΙΑ'               {Επιστροφή μηνύματος αποτυχίας}
     endif
```

Ο αντίστοιχος αλγόριθμος, είναι ο Αλγόριθμος 2.1 (GRAM-ANAZHTHSH) και παρουσιάζεται στο προηγούμενο πλαίσιο, όπου θεωρούμε ένα πίνακα *Α* μεγέθους *Ν*. Η μεταβλητή (δείκτης) Ι χρησιμοποιείται για καταχώρηση της θέσης του στοιχείου. Η διάταξη while (βήματα 2-3) αυξάνει διαδοχικά την τιμή του δείκτη Ι κατά ένα (βήμα 3), ελέγχοντας κάθε φορά (στη συνθήκη της), αν το στοιχείο του πίνακα στην παρούσα θέση, δηλ. το Α[Ι], είναι ίσο με Χ. Οι επαναλήψεις (συγκρίσεις) σταματούν είτε όταν δεν υπάρχουν άλλα στοιχεία του πίνακα για εξέταση (Ι > Ν) είτε όταν βρεθεί το ζητούμενο στοιχείο (Α[Ι]=Χ). Σημειώστε ότι η σειρά των όρων του τελεστή **and** έχει σημασία. Αν ήταν αντίστροφη, τότε στην περίπτωση αποτυχίας, όταν θα συμβεί Ι > Ν, θα αναζητηθεί το ανύπαρκτο στοιχείο Α[Ι], λόγω της σύγκρισης (Α[Ι] ≠ Χ), πριν εκτιμηθεί ο δεύτερος όρος (Ι ≤ Ν) και σταματήσει η παραπέρα διαδικασία. Στη συνέχεια, μια διάταξη if (βήματα 4-6) επιστρέφει, ανάλογα με την περίπτωση, είτε την θέση του στοιχείου (βήμα 5) είτε το μήνυμα 'ΑΠΟΤΥΧΙΑ' (βήμα 6).

Η πολυπλοκότητα του αλγορίθμου υπολογίζεται με βάση τον αριθμό των συγκρίσεων που γίνονται στο βήμα 2. Στη χειρότερη περίπτωση, δηλαδή όταν το ζητούμενο στοιχείο είναι το τελευταίο ή δεν υπάρχει, είναι $f(n) = n$, διότι θα γίνουν τόσες συγκρίσεις όσα και τα στοιχεία του πίνακα. Στη μέση περίπτωση, αποδεικνύεται ότι,

$$f(n) = \frac{n+1}{2}.$$

Παράδειγμα 2.3

Γραμμική αναζήτηση σε διατεταγμένο πίνακα

Θεωρούμε ένα διατεταγμένο πίνακα, δηλαδή έναν πίνακα που τα στοιχεία του είναι τοποθετημένα κατά αύξουσα (ή αλφαβητική) σειρά. Το ζητούμενο είναι να σχεδιάσουμε ένα αλγόριθμο γραμμικής αναζήτησης για ένα τέτοιο πίνακα.

Το γεγονός ότι ο πίνακας είναι διατεταγμένος μάς δίνει τη δυνατότητα για αποδοτικότερη εφαρμογή της γραμμικής αναζήτησης. Η βασική διαφορά από την περίπτωση του μη διατεταγμένου πίνακα είναι ότι τώρα δε χρειάζεται να συνεχίσουμε την αναζήτηση πέραν του πρώτου στοιχείου που θα έχει τιμή μεγαλύτερη από τη δοθείσα, διότι δεν έχει νόημα.

ΑΛΓΟΡΙΘΜΟΣ Π2.3: ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ ΣΕ ΔΙΑΤΕΤΑΓΜΕΝΟ ΠΙΝΑΚΑ

Είσοδος: Ένας πίνακας (A), το μέγεθος του πίνακα (N) και η προς αναζήτηση τιμή (X).
Έξοδος: Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα σε περίπτωση επιτυχίας ή μήνυμα αποτυχίας. Εσωτερικά δεν συμβαίνει τίποτα.

```
DIAT-GRAM-ANAZHTHSH (A, N, X)
1    I ← 1                          {Αρχικοποίηση δείκτη θέσης}
2    while (I ≤ N) and (A[I] ≠ X)   {Έλεγχος τέλους επανάληψης}
3        I ← I + 1                  {Ενημέρωση δείκτη θέσης}
     endwhile
4    if A[I] = X                    {Αναγνώριση αποτελέσματος}
5        then print I               {Επιστροφή θέσης στοιχείου}
6        else print 'ΑΠΟΤΥΧΙΑ'      {Επιστροφή μηνύματος αποτυχίας}
     endif
```

Αυτή η διαφορά αντικατοπτρίζεται στον παραπάνω Αλγόριθμο Π2.3 (DIAT-GRAM-ANAZHTHSH), όπου η επανάληψη του σώματος του while (βήμα 3) εξακολουθεί, ενόσω η τιμή του τρέχοντος στοιχείου του πίνακα παραμένει μεγαλύτερη από τη ζητούμενη τιμή (A[I] > X) και δεν έχουμε φθάσει στο τέλος του πίνακα (I ≤ N). Δηλαδή, η επανάληψη σταματά όταν η τιμή του τρέχοντος στοιχείου είναι ίση με τη δοθείσα (επιτυχία) ή μεγαλύτερη από τη δοθείσα (αποτυχία) ή έχουμε εξαντλήσει τον πίνακα (αποτυχία). Το ποια από αυτές τις περιπτώσεις συνέβη, ανιχνεύεται από τη διάταξη if (βήματα 4-6).

Άσκηση Αυτοαξιολόγησης 2.3

Ενδεχομένως σ' έναν πίνακα να υπάρχουν περισσότερα του ενός στοιχεία που να έχουν τιμή ίδια με την προς αναζήτηση. Τροποποιήστε τον Αλγόριθμο 2.1 έτσι, ώστε να βρίσκει και να επιστρέφει τις θέσεις όλων των στοιχείων που έχουν τιμή ίδια με τη ζητούμενη. (*Σημείωση*: Αφού πρώτα προσπαθήσετε μόνοι σας να σχεδιάσετε τον τροποποιημένο αλγόριθμο γραμμικής αναζήτησης και διαπιστώσετε δυσκολία στη σύλληψη της βασικής ιδέας, ανατρέξτε στα 'Βοηθητικά Στοιχεία' που βρίσκονται στο τέλος του βιβλίου, πριν από την κυρίως απάντηση της άσκησης αυτής).

2.2.2 Δυαδική αναζήτηση

Η δυαδική αναζήτηση *προϋποθέτει* ότι τα στοιχεία του πίνακα είναι *διατεταγμένα*, δηλαδή βρίσκονται κατά αύξουσα (ή αλφαβητική) σειρά. Η λύση του προβλήματος κατά τη δυαδική αναζήτηση έχει ως εξής: Συγκρίνουμε την τιμή του δοθέντος στοιχείου με την τιμή του 'μεσαίου' στοιχείου του πίνακα. Αν το στοιχείο είναι το ζητούμενο τερματίζεται η αναζήτηση με επιτυχία. Αν όχι, τότε, αν η δοθείσα τιμή είναι μεγαλύτερη από αυτή του ‘μεσαίου’ στοιχείου, ερευνούμε με τον ίδιο τρόπο το τμήμα του πίνακα που είναι στα δεξιά του, αλλιώς αυτό που είναι στα αριστερά του. Αυτή η διαδικασία συνεχίζεται μέχρις ότου εντοπιστεί το ζητούμενο στοιχείο ή εξαντληθούν τα στοιχεία του πίνακα. Ο αντίστοιχος αλγόριθμος είναι ο Αλγόριθμος 2.2 (DIAD-ANAZHTHSH), που παρουσιάζεται στο επόμενο πλαίσιο.

Οι βοηθητικές μεταβλητές NL και NU, που αρχικοποιούνται στο βήμα 1, αντιπροσωπεύουν κάθε φορά τον μικρότερο και μεγαλύτερο δείκτη αντίστοιχα, δηλ. τα όρια του τμήματος του πίνακα που είναι υπό έρευνα, ενώ η μεταβλητή M τη μέση τιμή τους (βήμα 2). Οι τετραγωνικές παρενθέσεις '[ ]' σημαίνουν το ακέραιο μέρος (του αποτελέσματος) της παράστασης που βρίσκεται ανάμεσά τους (Στην Pascal, αυτό υλοποιείται με τη χρήση του τελεστή div.) Η διάταξη while (βήματα 3-7) ξεκινά μια επαναληπτική διαδικασία, σε κάθε επανάληψη της οποίας, ερευνάται και ένα τμήμα του πίνακα που είναι το ένα από τα δύο μισά του προηγουμένου. Η διάταξη if (βήματα 4-6) μέσα στην while, καθορίζει τα όρια του μισού του τρέχοντος τμήματος που θα γίνει το νέο υπό έρευνα τμήμα στην επόμενη επανάληψη. Η επανάληψη σταματά όταν, είτε δεν υπάρχει άλλο στοιχείο για εξέταση (NL > NU) (αποτυχία), είτε βρεθεί το ζητούμενο στοιχείο (A[I] = X) (επιτυχία). Η δεύτερη διάταξη if (βήματα 8-10) επιστρέφει το κατάλληλο αποτέλεσμα.

ΑΛΓΟΡΙΘΜΟΣ 2.2: ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

Είσοδος: Ένας πίνακας (A), το μέγεθος του πίνακα (N) και η προς αναζήτηση τιμή (X).
Έξοδος: Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα, σε περίπτωση επιτυχίας ή μήνυμα αποτυχίας. Εσωτερικά , δεν συμβαίνει τίποτα.

```
DIAD-ANAZHTHSH (A, N, X)
1    NL ← 1, NU ← N                          {Αρχικοποίηση μεταβλητών}
2    M ← [(NL+NU)/2]                         {Εύρεση δείκτη μεσαίου στοιχείου}
3    while (A[M] ≠ X) and (NL ≤ NU)          {Έλεγχος τέλους επανάληψης}
4        if A[M] < X                         {Σύγκριση στοιχείων}
5          then NL ← M + 1                   {Ενημέρωση κάτω ορίου}
6          else NU ← M - 1                   {Ενημέρωση πάνω ορίου}
         endif
7        M ← [(NL+NU)/2]                     {Ενημέρωση δείκτη μεσαίου στοιχείου}
     endwhile
8    if A[M] = X                             {Αναγνώριση αποτελέσματος}
9       then print M                         {Επιστροφή θέσης στοιχείου}
10      else print 'ΑΠΟΤΥΧΙΑ'                {Επιστροφή μηνύματος αποτυχίας}
   endif
```

Η πολυπλοκότητα της δυαδικής αναζήτησης, στη χειρότερη περίπτωση, είναι καλύτερη από αυτή της γραμμικής, δεδομένου ότι, σε κάθε επανάληψη ο αριθμός των συγκρίσεων μειώνεται στο μισό. Αποδεικνύεται ότι είναι $f(n) = O(\log n)$ και στη χειρότερη και στη μέση περίπτωση.

Παράδειγμα 2.4

Αναλυτική περιγραφή δυαδικής αναζήτησης με παράδειγμα

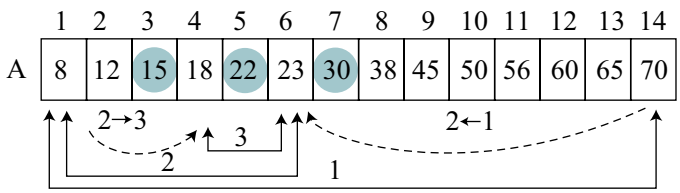
Θεωρούμε ότι μας δίνεται ο παρακάτω πίνακας A με διατεταγμένα στοιχεία και ζητείται να περιγράψουμε τα βήματα της εφαρμογής της δυαδικής αναζήτησης, στην περίπτωση αναζήτησης της τιμής X = 22. Επίσης, να μετρήσουμε τον αριθμό των συγκρίσεων που γίνονται και να τον συγκρίνουμε με αυτόν της γραμμικής αναζήτησης.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	8	12	15	18	22	23	30	38	45	50	56	60	65	70

Για την παρουσίαση των βημάτων του αλγορίθμου δυαδικής αναζήτησης, χρησιμοποιούμε τον παρακάτω πίνακα όπου φαίνονται οι τιμές των εμπλεκομένων στοιχείων (μεταβλητών κλπ) του Αλγορίθμου 2.2.

Βήμα	A[M]	A[M]≠X	NL≤NU	NL	NU	M
0	-	-	-	1	14	7
1	30	TRUE	TRUE	1	6	3
2	15	TRUE	TRUE	4	6	5
3	22	FALSE	TRUE	4	6	5

Το βήμα 0 αναφέρεται στην αρχικοποίηση των μεταβλητών. Η κάθε σειρά, στον παραπάνω πίνακα, περιέχει τις τιμές των αντίστοιχων στοιχείων (εκφράσεων ή μεταβλητών) μετά την εκτέλεση του αντίστοιχου βήματος, δηλαδή της αντίστοιχης επανάληψης του σώματος της διάταξης while του Αλγορίθμου 2.2. Παρατηρήστε ότι, μετά την εκτέλεση του τρίτου βήματος σταματά η επανάληψη, διότι τότε A[M] = X (A[5] = X = 22). Η διαδικασία απεικονίζεται γραφικά στο παρακάτω σχήμα όπου, με συνεχή αριθμημένα διπλά βέλη καταδεικνύονται τα υπό εξέταση τμήματα του πίνακα στα αντίστοιχα βήματα, και με διακεκομμένα απλά βέλη οι μεταβολές των άκρων (ορίων) των τμημάτων στις αλλαγές βήματος. Επίσης, κάθε ● δείχνει το μεσαίο στοιχείο ενός τμήματος.



Όπως είναι φανερό, γίνονται μόνο τρεις συγκρίσεις, ενώ για την εφαρμογή του αλγορίθμου γραμμικής αναζήτησης θα απαιτούνταν πέντε συγκρίσεις, δηλαδή τόσες όσος είναι ο αριθμός των στοιχείων από την αρχή του πίνακα μέχρι και το ζητούμενο. Η διαφορά δεν φαίνεται μεγάλη, όμως, αν το στοιχείο βρισκόταν σε θέση δεξιά από τη μέση του πίνακα, θα ήταν μεγαλύτερη. Γενικά, όσο πιο κοντά στην αρχή του πίνακα βρίσκεται ένα στοιχείο, τόσο πιο πλεονεκτική γίνεται η γραμμική αναζήτηση.

Άσκηση Αυτοαξιολόγησης 2.4

Ο αλγόριθμος δυαδικής αναζήτησης στηρίζεται στη διχοτόμηση του πίνακα ή του υπό εξέταση τμήματός του. Με βάση τον αλγόριθμο δυαδικής αναζήτησης σχεδιάστε ένα αλγόριθμο 'τριαδικής' αναζήτησης, δηλαδή ένα αλγόριθμο που να τριχοτομεί τον πίνακα ή το υπό εξέταση τμήμα του. Στην περίπτωση αυτή, επιλέγουμε ένα από τα τρία τμήματα και προχωρούμε τριχοτομώντας το, έως ότου είτε βρούμε το ζητούμενο στοιχείο είτε αποτύχουμε.

2.3 Εγγραφές

Όπως αναφέραμε και στο Κεφάλαιο 1, η *εγγραφή* (*record*) είναι μια θεμελιώδης δομή δεδομένων που υπάρχει ενσωματωμένη σε πολλές γλώσσες προγραμματισμού υψηλού επιπέδου. Είναι μια σημαντική δομή που χρησιμοποιείται σαν δομικό στοιχείο για άλλες ανώτερες δομές.

2.3.1 Ορισμός και αναπαράσταση

Εγγραφή είναι μια συλλογή ενός πεπερασμένου πλήθους αλληλοσχετιζόμενων στοιχείων που, συνήθως, δεν είναι του ίδιου τύπου, αντίθετα με ότι συμβαίνει σ' ένα πίνακα. Λόγω αυτού του χαρακτηριστικού, η εγγραφή θεωρείται ως *ετερογενής δομή* (*heterogeneous structure*).

Κάθε στοιχείο μιας εγγραφής περιγράφεται από ένα *πεδίο* (*field*). Κάθε πεδίο έχει ένα *όνομα* (*identifier*) και ένα *τύπο* (*type*). Κάθε στοιχείο μπορεί να συνίσταται από άλλα, μερικά στοιχεία. Τότε καλείται *σύνθετο στοιχείο*, ενώ στην αντίθετη περίπτωση *απλό στοιχείο*. Κατ' αντιστοιχία, διακρίνουμε *απλά πεδία* και *σύνθετα πεδία*. ■

■ Μια **εγγραφή** ορίζεται από μια πεπερασμένη πλειάδα N στοιχείων (ή αλλιώς N-άδα), όχι απαραίτητα του ίδιου τύπου, και από μια πεπερασμένη πλειάδα N (N-άδα) διακεκριμένων συμβόλων (πεδίων), τέτοιες ώστε να υπάρχει μια αντιστοιχία (σχέση) μεταξύ των ομοθέσιων στοιχείων τους.

Έστω μια εγγραφή (ή μεταβλητή τύπου εγγραφής) R και ($fd_1, fd_2, \ldots, fd_n$) η αντίστοιχη n-άδα πεδίων με τύπους (δηλ. πεδία τιμών) $T_1, T_2, \ldots, T_n$, αντίστοιχα. Τότε, θα αναφερόμαστε στα στοιχεία της R με τους συμβολισμούς $fd_1(R), fd_2(R), \ldots, fd_n(R)$. Δηλαδή, τα πεδία λειτουργούν σαν δείκτες των στοιχείων μιας εγγραφής και, δεδομένου ότι είναι διαφόρων τύπων, δεν τίθεται θέμα διάταξης των στοιχείων μιας εγγραφής με βάση την τιμή τους, παρά μόνο με βάση τη θέση τους. Το πεδίο τιμών της εγγραφής R είναι το σύνολο $T = T_1 \times T_2 \times \ldots \times T_n$, δηλαδή το καρτεσιανό γινόμενο των πεδίων τιμών των πεδίων της εγγραφής.

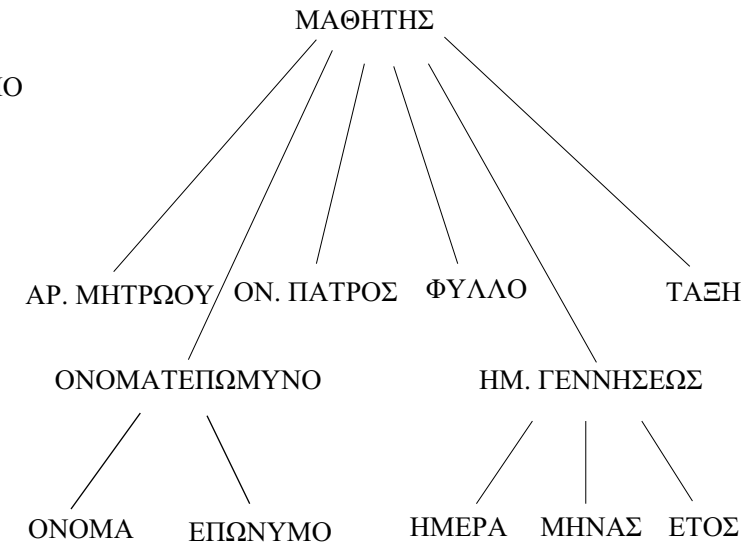
Αν κάποιο πεδίο fd_x είναι ένα σύνθετο πεδίο με μερικά (απλά) πεδία ($fd_x^1, fd_x^2, \ldots, fd_x^m$), τότε θα αναφερόμαστε στα αντίστοιχα απλά στοιχεία με τους συμβολισμούς $fd_x^1(fd_x(R)), fd_x^2(fd_x(R)), \ldots, fd_x^m(fd_x(R))$. Ο καθένας από τους παραπάνω συμβολισμούς λειτουργεί ως μεταβλητή. Τα πεδία $fd_1, fd_2, \ldots, fd_n$ καλούνται πεδία πρώτου επιπέδου, τα $fd_x^1, fd_x^2, \ldots, fd_x^m$ δεύτερου επιπέδου κ.ο.κ.

Λόγω της ύπαρξης σύνθετων πεδίων διαφόρου βάθους επιπέδων, το οργανωτικό σχήμα μιας εγγραφής απεικονίζεται σαν μια ιεραρχία πεδίων.

Παράδειγμα 2.5 *Παράδειγμα-απεικόνιση εγγραφής*

Ένα σχολείο κρατά τα στοιχεία κάθε μαθητή σε μια εγγραφή με τα εξής πεδία: ΑΡ.ΜΗΤΡΩΟΥ, ΟΝΟΜΑΤΕΠΩΝΥΜΟ, ΟΝ.ΠΑΤΡΟΣ, ΦΥΛΛΟ, ΗΜ.ΓΕΝΝΗΣΕΩΣ, ΤΑΞΗ. Το ΟΝΟΜΑΤΕΠΩΝΥΜΟ είναι ένα σύνθετο πεδίο με μερικά πεδία τα ΟΝΟΜΑ και ΕΠΩΝΥΜΟ. Επίσης, το πεδίο ΗΜΕΡ.ΓΕΝΝΗΣΕΩΣ είναι ένα σύνθετο πεδίο με μερικά πεδία τα ΗΜΕΡΑ, ΜΗΝΑΣ, ΕΤΟΣ. Η εγγραφή αυτή απεικονίζεται παρακάτω με δύο τρόπους, απ' όπου φαίνεται η ιεραρχική δομή της.

```
ΜΑΘΗΤΗΣ
   ΑΡ. ΜΗΤΡΟΟΥ
   ΟΝΟΜΑΤΕΠΩΝΥΜΟ
      ΟΝΟΜΑ
      ΕΠΩΝΥΜΟ
   ΟΝ. ΠΑΤΡΟΣ
   ΦΥΛΛΟ
   ΗΜ. ΓΕΝΝΗΣΕΩΣ
      ΗΜΕΡΑ
      ΜΗΝΑΣ
      ΕΤΟΣ
   ΤΑΞΗ
```



Τα πεδία ΑΡ.ΜΗΤΡΩΟΥ, ΟΝΟΜΑΤΕΠΩΝΥΜΟ, ΟΝ.ΠΑΤΡΟΣ, ΦΥΛΛΟ, ΗΜ.ΓΕΝΝΗΣΗΣ και ΤΑΞΗ είναι πεδία πρώτου επιπέδου, ενώ τα ΟΝΟΜΑ, ΕΠΩΝΥΜΟ, ΗΜΕΡΑ, ΜΗΝΑΣ και ΕΤΟΣ, δεύτε-

ρου επιπέδου. Αν θέλουμε να αναφερθούμε στην τάξη ενός μαθητή, τότε χρησιμοποιούμε το συμβολισμό ΤΑΞΗ(ΜΑΘΗΤΗΣ), ενώ για να αναφερθούμε στο όνομα του μαθητή, τον συμβολισμό ΟΝΟΜΑ(ΟΝΟΜΑΤΕΠΩΝΥΜΟ(ΜΑΘΗΤΗΣ)). Κάθε ένα από τα απλά πεδία έχει σαν τιμή το αντίστοιχο στοιχείο (πληροφορία) για τον μαθητή.

Οι βασικές πράξεις σε μια εγγραφή, όπως και σε έναν πίνακα, είναι η *προσπέλαση* (ή *ανάκτηση*) στοιχείου και η *καταχώρηση* (ή *ενημέρωση*) στοιχείου. Στην αλγοριθμική μας γλώσσα αυτές εκφράζονται ως εξής:

$$X \leftarrow FDX(R) \text{ (προσπέλαση ή ανάκτηση)}$$

$$FDX(R) \leftarrow X \text{ (καταχώρηση ή ενημέρωση)}$$

όπου FDX είναι ένα πεδίο πρώτου επιπέδου.

Παράδειγμα 2.6α

Ορισμός εγγραφής στην Pascal

Ας θεωρήσουμε μια απλοποιημένη εκδοχή της εγγραφής του Παραδείγματος 2.5, όπου ένας μαθητής χαρακτηρίζεται μόνο από τον αριθμό μητρώου, το ονοματεπώνυμό του και το όνομα του πατέρα του. Η εγγραφή αυτή μπορεί να υλοποιηθεί στην Pascal, όπως παρουσιάζεται παρακάτω.

```
type
  onmateponrec = record
                   onoma:array [1..12] of char;
                   eponymo: array [1..24] of char;
                 end;

  mathitisrec = record
                  armitrwou: integer;
                  onomatepon: onomateponrec;
                  onpatros: array [1..12] of char:
                end;

var mathitis: mathitisrec;
```

Με τις παραπάνω εκφράσεις, ορίζουμε δύο τύπους δεδομένων με ονόματα oneponrec και mathitisrec που η κάθε μια είναι μια εγγραφή (record). Η δεύτερη, είναι η κυρίως εγγραφή που χρησιμοποιεί την πρώτη για να δηλώσει τον τύπο του ενός σύνθετου πεδίου της (onepon). Η πρώτη δομή εγγραφής (oneponrec) έχει δύο απλά πεδία (onoma, eponymo). Το πρώτο, δηλώνεται σαν ένας πίνακας 12 χαρακτήρων, ενώ το δεύτερο, σαν πίνακας 24 χαρακτήρων. Αυτός είναι ο τρόπος που δηλώνουμε ακολουθίες χαρακτήρων (π.χ. λέξεις) στην Pascal. Η κυρίως εγγραφή έχει δύο απλά πεδία (armitrwou και onpatros) και ένα σύνθετο (onomatepon). Το πρώτο δηλώνεται τύπου 'ακέραιος' (integer), ενώ το δεύτερο σαν ακολουθία 12 χαρακτήρων. Το σύνθετο πεδίο δηλώνεται τύπου (εγγραφής) 'onomateponrec'. Τέλος, δηλώνουμε μια μεταβλητή (mathitis) τύπου (εγγραφής) mathitisrec, στην οποία μπορούμε να εισάγουμε τιμές για τα διάφορα πεδία της εγγραφής.

Στην Pascal για να προσπελάσουμε το πεδίο 'armitrwou' της μεταβλητής 'mathitis' γράφουμε mathitis.armitrwou. Επίσης, προκειμένου για το πεδίο 'onoma' γράφουμε mathitis.onepon.onoma. Αυτές οι εκφράσεις λειτουργούν σαν μεταβλητές, στις οποίες μπορούμε να καταχωρήσουμε μια τιμή ή να καταχωρήσουμε την τιμή τους σε κάποια άλλη μεταβλητή του ίδιου τύπου.

Ορισμός εγγραφής στη C

Η εγγραφή του Παραδείγματος 2.6α μπορεί να υλοποιηθεί στην C ως εξής:

```
typedef
    struct {
        char onoma [12];
        char eponymo [24];
          } onomateponrec;
```

Παράδειγμα 2.6β

```
typedef
    struct {
        int armitrwou;
        onomateponrec onomatepon;
        char onpatros [12];
          } mathitisrec;

mathitisrec mathitis;
```

Ορίζονται δύο νέοι τύποι (onomateponrec, mathitisrec) που είναι δύο δομές (struct) της C και η καθεμία παριστάνει μια εγγραφή. Η δεύτερη, είναι η κυρίως εγγραφή (δομή) που χρησιμοποιεί την πρώτη στη δήλωση του τύπου ενός σύνθετου πεδίου (μέλους) της (onomatepon). Το int δηλώνει τον ακέραιο τύπο. Για τα υπόλοιπα, ισχύουν παρόμοια με αυτά στην Pascal.

2.3.2 Πίνακες εγγραφών

Πολλές φορές, στην πράξη χρησιμοποιούνται δομές που είναι συνδυασμοί δύο άλλων δομών. Αυτές οι δομές ονομάζονται *συνδυασμένες* ή *μεικτές δομές* (*composite structures*). Μια συνήθης και χρήσιμη περίπτωση μεικτής δομής είναι ο *πίνακας εγγραφών* (*array of records*), δηλαδή ένας πίνακας που τα στοιχεία του είναι εγγραφές. Συνήθως, κάθε εγγραφή έχει ένα πεδίο που ονομάζεται *κλειδί* (*key*) και προσδιορίζει με μοναδικό τρόπο κάθε στιγμιότυπο της εγγραφής. Δηλαδή, δεν υπάρχουν δύο στιγμιότυπα μιας εγγραφής στις οποίες το κλειδί να έχει την ίδια τιμή. Στιγμιότυπο λέγεται κάθε συγκεκριμένη εγγραφή. Π.χ. κάθε εγγραφή που έχει στοιχεία ενός συγκεκριμένου μαθητή είναι στιγμιότυπο της εγγραφής ΜΑΘΗΤΗΣ (Παράδειγμα 2.5). Ένα κλειδί στην εγγραφή ΜΑΘΗΤΗΣ είναι το ΑΡ.ΜΗΤΡΩΟΥ. Η αναζήτηση στα στοιχεία ενός τέτοιου πίνακα γίνεται με βάση το κλειδί των εγγραφών-στοιχείων του πίνακα.

Παράδειγμα 2.7 *Γραμμική αναζήτηση σε διατεταγμένο πίνακα εγγραφών*

Θεωρούμε έναν πίνακα εγγραφών, όπου συμβολίζουμε με KLEIDI το πεδίο που αποτελεί το κλειδί κάθε εγγραφής. Επίσης, θεωρούμε ότι ο πίνακας είναι διατεταγμένος με βάση το κλειδί κάθε εγγραφής. Το ζητούμενο είναι να σχεδιάσουμε έναν αλγόριθμο γραμμικής αναζήτησης για ένα τέτοιο πίνακα.

Ο ζητούμενος αλγόριθμος αποτελεί μια παραλλαγή του Αλγορίθμου Π2.3. Το σημείο παραλλαγής προέρχεται από το γεγονός ότι τα στοιχεία του πίνακα τώρα είναι εγγραφές και η τιμή που αναζητούμε αφορά το κλειδί κάθε εγγραφής. Έτσι, ο νέος αλγόριθμος προκύπτει από τον Π2.3, αν όπου Α[Ι] θέσουμε KLEIDI(Α[Ι]).

Σύνοψη

Ο ***πίνακας*** και η ***εγγραφή*** είναι δύο θεμελιώδεις δομές, στις οποίες στηρίζεται η υλοποίηση άλλων ανώτερων δομών. Και οι δύο δομές αποτελούν συλλογές στοιχείων. Κάθε στοιχείο ενός πίνακα αντιστοιχεί σ' ένα ***δείκτη***, από ένα σύνολο δεικτών, ενώ κάθε στοιχείο μιας εγγραφής σ' ένα ***πεδίο*** από μια πλειάδα πεδίων. Τα στοιχεία ενός πίνακα είναι του ίδιου τύπου και η σχέση μεταξύ τους γραμμική. Γι' αυτό, ο πίνακας είναι μια ***ομογενής*** και ***γραμμική δομή***. Αντίθετα, τα στοιχεία μιας εγγραφής δεν είναι, συνήθως, του ίδιου τύπου και η σχέση μεταξύ τους είναι ιεραρχική. Γι' αυτό, η εγγραφή είναι μια ***ετερογενής*** και ***μη γραμμική δομή***. Και οι δύο, όμως, είναι ***στατικές δομές***, δηλαδή το μέγεθός τους δεν μεταβάλλεται κατά την εκτέλεση του προγράμματος.

Τα στοιχεία ενός πίνακα αποθηκεύονται σε γειτονικές θέσεις στη μνήμη του Η/Υ. Αυτό δίνει τη δυνατότητα στον πίνακα να είναι ***δομή τυχαίας προσπέλασης***, δηλαδή ο χρόνος προσπέλασης ενός στοιχείου είναι ανεξάρτητος από τη θέση του στοιχείου. Ο υπολογισμός της διεύθυνσης ενός στοιχείου στη μνήμη, από ένα μεταφραστικό πρόγραμμα, γίνεται με τη ***συνάρτηση απεικόνισης πίνακα*** (ΣΑΠ), που είναι μια γραμμική συνάρτηση του δείκτη του στοιχείου. Το ίδιο συμβαίνει και στους ***δισδιάστατους*** και στους ***τρισδιάστατους*** και, εν γένει, στους ***πολυδιάστατους πίνακες***, μόνο που τώρα αντί για ένα

δείκτη έχουμε ζεύγος, τριάδα, …, πλειάδα δεικτών αντίστοιχα για κάθε στοιχείο. Πλην των πολυδιάστατων πινάκων, υπάρχουν και οι ειδικοί πίνακες, όπως ο ***συμμετρικός***, ο ***τριδιαγώνιος***, ο ***τριγωνικός*** και ο ***αραιός*** πίνακας. Οι πίνακες αυτοί έχουν κάποιες ιδιαιτερότητες και χρήζουν ειδικής αντιμετώπισης.

Και σε μια εγγραφή όμως, δεδομένου ότι τα πεδία λειτουργούν σαν δείκτες, έχουμε τυχαία προσπέλαση σε κάθε πεδίο.

Η ***αναζήτηση*** είναι μια από τις πράξεις σ' ένα πίνακα που μας ενδιαφέρει. Η ***γραμμική αναζήτηση*** είναι μια μέθοδος αναζήτησης που ψάχνει ένα-ένα τα στοιχεία ενός πίνακα από την αρχή μέχρι το τέλος, έως ότου είτε βρει το ζητούμενο στοιχείο ή αποτύχει. Η ***δυαδική αναζήτηση***, που εφαρμόζεται μόνο σε διατεταγμένους πίνακες, στηρίζεται σε διαδοχικές διχοτομήσεις του πίνακα. Εν γένει, η δυαδική αναζήτηση έχει καλύτερη χρονική απόδοση (μικρότερη πολυπλοκότητα) από τη γραμμική.

Ο ***πίνακας εγγραφών*** είναι μια ενδιαφέρουσα περίπτωση ***μεικτής δομής***, όπου κάθε στοιχείο του πίνακα είναι ένα στιγμιότυπο μιας εγγραφής. Εδώ, η αναζήτηση γίνεται με βάση το ***κλειδί*** των εγγραφών, το πεδίο δηλαδή που έχει μοναδική τιμή σε κάθε στιγμιότυπο.

Οδηγός για περαιτέρω μελέτη

1. Κοίλιας, Χ.**,** *Δομές Δεδομένων και Οργανώσεις Αρχείων*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1993.

   Στο κεφάλαιο 5 του βιβλίου αυτού με τίτλο «Αναζήτηση» υπάρχει μια παρουσίαση διαφόρων μεθόδων αναζήτησης σε πίνακα, όπου εκτός από την γραμμική και δυαδική αναζήτηση παρουσιάζονται και η αναζήτηση κατά ομάδες, η αναζήτηση Fibonacci και η αναζήτηση παρεμβολής.

2. Kruse, L. R**.,** *Data Structures and Program Design*, 2nd Edition, Prentice Hall, Englewood Cliffs, NJ 1991 (Υπάρχει και 3η έκδοση του βιβλίου, 1994).

   Στην ενότητα «6.2 Rectangular Arrays», στο τμήμα με τίτλο «3. Variation: Access Tables» υπάρχει μια σύντομη παρουσίαση για τους πίνακες προσπέλασης. Επίσης, στην ενότητα «6.3 Tables of

various shapes» υπάρχουν στοιχεία για τους ειδικούς πίνακες. Τέλος, στην ενότητα «5.2 Sequential Search», στο τμήμα με τίτλο «4. Analysis of Sequential Search» υπάρχει η ανάλυση της πολυπλοκότητας της γραμμικής αναζήτησης.

3. Μανωλόπουλος, Ι., *Δομές Δεδομένων*, τόμ. Α', ART of TEXT, 2η έκδοση, Θεσσαλονίκη 1992.

   Στην ενότητα «2.3 Πίνακες Προσπέλασης» του βιβλίου αυτού μπορείτε να βρείτε μια συνοπτική παρουσίαση για τους 'πίνακες προσπέλασης' και στην ενότητα «2.4 Ειδικές Μορφές Πινάκων» μια συνοπτική, επίσης, διαπραγμάτευση για τους ειδικούς πίνακες. Τέλος, στο κεφάλαιο 7 με τίτλο «Αναζήτηση» υπάρχει μια παρουσίαση διαφόρων μεθόδων αναζήτησης, όπως σειριακή, δυαδική, Fibonacci, παρεμβολής και αναζήτηση ανά ομάδες.

4. Standish, A. T., *Data Structures*, *Algorithm and Software Principles*, Addison-Wesley, Reading, MA 1994.

   Στην ενότητα 6.5 Analyzing Simple Algorithms του βιβλίου αυτού, στο τμήμα της με τίτλο «Analysis of Sequential Search» υπάρχει μια αναλυτική εξέταση της πολυπλοκότητας της γραμμικής αναζήτησης και στο τμήμα με τίτλο «Analysis of Binary Search» της δυαδικής αναζήτησης.

5. Stubbs, F. D. και Webre, W. N., *Data Structures with Abstract Data Types and Pascal*, 2nd Edition, Brooks/Cole (Wadsworth), Pacific Grove, CA 1989.

   Στην υποενότητα «2.2.4 Special Arrays» του βιβλίου αυτού μπορείτε να βρείτε μια λεπτομερέστερη διαπραγμάτευση για τους ειδικούς πίνακες, στην ενότητα «2.3 Records» για τις εγγραφές και στην ενότητα «2.5 Composite Structures» για τις μεικτές δομές.

Γενικές Λίστες

ΚΕΦΑΛΑΙΟ 3

Σκοπός

Στο κεφάλαιο αυτό παρουσιάζονται οι δύο βασικοί τύποι γενικών λιστών, οι 'συνεχόμενες' και οι 'συνδεδεμένες' λίστες. Η παρουσίαση, κυρίως αφορά τον ορισμό των δομών αυτών, τις πράξεις σ' αυτές και τους αντίστοιχους αλγορίθμους υλοποίησής τους.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό, θα είστε σε θέση να:

- *περιγράφετε τα πλεονεκτήματα και μειονεκτήματα της συνεχόμενης και συνδεδεμένης αναπαράστασης μιας λίστας,*
- *σχεδιάζετε τροποποιήσεις, επεκτάσεις ή συνδυασμούς των βασικών αλγορίθμων των πράξεων σε συνεχόμενες λίστες,*
- *σχεδιάζετε τροποποιήσεις επεκτάσεις ή συνδυασμούς των βασικών αλγορίθμων των πράξεων σε συνδεδεμένες λίστες.*

Έννοιες κλειδιά

- *Γενική Λίστα*
- *Δομή Σειριακής Προσπέλασης*
- *Συνεχόμενη Αναπαράσταση*
- *Συνδεδεμένη Αναπαράσταση*
- *Συνεχόμενη Λίστα*
- *Απλά Συνδεδεμένη Λίστα*
- *Κεφαλή*
- *Κόμβος*
- *Σύνδεσμος*
- *Δείκτης*
- *Μεταβλητή Δείκτη*
- *Διαπέραση Λίστας*
- *Εισαγωγή Στοιχείου*
- *Διαγραφή Στοιχείου*

Ενότητες Κεφαλαίου 3

3.1 Ορισμός, αναπαράσταση και τύποι λίστας

3.2 Πράξεις σε συνεχόμενη λίστα

3.3 Πράξεις σε απλά συνδεδεμένη λίστα

3.4 Άλλες κατηγορίες συνδεδεμένης λίστας

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό διαπραγματευόμαστε τις γενικές λίστες, μια κατηγορία ανώτερων δομών δεδομένων. Στην πρώτη ενότητα, προσδιορίζεται η έννοια της λίστας σε αντιπαράθεση με αυτή του πίνακα. Επίσης, παρουσιάζονται οι διάφοροι τύποι λιστών και περιγράφονται οι αναπαραστάσεις των δύο βασικών τύπων, της συνεχόμενης και της απλά συνδεδεμένης λίστας.

Στη δεύτερη και τρίτη ενότητα, παρουσιάζουμε τους βασικούς αλγορίθμους για τις πράξεις διαπέραση, εισαγωγή και διαγραφή σε συνεχόμενη και συνδεδεμένη λίστα αντίστοιχα. Επίσης, δίνονται υποδείξεις για την πράξη της αναζήτησης στις λίστες αυτές. Εδώ, θεωρούμε σκόπιμο να κάνουμε την εξής διευκρίνιση: Υπάρχει ένα σύνολο πράξεων σε μια λίστα που αποτελεί μέρος του ορισμού της ως δομής δεδομένων, όπως εξηγήσαμε στην υποενότητα "1.1.2 Τύποι και Δομές Δεδομένων" στο Κεφάλαιο 1 . Τέτοιες πράξεις, είναι η δημιουργία λίστας, ο έλεγχος κενής λίστας, ο έλεγχος γεμάτης λίστας, η εισαγωγή στοιχείου, η διαγραφή στοιχείου, κλπ. Πράξεις σαν τις τρεις πρώτες, θα μπορούσαμε να τις χαρακτηρίσουμε ως 'στοιχειώδεις'. Η υλοποίησή τους είναι σχετικά απλή και συνδέεται άμεσα με την αντίστοιχη γλώσσα προγραμματισμού. Στο βιβλίο αυτό, δε θα ασχοληθούμε με αυτές τις στοιχειώδεις πράξεις, αλλά με τις υπόλοιπες, που τις χαρακτηρίσαμε ως 'κυριότερες', στην υποενότητα '1.1.5 Υλοποίηση Δομών Δεδομένων' του Κεφαλαίου 1.

Τέλος, στην τέταρτη ενότητα, παρουσιάζουμε συνοπτικά ορισμένες άλλες κατηγορίες συνδεδεμένης λίστας, όπως η διπλά συνδεδεμένη λίστα και η κυκλική λίστα.

3.1 Ορισμός, αναπαράσταση και τύποι λιστών

3.1.1 Έννοια και τύποι λίστας

Στην καθημερινή μας ζωή, πολλές φορές, χρησιμοποιούμε την έννοια της λίστας. Για παράδειγμα, χρησιμοποιούμε λίστες αγορών, εργασιών κλπ. Ας θεωρήσουμε μια λίστα εργασιών που πρέπει να διεκπεραιώσουμε σε μια ημέρα. Οι εργασίες αυτές συνήθως αναγράφονται με κάποια σειρά, π.χ. τέτοια, ώστε να ελαχιστοποιεί το συνολικό χρόνο διεκπεραίωσης. Έτσι, υπάρχει η πρώτη, η δεύτερη, η τρίτη κλπ και η τελευταία εργασία. Έχουμε δε κατά νου, να διέλθουμε όλη τη λίστα ξεκινώντας από την πρώτη εργασία και καταλήγοντας στην τελευταία για να τις διεκπεραιώσουμε. Κατά τη διάρκεια της ημέρας όμως, θα χρειαστεί είτε να σβήσουμε κάποιες εργασίες που πραγματοποιήθηκαν είτε να εισάγουμε κάποιες νέες που προέκυψαν εν τω μεταξύ είτε και να μετακινήσουμε κάποιες εργασίες σε άλλες θέσεις.

Το παραπάνω παράδειγμα αντανακλά με αρκετή ακρίβεια την έννοια της λίστας ■

■ Μια **λίστα** είναι μια πεπερασμένη ακολουθία στοιχείων του ίδιου τύπου (τύπος βάσης).

Η *ακολουθία* (*sequence*) είναι μια μαθηματική έννοια που αναφέρεται σε μια συλλογή θεωρητικά άπειρων στοιχείων. Μια πεπερασμένη ακολουθία L, n στοιχείων, είναι μια πλειάδα *n στοιχείων* (ή αλλιώς μια *n-άδα στοιχείων*) $(L_1, L_2, \ldots, L_n)$, όπου L_1 είναι το πρώτο στοιχείο και L_n το τελευταίο. Κάθε στοιχείο L_i έχει ένα προηγούμενο (predecessor) στοιχείο (L_{i-1}) και ένα επόμενο (successor) στοιχείο (L_{i+1}), πλην του πρώτου, που έχει μόνο επόμενο, και του τελευταίου, που έχει μόνο προηγούμενο.

Η *θέση* (*position*) ενός στοιχείου σε μια λίστα (ακολουθία) είναι χαρακτηριστική ιδιότητα του στοιχείου. Υπάρχει, δηλαδή, μια διάταξη σε μια λίστα που δεν καθορίζεται από την τιμή του στοιχείου αλλά από τη θέση του. Η θέση του στοιχείου L_i είναι η 'i'. Το πρώτο στοιχείο μιας λίστας ονομάζεται *κεφαλή* (*head*). Μια λίστα με κανένα στοιχείο ονομάζεται *κενή λίστα* (*null list*).

Από τα παραπάνω, εκ πρώτης όψεως, ίσως να συμπεραίνεται ότι η λίστα L δεν είναι κάτι διαφορετικό από ένα πίνακα L με $I = \{1, \ldots, n\}$. Όμως, δεν είναι έτσι. Υπάρχουν σημαντικές διαφορές μεταξύ τους, που πηγάζουν από το γεγονός ότι ο πίνακας στηρίζεται στις έννοιες

του συνόλου και της απεικόνισης (συνάρτησης), ενώ η λίστα στηρίζεται στην έννοια της ακολουθίας. Έτσι, ένας πίνακας θεωρείται σαν μια δομή τυχαίας προσπέλασης, όπως είδαμε στην ενότητα 2.1, ενώ μια λίστα είναι στην ουσία μια δομή *ακολουθιακής* ή *σειριακής προσπέλασης* (*sequential access*). Για να φθάσουμε, δηλαδή, σ' ένα στοιχείο μιας λίστας πρέπει να περάσουμε από όλα τα προηγούμενα ξεκινώντας από το πρώτο. Δεύτερον, το μέγεθος ενός πίνακα παραμένει σταθερό, ενώ το μέγεθος μιας λίστας όχι, διότι συνήθως απαιτούνται διαγραφές παλαιών και εισαγωγές νέων στοιχείων. Δηλαδή, μια λίστα είναι μια *δυναμική δομή* και όχι στατική, όπως ο πίνακας.

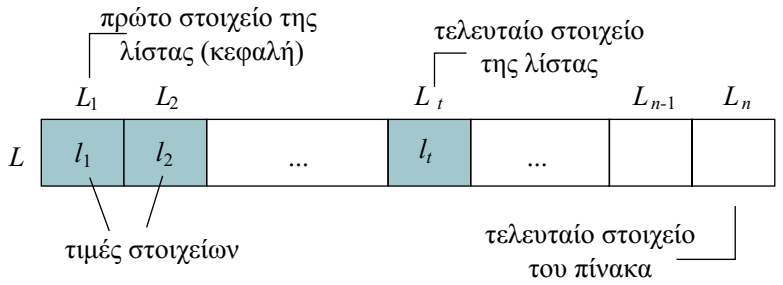
Τις λίστες μπορούμε να τις διακρίνουμε σε δύο μεγάλες κατηγορίες, ανάλογα με το βαθμό ελευθερίας που παρέχουν στην εφαρμογή των διαφόρων πράξεων σ' αυτές, τις *γενικές λίστες* (*general lists*)[1] και τις *ειδικές λίστες* (*special lists*). Οι γενικές λίστες επιτρέπουν την εισαγωγή και διαγραφή στοιχείων/κόμβων σε οποιαδήποτε θέση στη λίστα, ενώ οι ειδικές θέτουν περιορισμούς. Ειδικές λίστες είναι η 'στοίβα' και η 'ουρά', που εξετάζονται στο επόμενο κεφάλαιο.

Υπάρχουν δύο τρόποι αναπαράστασης μιας λίστας είτε γενικής είτε ειδικής. Με τον πρώτο τρόπο, την *συνεχόμενη αναπαράσταση* (*contiguous representation*), τα στοιχεία της λίστας καταλαμβάνουν γειτονικές ή συνεχόμενες (contiguous) θέσεις στη μνήμη. Εδώ, η μετάβαση από το ένα στοιχείο στο επόμενο γίνεται με τη μετάβαση στην επόμενη θέση μνήμης. Με το δεύτερο τρόπο, την *συνδεδεμένη αναπαράσταση* (*linked representation*), τα στοιχεία της λίστας αποθηκεύονται σε μη διαδοχικές θέσεις στη μνήμη του Η/Υ. Τώρα, η μετάβαση από το ένα στοιχείο στο επόμενο, πραγματοποιείται με τη χρήση ενός συνδέσμου (link) μεταξύ των στοιχείων. Αντίστοιχα, διακρίνουμε δύο κατηγορίες γενικών λιστών, τις *συνεχόμενες λίστες* (*contiguous lists*) και τις *συνδεδεμένες λίστες* (*linked lists*).

[1] Δεν πρέπει να συγχέεται με τον όρο 'γενικευμένες λίστες' (generalized lists), που αναφέρεται σε λίστες που μπορούν να έχουν σαν στοιχεία άλλες λίστες.

3.1.2 Αναπαράσταση συνεχόμενης λίστας

Η αναπαράσταση μιας συνεχόμενης λίστας L γίνεται μέσω ενός μονοδιάστατου πίνακα. Στην περίπτωση αυτή, τα στοιχεία της λίστας καταλαμβάνουν μέρος του πίνακα, έτσι, ώστε να είναι δυνατόν να μεταβάλλεται το μέγεθος της λίστας, όπως απεικονίζεται στο Σχήμα 3.1[2], όπου χρησιμοποιούμε ένα πίνακα L με $I = \{1, \ldots, n\}$. Το μέγεθος του πίνακα πρέπει να είναι μεγαλύτερο από ή ίσο με το πιθανό μέγιστο μέγεθος της λίστας που αναπαριστά. Δηλαδή, το τελευταίο στοιχείο της λίστας (L_t) πρέπει να βρίσκεται κάθε φορά πριν από την τελευταία θέση του πίνακα (n), ώστε να υπάρχει περιθώριο μεταβολής του μεγέθους της λίστας, λόγω εισαγωγής νέων στοιχείων.



Σχήμα 3.1
Αναπαράσταση συνεχόμενης λίστας

Η δυσκολία στην περίπτωση αυτή είναι ο καθορισμός του μεγέθους του πίνακα. Αν είναι σχετικά μικρό, τότε ενδεχομένως, θα έχουμε *υπερχείλιση* (*overflow*), δηλαδή χάσιμο, στοιχείων της λίστας σε κάποιες μετακινήσεις. Αν είναι σχετικά μεγάλο, τότε θα έχουμε σπατάλη χώρου στη μνήμη, διότι μέρος του πίνακα θα μένει αχρησιμοποίητο. Επίσης, πρέπει να υπάρχει κάποια μεταβλητή που να περιέχει κάθε φορά το δείκτη του τελευταίου στοιχείου της λίστας για να γνωρίζουμε το τέλος της. Τέλος, αξίζει να σημειωθεί ότι, λόγω της υλοποίησης της συνεχόμενης λίστας μέσω ενός πίνακα, έχουμε δυνατότητα άμεσης προσπέλασης των στοιχείων της.

Συνήθως, σε μια γλώσσα υψηλού επιπέδου μια συνεχόμενη λίστα ορίζεται σαν μια εγγραφή με δύο πεδία, από τα οποία το ένα είναι ένας πίνακας κάποιου μεγέθους και το άλλο είναι ένας μετρητής που αποθηκεύει το δείκτη του τελευταίου στοιχείου της λίστας. Στο παρακάτω Παράδειγμα 3.1α, παρουσιάζεται μια τέτοια υλοποίηση στην Pascal και στο Παράδειγμα 3.1β στη C.

[2] Η οριζόντια απεικόνιση ενός πίνακα, όπως αυτή στο Σχήμα 3.1, είναι εναλλακτική της κατακόρυφης, όπως αυτή στο Σχήμα 2.1β.

Παράδειγμα 3.1α *Ορισμός συνεχόμενης λίστας στην Pascal*

Ο ορισμός της δομής μιας συνεχόμενης λίστας μπορεί να γίνει στην Pascal ως εξής:

```
type
    slista = record
                stoixeia: array [1..n] of typostoixeiou;
                telos: 0..n
             end;
```

Ορίζεται, δηλαδή, ένας νέος τύπος δεδομένων (type) με όνομα slista που είναι μια εγγραφή (record) με δύο πεδία. Το πρώτο πεδίο (stoixeia) είναι ένας πίνακας (array) με δείκτες από το σύνολο ακεραίων {1, 2, …, *n*}, δηλ. μεγέθους *n*, και στοιχεία κάποιου τύπου, έστω 'typostoixeiou'. Το δεύτερο πεδίο (telos) είναι μια ακέραια μεταβλητή που παίρνει τιμές από το σύνολο ακεραίων {0, 1, 2, …, *n*}.

Στο πεδίο 'stoixeia' αποθηκεύονται τα στοιχεία της λίστας. Το πεδίο 'telos' παίζει το ρόλο του δείκτη του τέλους της λίστας και ενημερώνεται κάθε φορά που γίνεται κάποια αλλαγή στο μέγεθος της λίστας, λόγω εισαγωγής ή διαγραφής κάποιου στοιχείου της. Επομένως, το πεδίο αυτό μας δείχνει το μέγεθος της λίστας. Η τιμή του πεδίου 'telos' δεν πρέπει να υπερβεί την τιμή '*n*', διότι τότε έχουμε υπερχείλιση. Επίσης, παίρνει την τιμή '0' στην περίπτωση κενής λίστας.

Παράδειγμα 3.1β *Ορισμός συνεχόμενης λίστας στη C*

Ο ορισμός της δομής μιας συνεχόμενης λίστας μπορεί να γίνει στην C ως εξής:

```
typedef
    struct {
        typostoixeiou stoixeia [n];
        int telos;
      } slista;
```

Ορίζεται, δηλαδή, ένας νέος τύπος δεδομένων (typedef) με όνομα slista που είναι μια δομή/εγγραφή (struct) με δύο μέλη/πεδία. Το πρώτο πεδίο (stoixeia) είναι ένας πίνακας μεγέθους *n*, και στοιχεία κάποιου τύπου, έστω 'typostoixeiou'. Το δεύτερο πεδίο (telos) είναι μια ακέραια (int) μεταβλητή (στη C δεν υπάρχει τύπος αντίστοιχος της υποπεριοχής ακεραίων ([1..*n*]) της Pascal). Για τα υπόλοιπα ισχύουν παρόμοια, όπως και στο Παράδειγμα 3.1α.

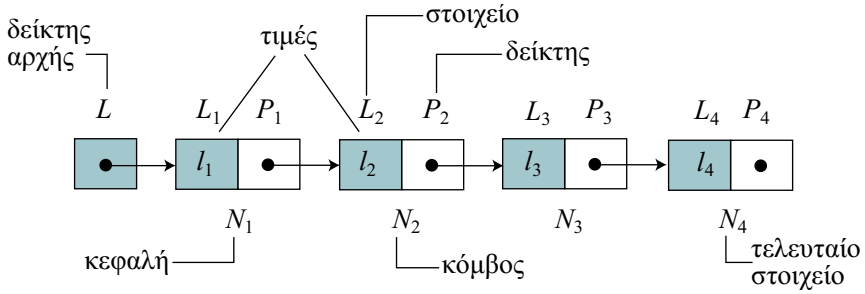
3.1.3 Αναπαράσταση συνδεδεμένης λίστας

Για την αναπαράσταση μιας συνδεδεμένης λίστας μαζί με την τιμή κάθε στοιχείου αποθηκεύεται και ένας *σύνδεσμος* (*link*) ή *δείκτης*[3] (*pointer*) που «δείχνει» στην (πρώτη) θέση μνήμης του επόμενου στοιχείου της λίστας. Επειδή λοιπόν, τα «στοιχεία» μιας συνδεδεμένης λίστας είναι πιο σύνθετα από αυτά μιας συνεχόμενης λίστας, αποκαλούνται *κόμβοι* (*nodes*). Ένας κόμβος αποτελείται από δύο τμήματα. Το αριστερό τμήμα περιέχει την πληροφορία (τιμή) του κόμβου και συνιστά το *στοιχείο* του κόμβου. Το δεξιό τμήμα περιέχει τη διεύθυνση της (πρώτης) θέσης μνήμης του επόμενου κόμβου, είναι δηλαδή ένας *δείκτης* στον επόμενο κόμβο της λίστας. Μια λίστα με κόμβους τέτοιας δομής ονομάζεται *απλά συνδεμένη λίστα* (*one-way or singly linked list*). Υπάρχουν και άλλοι τύποι συνδεδεμένης λίστας, που παρουσιάζονται στην τελευταία ενότητα του κεφαλαίου αυτού.

Στο Σχήμα 3.2 παρουσιάζεται η γραφική απεικόνιση μιας απλά συνδεδεμένης λίστας τεσσάρων στοιχείων $L = (l_1, l_2, l_3, l_4)$, που αποτελείται δηλαδή από τέσσερις κόμβους (N_1, N_2, N_3, N_4). Κάθε κόμβος N_i παριστάνεται από ένα επίμηκες ορθογώνιο παραλληλόγραμμο χωρισμένο σε δύο τμήματα. Το αριστερό τμήμα παριστάνει το αντίστοιχο στοιχείο L_i της λίστας και περιέχει την αντίστοιχη τιμή l_i. Το δεξιό τμήμα παριστάνει το δείκτη P_i του κόμβου που περιέχει τη διεύθυνση του επόμενου κόμβου της λίστας. Το γεγονός ότι ο δείκτης αυτός «δείχνει» στον επόμενο κόμβο απεικονίζεται με ένα ειδικό βέλος ($\bullet\longrightarrow$). Το τελευταίο στοιχείο της λίστας (κόμβος N_4) δεν περιέχει βέλος, αλλά μόνο την έντονη τελεία του βέλους, διότι εφόσον είναι ο τελευταίος κόμβος της λίστας δε «δείχνει» πουθενά. Η τελεία αυτή συμβολίζει τον *κενό δείκτη* (*null pointer*). Ο κενός δείκτης θωρείται ότι έχει σαν τιμή την

[3] Προσοχή να μη γίνεται σύγχυση μεταξύ αυτών των 'δεικτών' (pointes) και των 'δεικτών' (indexes) ενός πίνακα.

ειδική τιμή 'NIL'. Η μεταβλητή δείκτη L ονομάζεται *δείκτης αρχής*, «δείχνει» στην κεφαλή της λίστας (κόμβος N_1) και ουσιαστικά, αντιπροσωπεύει τη λίστα. Σε κενή λίστα, ο δείκτης αυτός έχει τιμή NIL.



Σχήμα 3.2

Σχηματική παράσταση απλά συνδεδεμένης λίστας

Η υλοποίηση μιας απλά συνδεδεμένης λίστας μπορεί να γίνει κατά δύο τρόπους. Ο ένας τρόπος είναι με τη χρήση πινάκων. Συνήθως, χρησιμοποιούνται δύο πίνακες, εκ των οποίων ο ένας περιέχει τα στοιχεία των κόμβων και ο δεύτερος τους δείκτες των κόμβων. Δηλαδή, τα τμήματα κάθε κόμβου (στοιχείο, δείκτης) αντιστοιχούν σε ομοθέσια στοιχεία των δύο πινάκων. Ο τρόπος αυτός χρησιμοποιείται για την υλοποίηση συνδεδεμένων λιστών στις παλαιότερες γλώσσες υψηλού επιπέδου, όπως η FORTRAN, η BASIC και η COBOL, που δε διαθέτουν μηχανισμό δημιουργίας δεικτών (συνδέσμων).

Ο δεύτερος τρόπος χρησιμοποιεί *μεταβλητές δείκτη* (*pointer variables*)[4]. Ο *δείκτης* (*pointer*) είναι ένας τύπος δεδομένων που έχει πεδίο τιμών το σύνολο των διευθύνσεων της μνήμης του Η/Υ. Μια μεταβλητή δείκτη παίρνει σαν τιμές διευθύνσεις θέσεων μνήμης που αντιστοιχούν σε κόμβους, δηλαδή σε σύνθετες τιμές. Έτσι, για κάθε κόμβο χρησιμοποιείται και μια μεταβλητή δείκτη (ή ένας δείκτης).

Ο τρόπος αυτός χρησιμοποιείται για υλοποίηση με γλώσσες υψηλού επιπέδου που παρέχουν κάποιο μηχανισμό δημιουργίας δεικτών, όπως η Pascal και η C. Στο κεφάλαιο αυτό, ασχολούμαστε με αυτό το δεύτερο τρόπο υλοποίησης συνδεδεμένων λιστών. Συνήθως, σε μια τέτοια γλώσσα, ο κάθε κόμβος ορίζεται σαν μια εγγραφή με δύο πεδία, το ένα περιέχει το στοιχείο και το άλλο τον δείκτη.

---

[4] Προσοχή να μη γίνεται σύγχυση μεταξύ *μεταβλητών με δείκτη* (*indexed variables*), που αναφέρονται σε πίνακες, και *μεταβλητών δείκτη* (*pointer variables*), που αναφέρονται σε συνδεδεμένες λίστες.

Παράδειγμα 3.2α

Ορισμός απλά συνδεδεμένης λίστας στην Pascal

Στην Pascal, ο ορισμός μιας απλά συνδεδεμένης λίστας ως τύπου δεδομένων, μπορεί να γίνει ως εξής:

```
type
   deiktiskombou = ^kombos;
   kombos = record
                stoixeio: typostoixeiou;
                deiktis: deiktiskombou
            end;
var
   L: deiktiskombou;
```

Το σύμβολο '^' (ή '↑') χρησιμοποιείται στην Pascal για τη δήλωση δεικτών ακολουθούμενο από τον τύπο δεδομένων για στιγμιότυπα του οποίου θέλουμε να δημιουργήσουμε δείκτες/μεταβλητές δείκτη. Στις παραπάνω δηλώσεις, ορίζεται ένας τύπος δείκτη με το όνομα 'deiktiskombou'. Μεταβλητές του τύπου αυτού «δείχνουν» σε δεδομένα τύπου 'kombos', δηλαδή του τύπου που ακολουθεί το σύμβολο '^' στην πρώτη από τις παραπάνω δηλώσεις. Στη δεύτερη δήλωση, δηλώνεται ο τύπος δεδομένων 'kombos' σαν μια εγγραφή (record) με δύο πεδία. Το πρώτο, 'stoixeio', αντιπροσωπεύει το στοιχείο ενός κόμβου και είναι κάποιου τύπου, έστω 'typostoixeiou'. Το δεύτερο πεδίο, 'deiktis', είναι τύπου 'deiktiskombou', δηλαδή ένας δείκτης σε ένα άλλο κόμβο. Τέλος, δηλώνεται μια μεταβλητή L τύπου 'deiktiskombou', που αντιπροσωπεύει τον δείκτη αρχής της λίστας. Με βάση το δείκτη αυτό, μπορούμε να προσπελάσουμε όλους τους κόμβους μιας συνδεδεμένης λίστας.

Παράδειγμα 3.2β *Ορισμός απλά συνδεδεμένης λίστας στη C*

Στην C ο ορισμός μιας απλά συνδεδεμένης λίστας μπορεί να γίνει ως εξής:

```
typedef struct kombos *Deiktiskombou;

typedef struct kombos {
        typostoixeiou stoixeio;
        deiktiskombou deiktis;
          } KOMBOS;

Deiktiskombou L;
```

Εδώ, το '*' παίζει ένα ρόλο αντίστοιχο του '^' της Pascal. Έτσι, ορίζεται ένας τύπος δείκτη με το όνομα 'Deiktiskombou'. Μεταβλητές του τύπου αυτού, όπως η L, «δείχνουν» σε δεδομένα τύπου 'KOMBOS', δηλαδή σε μια εγγραφή που περιγράφεται από τη δομή (struct) 'kombos'. Τα υπόλοιπα, ισχύουν όπως και στο Παράδειγμα 3.2α.

Δραστηριότητα 3.1

Μελετήστε το τμήμα της ενότητας «8.2 Lists» με τίτλο «Comparing Sequential and Linked List Representation» του βιβλίου του Thomas A. Standish, *"Data Structures, Algorithms and Software Principles"*, Addison-Wesley, 1994, και την υποενότητα «4.1.3 Comparison of List Implementations» του βιβλίου του Clifford A. Shaffer, *"A Practical Introduction to Data Structures and Algorithm Analysis"*, Prentice Hall, 1997 και συνοψίστε τα κυριότερα σημεία τους σε μιαδύο σελίδες. Θα ήταν χρήσιμο να επανέλθετε στα τμήματα αυτά και μετά τη μελέτη των επόμενων δύο ενοτήτων. Τη δική μας απάντηση-σύνοψη θα τη βρείτε στο τέλος του βιβλίου.

3.2 Πράξεις σε συνεχόμενη λίστα

3.2.1 Διαπέραση

Η *διαπέραση* (*traversal*) είναι μια από τις συνήθεις πράξεις στις δομές δεδομένων. Διαπέραση σε μια λίστα (ή έναν πίνακα) σημαίνει να «περάσουμε» από όλα τα στοιχεία της, το ένα μετά το άλλο, και να εφαρμόσουμε κάποιο είδος επεξεργασίας σ' αυτά. Αυτό, συχνά ονομάζεται και *επίσκεψη* (*visiting*). Το είδος της επεξεργασίας εξαρτάται από το συγκεκριμένο πρόβλημα. Π.χ., αύξηση της τιμής των στοιχείων κατά ένα.

ΑΛΓΟΡΙΘΜΟΣ 3.1: ΔΙΑΠΕΡΑΣΗ ΣΥΝΕΧΟΜΕΝΗΣ ΛΙΣΤΑΣ

Είσοδος: Ένας πίνακας (L) και το τρέχον μέγεθος της αντίστοιχης λίστας (T).

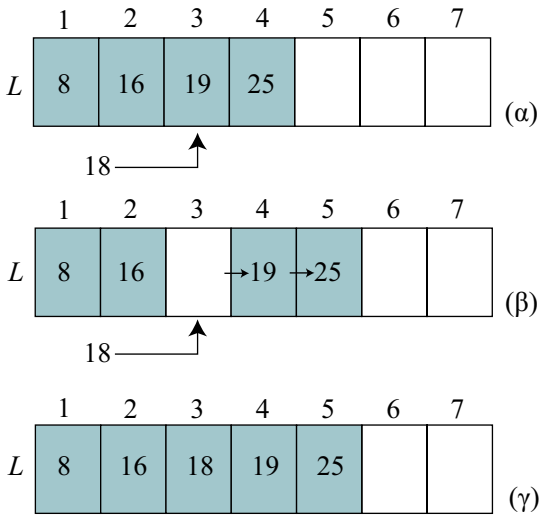
Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται μεταβολές στις τιμές των στοιχείων της λίστας.

```
SL-DIAPERASH (L, T)
1    for K ← 1 to T                {Καθορισμός αριθμού επαναλήψεων}
2      PROCESS (L[K])              {Εφαρμογή διαδικασίας}
     endfor
```

Ο αλγόριθμος για τη διαπέραση μιας συνεχόμενης λίστας παρουσιάζεται στο παραπάνω πλαίσιο, ως Αλγόριθμος 3.1 (SL-DIAPERASH). Την επεξεργασία που υφίσταται κάθε στοιχείο αντιπροσωπεύει η διαδικασία PROCESS, που δέχεται σαν παράμετρο το τρέχον στοιχείο. Η παράμετρος T είναι ο δείκτης τέλους της λίστας, δηλαδή αντιπροσωπεύει το τρέχον μέγεθος της λίστας. Ο Αλγόριθμος 3.1 μπορεί να εφαρμοστεί και για την διαπέραση ενός πίνακα, με τη μόνη διαφορά ότι το T τότε αντιπροσωπεύει το (σταθερό) μέγεθος του πίνακα. Η λογική του αλγορίθμου είναι πολύ απλή. Η διαδικασία PROCESS μπορεί να έχει και άλλες παραμέτρους εκτός του τρέχοντος στοιχείου.

3.2.2 Εισαγωγή

Η *εισαγωγή* (*insertion*) είναι η πράξη με την οποία εισάγουμε ένα νέο στοιχείο (τιμή) σε μια λίστα. Η εισαγωγή ενός στοιχείου στο τέλος μιας λίστας, είναι μια εύκολη διαδικασία, αρκεί το μέγεθος της λίστας να μην υπερβεί το μέγεθος του πίνακα που την αναπαριστά. Η διαδικασία δυσκολεύει όταν θέλουμε να εισάγουμε ένα στοιχείο σε μια ενδιάμεση θέση μεταξύ των στοιχείων της λίστας. Τότε, πρέπει τα στοιχεία που βρίσκονται δεξιά της θέσης εισαγωγής να μετακινηθούν μια θέση δεξιότερα. Στο Σχήμα 3.3 περιγράφονται οι φάσεις της διαδικασίας εισαγωγής ενός στοιχείου σε μια λίστα με τρέχον μέγεθος 4, που παριστάνεται μέσω ενός πίνακα μεγέθους 7.



Σχήμα 3.3
(α) Πριν την εισαγωγή (β) Μετακίνηση στοιχείων (γ) Εισαγωγή στοιχείου

ΑΛΓΟΡΙΘΜΟΣ 3.2: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), το μέγεθός του (N), ο δείκτης τέλους της αντίστοιχης λίστας (T), η θέση εισαγωγής (I) και η προς εισαγωγή τιμή (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα υπερχείλισης στην περίπτωση γεμάτου πίνακα, αλλιώς τίποτα. Εσωτερικά, γίνονται οι απαραίτητες μετακινήσεις, καταχώρηση της τιμής και ενημέρωση του δείκτη τέλους σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
SL-EISAGWGH (L, N, T, I, X)
1    if T=N                          {Έλεγχος γεμάτης λίστας}
2      then print 'ΥΠΕΡΧΕΙΛΙΣΗ'      {Μήνυμα υπερχείλισης}
3      else K ← T                    {Αρχικοποίηση μετρητή}
4           while K ≥ I              {Έλεγχος τέλους επανάληψης}
5              L[K+1] ← L[K]         {Μετακίνηση στοιχείου μια θέση δεξιά}
6              K ← K-1               {Ενημέρωση μετρητή}
           endwhile
7           L[I] ← X                 {Εισαγωγή στοιχείου}
8           T ← T+1                  {Ενημέρωση δείκτη τέλους}
     endif
```

Ο Αλγόριθμος 3.2 (SL-EISAGWGH) είναι αυτός που υλοποιεί την εισαγωγή ενός στοιχείου σε μια συνεχόμενη λίστα. Η λογική του Αλγορίθμου 3.2 έχει ως εξής: Μετά από τον έλεγχο για το αν ο πίνα-

κας είναι γεμάτος, οπότε επιστρέφει μήνυμα υπερχείλισης (βήματα 1-2), αρχίζει ο κυρίως αλγόριθμος. Γίνεται μετακίνηση κατά μια θέση δεξιά των στοιχείων της λίστας που βρίσκονται στη θέση I και δεξιά της, αρχίζοντας από το τέλος της λίστας (θέση T) και προχωρώντας προς τα αριστερά (μέχρι και τη θέση I). Αυτό επιτυγχάνεται με τη διάταξη while (βήματα 4-6) με κατάλληλη αρχικοποίηση του μετρητή K (βήμα 3). Στη συνέχεια, γίνεται καταχώρηση του στοιχείου X στην κενή θέση I του πίνακα (βήμα 7) και αυξάνεται ο δείκτης τέλους της λίστας κατά ένα (βήμα 8), αφού το μέγεθος της λίστας αυξήθηκε κατά ένα, λόγω της εισαγωγής του στοιχείου.

3.2.3 Διαγραφή

Η *διαγραφή* (*deletion*) ενός στοιχείου από μια λίστα πραγματοποιείται με τη μετακίνηση των στοιχείων της λίστας που είναι στα δεξιά του, κατά μια θέση αριστερά. Ο αντίστοιχος αλγόριθμος είναι ο Αλγόριθμος 3.3 (SL-DIAGRAFH).

ΑΛΓΟΡΙΘΜΟΣ 3.3: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), ο δείκτης τέλους της αντίστοιχης λίστας (T) και η θέση τού προς διαγραφή στοιχείου (I).

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται οι απαραίτητες μετακινήσεις και ενημερώνεται ο δείκτης τέλους της λίστας.

```
SL-DIAGRAFH (L, T, I)
1   for K = I to T-1                    {Καθορισμός ορίων επανάληψης}
2       L[K] ← L[K+1]                   {Μετακίνηση στοιχείου μια θέση αριστερά}
    endfor
3   T ← T-1                             {Ενημέρωση δείκτη τέλους της λίστας}
```

Η λογική του αλγορίθμου αυτού έχει ως εξής: Γίνεται μετακίνηση των στοιχείων που βρίσκονται στα δεξιά της θέσης I του προς διαγραφή στοιχείου κατά μια θέση αριστερά, αρχίζοντας από το πρώτο δεξιά στοιχείο, με τη διάταξη for (βήματα 1-2). Στη συνέχεια, ελαττώνεται ο δείκτης τέλους της λίστας T κατά ένα (βήμα 3), αφού το μέγεθος της λίστας ελαττώθηκε.

3.2.4 Αναζήτηση

Οι μέθοδοι γραμμικής και δυαδικής αναζήτησης σε πίνακα εφαρμόζονται με τον ίδιο ακριβώς τρόπο και σε συνεχόμενη λίστα. Οι αντίστοιχοι αλγόριθμοι (2.1 και 2.2) παραμένουν οι ίδιοι με τις εξής μετονομασίες παραμέτρων μόνο: L αντί A και T αντί N.

Παράδειγμα 3.3. *Αλγόριθμος Αναζήτησης και Διαγραφής Στοιχείου*

Θεωρούμε μια συνεχόμενη λίστα και μια τιμή. Το ζητούμενο είναι να σχεδιάσουμε ένα αλγόριθμο που να βρίσκει αν υπάρχει στοιχείο με αυτή την τιμή στη λίστα και, αν υπάρχει, να το διαγράφει.

ΑΛΓΟΡΙΘΜΟΣ Π3.3: ΑΝΑΖΗΤΗΣΗ - ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), ο δείκτης τέλους της αντίστοιχης λίστας (T) και η τιμή τού προς διαγραφή στοιχείου (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους στην περίπτωση που το X δεν υπάρχει, αλλιώς τίποτα. Εσωτερικά, διαγράφεται το στοιχείο, γίνονται οι απαραίτητες μετακινήσεις και ενημερώνεται ο δείκτης τέλους σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
SL-ANAZHTHSH-DIAGRAFH(L, T, X)
1    I ← 1                                  {Αρχικοποίηση μεταβλητών}
2    while (I ≤ T)  and (L[I] ≠ X)          {Έλεγχος τέλους επανάληψης}
3      I ← I + 1                            {Ενημέρωση δείκτη θέσης}
     endwhile
4    if L[I] = X                            {Αναγνώριση αποτελέσματος}
5      then for K = I to T-1                {Καθορισμός ορίων επανάληψης}
6             L[K] ← L[K+1]                 {Μετακίνηση μια θέση αριστερά}
           endfor
7          T ← T - 1                        {Ενημέρωση δείκτη τέλους }
       else print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'       {Μήνυμα λάθους}
8    endif
```

Ο ζητούμενος αλγόριθμος, που είναι ένας συνδυασμός των αλγορίθμων αναζήτησης (Αλγόριθμος 2.1) και διαγραφής (Αλγόριθμος 3.3), παρουσιάζεται στο παραπάνω πλαίσιο ως Αλγόριθμος Π3.3. Η λογική του αλγορίθμου είναι η ακόλουθη: Κατ' αρχήν, αναζητούμε το στοιχείο με τη δοθείσα τιμή (βήματα 1-3) και, όταν το βρούμε (βήμα 4), το διαγράφουμε (βήματα 5-7), αλλιώς επιστρέφουμε ένα μήνυμα αποτυχίας (βήμα 8).

Άσκηση Αυτοαξιολόγησης 3.1

Δίνεται μια τιμή (στοιχείο) και μια διατεταγμένη συνεχόμενη λίστα. Ζητείται να σχεδιαστεί αλγόριθμος που να εισάγει την τιμή στη σωστή θέση μέσα στη λίστα.

Άσκηση Αυτοαξιολόγησης 3.2

Δίνεται μια τιμή (στοιχείο) και μια συνεχόμενη λίστα. Ζητείται να σχεδιαστεί αλγόριθμος που να διαγράφει όλα τα στοιχεία της λίστας που έχουν τιμή ίση με τη δοθείσα.

3.3 Πράξεις σε απλά συνδεδεμένη λίστα

Κάθε γλώσσα προγραμματισμού που παρέχει δείκτες, διαθέτει και αντίστοιχες διαδικασίες δημιουργίας και απαλοιφής δεικτών. Π.χ., η Pascal διαθέτει τις διαδικασίες 'new(p)' και 'dispose(p)' για τη δημιουργία και απαλοιφή αντίστοιχα, ενός κόμβου στον οποίο «δείχνει» ο δείκτης 'p'. Στους αλγορίθμους που σχετίζονται με τη συνδεδεμένη αναπαράσταση, δε μας απασχολούν τέτοια θέματα. Θεωρούμε ότι οι κόμβοι έχουν ήδη με κάποιο τρόπο παραχθεί και στη διαγραφή τους δεν μας απασχολεί θέμα απελευθέρωσης μνήμης.

Επίσης, χρησιμοποιούμε τη βοηθητική συνάρτηση KOMBOS. Η KOMBOS(P), όπου P δείκτης, επιστρέφει τον κόμβο στον οποίο «δείχνει» ο δείκτης. Ακόμη, τα STOIXEIO(KOMBOS(P)), DEIKTHS (KOMBOS(P)) ενεργούν ως τρόποι (μεταβλητές) προσπέλασης των τμημάτων του κόμβου KOMBOS(P).

Τέλος, πρέπει να διευκρινίσουμε τις διαφορές στις έννοιες των εκφράσεων «κόμβος του δείκτη P_i», «δείκτης του κόμβου N_i» και «δείκτης στον κόμβο N_i», που χρησιμοποιούνται στη συνέχεια. Αναφερόμενοι στο Σχήμα 3.2, ο «κόμβος του P_1», που εκφράζεται ως KOMBOS(P1), είναι ο κόμβος N_2, δηλαδή ο κόμβος στον οποίο «δείχνει» ο δείκτης P_1. Ο «δείκτης του κόμβου N_2», που εκφράζεται ως DEIKTHS(KOMBOS(P1)), είναι ο P_2, δηλαδή ο «δείκτης στον κόμβο N_3».

3.3.1 Διαπέραση

Ο αλγόριθμος για τη διαπέραση μιας απλά συνδεδεμένης λίστας παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 3.4 (DL-

DIAPERASH). Η μεταβλητή L είναι ο δείκτης αρχής της λίστας, δηλαδή περιέχει τη διεύθυνση του πρώτου κόμβου της λίστας, ενώ η μεταβλητή P είναι μια βοηθητική (τοπική) μεταβλητή δείκτη, που αντιπροσωπεύει τη διεύθυνση του τρέχοντος κόμβου. Επίσης, χρησιμοποιείται πάλι η διαδικασία PROCESS.

Η λογική του Αλγορίθμου 3.4 έχει ως ακολούθως: Κατ' αρχήν, αρχικοποιούμε τη βοηθητική (τοπική) μεταβλητή P, δίνοντάς της σαν τιμή τη διεύθυνση του πρώτου κόμβου της λίστας. Στη συνέχεια, ξεκινά μια επαναληπτική διαδικασία που έχει δύο βήματα στο σώμα της. Στο πρώτο (βήμα 3), εφαρμόζεται η διαδικασία PROCESS στο στοιχείο του τρέχοντος κόμβου. Στο δεύτερο (βήμα 4), ενημερώνεται ο δείκτης P ώστε να δείχνει τον επόμενο κόμβο της λίστας. Η εκτέλεση των βημάτων αυτών τερματίζεται όταν ο δείκτης P πάρει την τιμή NIL, που σημαίνει ότι δεν υπάρχει επόμενος κόμβος.

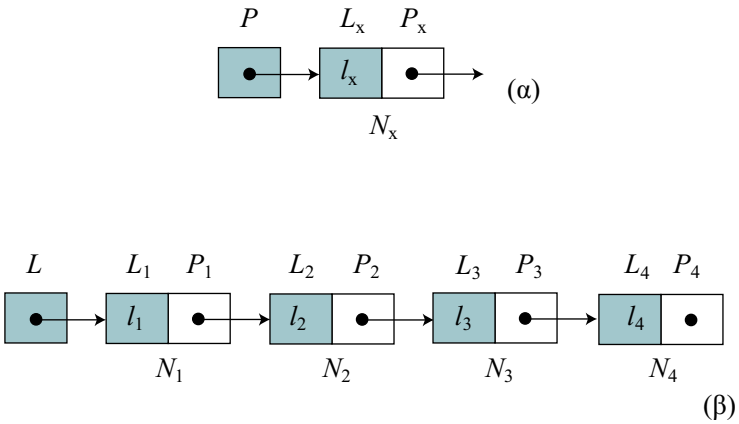
ΑΛΓΟΡΙΘΜΟΣ 3.4: ΔΙΑΠΕΡΑΣΗ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ

Είσοδος: Ο δείκτης αρχής (L) μιας συνδεδεμένης λίστας.
Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται μεταβολές στις τιμές των στοιχείων της λίστας.

```
DL-DIAPERASH (L)
1    P ← L                                  {Αρχικοποίηση δείκτη}
2    while P ≠ NIL                          {Συνθήκη τερματισμού}
3        ROCESS (STOIXEIO(KOMBOS(P)))       {Εφαρμογή διαδικασίας}
4        P ← DEIKTHS(KOMBOS(P))             {Ενημέρωση τρέχοντος δείκτη}
     endwhile
```

3.3.2 Εισαγωγή

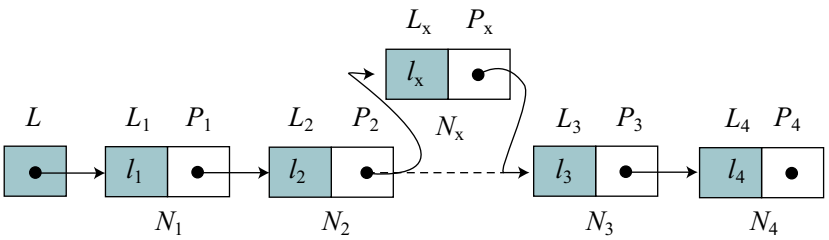
Η εισαγωγή σε μια συνδεδεμένη λίστα σημαίνει την παρεμβολή ενός κόμβου σε κάποια θέση μεταξύ δύο ήδη υπαρχόντων κόμβων της λίστας. Η αρχική κατάσταση της διαδικασίας αυτής απεικονίζεται στο Σχήμα 3.4, ενώ η τελική στο Σχήμα 3.5.



Σχήμα 3.4

Κατάσταση πριν την εισαγωγή κόμβου σε συνδεδεμένη λίστα: (α) προς εισαγωγή κόμβος, (β) αρχική συνδεδεμένη λίστα

Ο αλγόριθμος που πραγματοποιεί την εισαγωγή αυτή, παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 3.5 (DL-EISAGWGH-META). Δέχεται δύο παραμέτρους εισόδου, τον δείκτη στον προς εισαγωγή κόμβο (P) και τον δείκτη στον κόμβο μετά τον οποίο θα γίνει η εισαγωγή (PI). Μετά τον έλεγχο για την ύπαρξη των δύο δεικτών και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), αρχίζει ο κυρίως αλγόριθμος που αποτελείται από δύο βήματα. Στο βήμα 3, ο δείκτης του (προς εισαγωγή) κόμβου P γίνεται δείκτης στον κόμβο που είναι μετά τον κόμβο PI. Αναφερόμενοι στο Σχήμα 3.4, για το οποίο P = P και PI = P_1, ο δείκτης (του κόμβου N_x) P_x παίρνει την τιμή του P_2, δηλαδή δείχνει στον κόμβο N_3. Στο βήμα 4, ο δείκτης του κόμβου PI παίρνει την τιμή του δείκτη στον προς εισαγωγή κόμβο. Σε αναφορά πάλι με το Σχήμα 3.4, ο δείκτης P_2 παίρνει την τιμή του P.



Σχήμα 3.5

Κατάσταση μετά την εισαγωγή κόμβου σε συνδεδεμένη λίστα

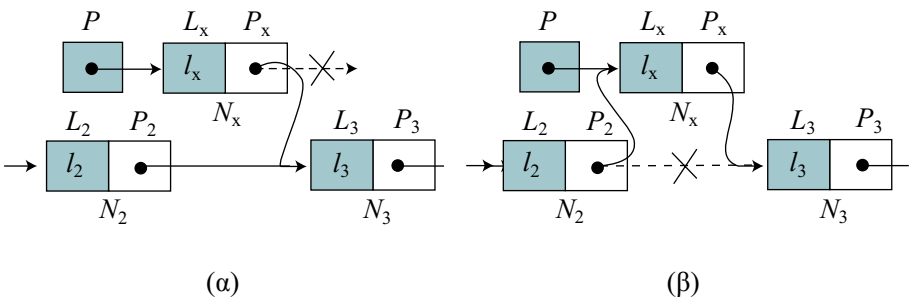
Τα βήματα 3 και 4 του αλγορίθμου παριστάνονται διαγραμματικά, για τα δεδομένα των Σχημάτων 3.4 και 3.5, στο Σχήμα 3.6.

ΑΛΓΟΡΙΘΜΟΣ 3.5: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ο δείκτης στον προς εισαγωγή κόμβο (P) και ο δείκτης στον κόμβο της λίστας μετά τον οποίο θα γίνει η εισαγωγή του νέου κόμβου (PI).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους στην περίπτωση μη έγκυρων δεικτών, αλλιώς τίποτα. Εσωτερικά, γίνονται μεταβολές στη λίστα σε περίπτωση επιτυχούς εισαγωγής, αλλιώς τίποτα.

```
DL-EISAGWGH-META (P, PI)
1    if (P = NIL) or (PI = NIL)                          {Έλεγχος δεικτών}
2       then print 'ΑΝΥΠΑΡΚΤΟΣ ΚΟΜΒΟΣ'                   {Μήνυμα λάθους}
3       else DEIKTHS(KOMBOS(P)) ← DEIKTHS(KOMBOS(PI))   {Ενημέρωση δείκτη}
4            DEIKTHS(KOMBOS(PI)) ← P                    {Ενημέρωση δείκτη}
    endif
```



Σχήμα 3.6
Διαγραμματική αναπαράσταση (α) βήμα 3 (β) βήμα 4 του Αλγορίθμου 3.5

3.3.3 Διαγραφή

Η διαγραφή ενός κόμβου από μια συνδεδεμένη λίστα απεικονίζεται διαγραμματικά στο Σχήμα 3.7. Ο αντίστοιχος αλγόριθμος παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 3.6 (DL-DIAGRAFH-META). Δέχεται μια παράμετρο (PI) που αντιπροσωπεύει την τιμή του δείκτη που «δείχνει» στον προηγούμενο, του προς διαγραφή, κόμβο (π.χ. την τιμή του P_1 στο Σχήμα 3.7, που δείχνει στον N_2). Η παράμετρος αυτή, δεν αντιπροσωπεύει τον δείκτη που «δείχνει» στον προς διαγραφή αλλά στον προηγούμενο κόμβο, διότι, όπως φαίνεται από το Σχήμα 3.7, απαιτείται ο δείκτης του προηγούμενου κόμβου να αλλάξει τιμή και να «δείχνει» στον επόμενο, του προς διαγραφή, κόμβο. Αν μας δίνονταν μόνο η τιμή του δείκτη στον προς διαγραφή κόμβο, δε θα ήταν δυνατό να προσπελάσουμε το δείκτη του προηγούμενου κόμβου, ενώ το αντίστροφο είναι δυνατό.

ΑΛΓΟΡΙΘΜΟΣ 3.6: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

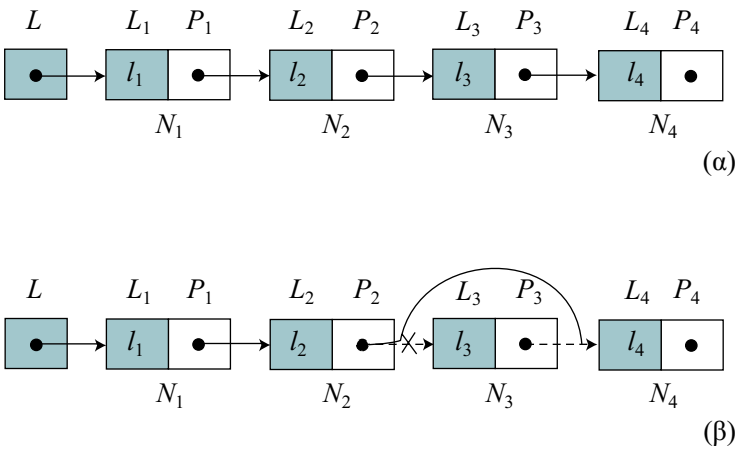
Είσοδος: Ο δείκτης (PI) στον προηγούμενο, του προς διαγραφή, κόμβο.

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους στην περίπτωση μη έγκυρων δεικτών, αλλιώς τίποτα. Εσωτερικά, γίνονται μεταβολές στη λίστα σε περίπτωση επιτυχούς διαγραφής, αλλιώς τίποτα.

```
DL-DIAGRAFH(PI)
1   if (PI=NIL) or (DEIKTHS(KOMBOS(PI))=NIL)        {Έλεγχος δεικτών}
2     then print 'ΑΝΥΠΑΡΚΤΟΣ ΚΟΜΒΟΣ'                  {Μήνυμα λάθους}
3     else P ← DEIKTHS(KOMBOS(PI))                  {Ενημέρωση δεικτών}
4          DEIKTHS(KOMBOS(PI)) ← DEIKTHS(KOMBOS(P))
    endif
```

Στο βήμα 1 γίνεται έλεγχος ύπαρξης του προηγούμενου (PI = NIL) και του προς διαγραφή (DEIKTHS(KOMBOS(PI)) = NIL) κόμβου. Αν κάποιος από αυτούς δεν υπάρχει, τότε επιστρέφει το αντίστοιχο μήνυμα. Αλλιώς, προχωρά στον κυρίως αλγόριθμο που είναι τα βήματα 3 και 4. Στο βήμα 3 κάνουμε την τιμή μιας βοηθητικής μεταβλητής δείκτη P ίση με την τιμή του δείκτη στον υπό διαγραφή κόμβο (ή, με άλλα λόγια, του δείκτη του προηγουμένου τού υπό διαγραφή κόμβου), δηλαδή αποθηκεύουμε την τρέχουσα τιμή του «δείκτη στον υπό διαγραφή κόμβο». Σε αναφορά με το Σχήμα 3.7, η P παίρνει την τιμή του P_2. Στο βήμα 4 ενημερώνουμε το δείκτη του προηγουμένου τού υπό διαγραφή κόμβου (δηλαδή του κόμβου N_2 στο Σχήμα 3.7) με την τιμή του δείκτη του προς διαγραφή κόμβου (ή με άλλα λόγια του δείκτη στον επόμενο τού προς διαγραφή κόμβου) ώστε να «δείχνει» στον επόμενο τού προς διαγραφή κόμβου. Δηλαδή, ο P_2 παίρνει την τιμή του P_3, ώστε να «δείχνει» στον N_4.

Μια ειδική περίπτωση διαγραφής, είναι εκείνη όπου θέλουμε να διαγράψουμε την κεφαλή μιας λίστας. Στην περίπτωση αυτή, ο Αλγόριθμος 3.6 μετατρέπεται ως εξής: α) τη θέση του PI παίρνει ο L, β) ο έλεγχος στο βήμα 1 περιλαμβάνει μόνο το L και γ) τα βήματα 3, 4 συνοψίζονται στο L ← DEIKTHS(KOMBOS(L)).



Σχήμα 3.7
(α) Κατάσταση πριν, (β) κατάσταση μετά τη διαγραφή κόμβου

Παράδειγμα 3.4

Διαγραφή Κ-οστού κόμβου συνδεδεμένης λίστας

Δίνεται μια απλά συνδεδεμένη λίστα και ένας ακέραιος αριθμός Κ. Ζητείται να σχεδιαστεί αλγόριθμος που να διαγράφει το Κ-οστό κόμβο της λίστας.

Ενώ σε μια συνεχόμενη λίστα η πρόσβαση σ' ένα στοιχείο που δίνεται η σειρά του είναι εύκολη, λόγω της άμεσης προσπέλασης που προσφέρει η αναπαράσταση μέσω πίνακα, σε μια συνδεδεμένη λίστα δεν είναι. Για να γίνει αυτό, πρέπει να γίνει μια διαπέραση της λίστας κατά την οποία, αντί επεξεργασίας, θα γίνεται καταμέτρηση των κόμβων μέχρι τη θέση Κ. Αυτό γίνεται στο πρώτο τμήμα του Αλγορίθμου Π3.4 (DL-DIAGRAFH-K), που είναι ο ζητούμενος και παρουσιάζεται στο παρακάτω πλαίσιο.

Το πρώτο αυτό τμήμα αποτελείται βασικά από τη διάταξη επανάληψης while (βήματα 2-5). Το βήμα 1 δίνει αρχικές τιμές σε δύο τοπικές μεταβλητές, την PK και τη I. Η πρώτη, αντιπροσωπεύει το δείκτη του τρέχοντος κόμβου και, τελικά, το δείκτη του Κ-οστού κόμβου, ενώ η δεύτερη, τη θέση του κόμβου. Μέσα στη διάταξη while χρησιμοποιείται και η μεταβλητή PI που αντιπροσωπεύει το δείκτη στον προηγούμενο, του τρέχοντος, κόμβο. Θυμηθείτε ότι, η διαγραφή ενός στοιχείου από συνδεδεμένη λίστα απαιτεί και αυτόν το δείκτη (Αλγόριθμος 3.6).

Η επανάληψη του σώματος του while σταματά όταν φθάσουμε είτε στον Κ-οστό κόμβο είτε στο τέλος της λίστας (P = NIL). Στην πρώτη

περίπτωση, γίνεται διαγραφή του κόμβου, ενώ στη δεύτερη, επιστρέφει ένα μήνυμα λάθους. Αυτό υλοποιείται από το δεύτερο τμήμα του αλγορίθμου, που είναι μια διάταξη if (βήματα 6-8).

ΑΛΓΟΡΙΘΜΟΣ Π3.4: ΔΙΑΓΡΑΦΗ Κ–ΟΣΤΟΥ ΚΟΜΒΟΥ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ

Είσοδος: Ο δείκτης αρχής μιας απλά συνδεδεμένης λίστας (L) και ένας ακέραιος αριθμός (K).

Έξοδος: Εξωτερικά, επιστρέφει ένα μήνυμα λάθους σε περίπτωση ανυπαρξίας του K-οστού κόμβου, αλλιώς τίποτα. Εσωτερικά, γίνεται διαγραφή του κόμβου σε περίπτωση επιτυχίας.

```
DL-DIAGRAFH-K(L, K)
1    PK ← L, I ← 1                                          {Αρχικοποίηση μεταβλητών}
2    while (PK≠NIL) and (I≤K)                               {Έλεγχος τέλους επανάληψης}
3        PI ← P                                             {Ενημέρωση δείκτη}
4        PK ← DEIKTHS(KOMBOS(PK))                           {Ενημέρωση δείκτη}
5        I ← I+1                                            {Ενημέρωση μετρητή}
     endwhile
6    if (PK≠NIL)                                            {Έλεγχος αποτελέσματα}
7      then DEIKTHS(KOMBOS(PI)) ← DEIKTHS(KOMBOS(PK))       {Επιτυχία}
8      else print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'                       {Αποτυχία}
     endif
```

Άσκηση Αυτοαξιολόγησης 3.3

Δίνονται μια απλά συνδεδεμένη λίστα και δύο ακέραιοι αριθμοί K και M. Να σχεδιαστεί αλγόριθμος που να εναλλάσσει τις τιμές των στοιχείων του K-οστού και του M-οστού κόμβου.

3.3.4 Αναζήτηση

Για την αναζήτηση ενός στοιχείου σε μια συνδεδεμένη λίστα μπορεί να εφαρμοστεί μόνο η μέθοδος της γραμμικής αναζήτησης. Η δυαδική αναζήτηση δεν μπορεί να εφαρμοστεί, διότι δεν υπάρχει απ' ευθείας τρόπος να υπολογιστεί ο δείκτης του 'μεσαίου' στοιχείου. Επομένως, είτε η λίστα είναι διατεταγμένη είτε όχι, χρησιμοποιούμε τη γραμμική αναζήτηση, ψάχνουμε δηλαδή τα στοιχεία της λίστας ένα-ένα. Βέβαια, όταν η λίστα είναι διατεταγμένη, έχουμε διαφορετική συνθήκη τερματισμού του αλγορίθμου.

Παράδειγμα 3.5 *Εύρεση Μηδενικών Στοιχείων σε Απλά Συνδεδεμένη Λίστα*

Θεωρούμε μια απλά συνδεδεμένη λίστα στην οποία αποθηκεύουμε αριθμούς. Το ζητούμενο είναι να σχεδιάσουμε έναν αλγόριθμο που να μετρά τον αριθμό των στοιχείων με μηδενική τιμή.

Ο ζητούμενος αλγόριθμος παρουσιάζεται στο παρακάτω πλαίσιο ως Αλγόριθμος Π3.5 (DL-EURESH-MHDEN). Θα μπορούσε να θεωρηθεί σαν μια τροποποίηση του αλγορίθμου διαπέρασης (Αλγόριθμος 3.4). Η βασική ιδέα είναι ότι περνάμε από όλους τους κόμβους της λίστας, μέσω μιας επαναληπτικής διάταξης while (βήματα 2-4), και σε κάθε κόμβο εξετάζουμε αν η τιμή του στοιχείου του είναι μηδενική (βήμα 3). Αν είναι, καταγράφουμε τη διαπίστωση με την αύξηση της τιμής του μετρητή I (βήμα 4). Αν όχι, προχωρούμε στον επόμενο κόμβο (βήμα 5). Στο τέλος, επιστρέφουμε την τιμή του μετρητή που αντιπροσωπεύει τον αριθμό των μηδενικών στοιχείων της λίστας (βήμα 6).

ΑΛΓΟΡΙΘΜΟΣ Π3.5: ΕΥΡΕΣΗ ΜΗΔΕΝΙΚΩΝ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ο δείκτης αρχής (L) μιας λίστας.
Έξοδος: Εξωτερικά, επιστρέφει τον αριθμό των μηδενικών στοιχείων της λίστας. Εσωτερικά, δεν γίνεται τίποτα.

```
DL-EURESH-MHDEN(L)
1    P ← L, I ← 0                          {Αρχικοποίηση μεταβλητών}
2    while P ≠ NIL                         {Συνθήκη επανάληψης}
3        if STOIXEIO(KOMBOS(P))=0          {Έλεγχος μηδενικού στοιχείου}
4            then I ← I+1                  {Ενημέρωση μετρητή}
         endif
5        P ← DEIKTHS(KOMBOS(P))            {Ενημέρωση τρέχοντος δείκτη}
     endwhile
6    print I                               {Επιστροφή τιμής μετρητή}
```

Άσκηση Αυτοαξιολόγησης 3.4

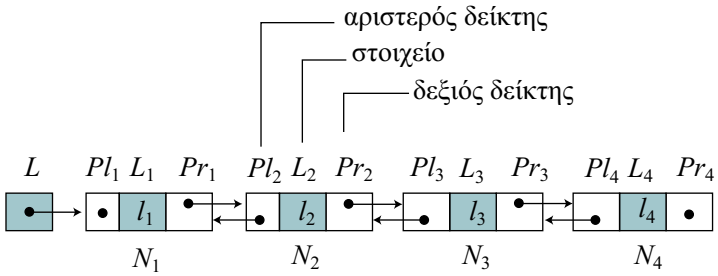
Να σχεδιάσετε ένα αλγόριθμο που να διαγράφει ένα στοιχείο μιας απλά συνδεδεμένης λίστας, όταν δίνεται η τιμή τού προς διαγραφή στοιχείου και όχι η τιμή του δείκτη στο προηγούμενο στοιχείο (όπως στον βασικό αλγόριθμο 3.6).

3.4 Αλλες κατηγορίες συνδεδεμένης λίστας

Εκτός από την απλά συνδεδεμένη λίστα υπάρχουν και άλλοι τύποι συνδεδεμένης λίστας. Οι πιο γνωστοί είναι τρεις, τους οποίους και περιγράφουμε συνοπτικά, στη συνέχεια.

3.4.1 Διπλά συνδεδεμένη λίστα

Στη *διπλά συνδεδεμένη λίστα* (*two-way or doubly linked list*), οι κόμβοι έχουν δύο δείκτες, έναν που «δείχνει» στον επόμενο κόμβο και ονομάζεται *δεξιός δείκτης* (*right pointer*), και έναν που «δείχνει» στον προηγούμενο κόμβο και ονομάζεται *αριστερός δείκτης* (*left pointer*). Στο Σχήμα 3.8, απεικονίζεται η σχηματική παράσταση μιας διπλά συνδεδεμένης λίστας. Όπως παρατηρείτε, ο κάθε κόμβος αποτελείται από τρία τμήματα, δύο ακραία για την αποθήκευση των δύο δεικτών και το μεσαίο για την αποθήκευση του στοιχείου (πληροφορίας) του κόμβου.



Σχήμα 3.8

Σχηματική παράσταση διπλά συνδεδεμένης λίστας

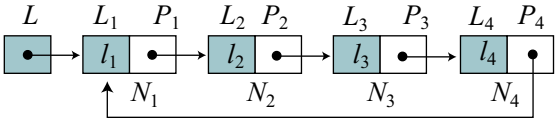
Δοθέντος ενός δείκτη P μπορούμε να προσπελάσουμε όλους τους υπόλοιπους κόμβους της λίστας, κινούμενοι είτε προς τη μια είτε προς την άλλη κατεύθυνση, πράγμα που δεν μπορεί να γίνει σε μια απλά συνδεδεμένη λίστα. Γι΄ αυτό, μπορούμε να εισάγουμε και να διαγράψουμε έναν κόμβο είτε πριν είτε μετά από τον δοθέντα.

3.4.2 Κυκλική συνδεδεμένη λίστα

Σε μια *κυκλική συνδεδεμένη λίστα* (*circular linked list*) ο δείκτης του τελευταίου κόμβου «δείχνει» πίσω στον πρώτο κόμβο της λίστας,

όπως απεικονίζεται στο Σχήμα 3.9. Η κυκλική λίστα έχει το πλεονέκτημα, όπως και η διπλά συνδεδεμένη, ότι μπορούμε να προσπελάσουμε από έναν κόμβο οποιοδήποτε άλλο κόμβο.

Σχήμα 3.9
Σχηματική παράσταση κυκλικής συνδεδεμένης λίστας



3.4.3 Συνδεδεμένη λίστα με κεφαλίδα

Μια *συνδεδεμένη λίστα με κεφαλίδα* (*header linked list*) περιλαμβάνει ένα ειδικό κόμβο, που ονομάζεται *κεφαλίδα* (*header*), στην αρχή της λίστας. Ο κόμβος αυτός μπορεί να είναι είτε ένας κανονικός κόμβος, όπως οι υπόλοιποι, είτε ένας δείκτης. Μια συνήθης χρήση της κεφαλίδας είναι σε μια κυκλική λίστα, όπου χρησιμοποιείται σαν δείκτης του τέλους της λίστας. Στην περίπτωση αυτή, η λίστα ονομάζεται *κυκλική συνδεδεμένη λίστα με κεφαλίδα* (*circular header linked list*).

Σύνοψη

Η λίστα είναι μια διαφορετική δομή από αυτή του πίνακα, αν και φαίνεται να μοιάζουν. Η λίστα είναι ***δομή σειριακής προσπέλασης***, ενώ ο πίνακας τυχαίας. Η λίστα είναι ***δυναμική δομή***, δηλαδή το μέγεθός της μεταβάλλεται κατά την εκτέλεση του αντίστοιχου προγράμματος, ενώ ο πίνακας στατική. Η λίστα, τέλος, στηρίζεται στην έννοια της ακολουθίας, ενώ ο πίνακας σ' αυτές του συνόλου και της συνάρτησης.

Διακρίνουμε δύο βασικά είδη λίστας ως προς τον τρόπο αναπαράστασής της, τις ***συνεχόμενες λίστες*** και τις ***συνδεδεμένες λίστες***. Οι συνεχόμενες λίστες αποθηκεύουν τα στοιχεία τους σε γειτονικές (συνεχόμενες) θέσεις μνήμης και υλοποιούνται, συνήθως, μέσω πίνακα, ενώ οι συνδεδεμένες λίστες σε, συνήθως, μη γειτονικές θέσεις μνήμης και υλοποιούνται μέσω συνδέσμων ή δεικτών. Τα στοιχεία μιας συνδεδεμένης λίστας είναι πιο σύνθετα από αυτά μιας συνεχόμενης, δεδομένου ότι κρατούν και πληροφορία για τη θέση του επόμενου στοιχείου και, ενδεχομένως, και του προηγούμενου στοιχείου, και ονομάζονται ***κόμβοι***. Υπάρχουν πολλά είδη συνδεδεμένων λιστών. Το πιο διαδεδομένο είναι η απλά συνδεδεμένη λίστα. Άλλα συνήθη

είδη συνδεδεμένης λίστας είναι η ***διπλά συνδεδεμένη λίστα***, η ***κυκλική συνδεδεμένη λίστα*** και η ***συνδεδεμένη λίστα με κεφαλίδα***.

Η δυναμική φύση μιας λίστας συνίσταται στη συχνή διαγραφή και εισαγωγή στοιχείων. Στην υλοποίηση των πράξεων αυτών σε μια συνεχόμενη λίστα σαν κύριο χαρακτηριστικό έχουμε τη μετακίνηση στοιχείων ενός πίνακα, ενώ σε μια συνδεδεμένη λίστα, την αλλαγή τιμών των δεικτών. Εκτός από την εισαγωγή και διαγραφή στοιχείων, η διαπέραση μιας λίστας και η αναζήτηση στοιχείου σε μια λίστα είναι άλλες δύο από τις κυριότερες πράξεις σε μια λίστα.

Οδηγός για περαιτέρω μελέτη

1. Κοίλιας, Χ. , *Δομές Δεδομένων και Οργανώσεις Αρχείων*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1993.

   Στην ενότητα «3.4 Λίστες», το βιβλίο αυτό διαπραγματεύεται δύο ενδιαφέροντα θέματα, όπως η υλοποίηση απλά συνδεδεμένων λιστών με πίνακες και η αποκομιδή άχρηστων δεδομένων (garbage collection).

2. Μανωλόπουλος, Ι. , *Δομές Δεδομένων*, τόμ. Α', ART of TEXT, 2η έκδοση, Θεσσαλονίκη 1992.

   Στην υποενότητα «3.5.1.Κόμβος Φρουρός» του βιβλίου αυτού μπορείτε να βρείτε μια συνοπτική παρουσίαση της τεχνικής του ‘κόμβου φρουρού’ (sentinel node) που χρησιμοποιείται στην πράξη της αναζήτησης σε λίστες.

3. Cormen, H. T., Leiserson E. C. και Rivest, L. R., *Introduction to Algorithms*, McGraw-Hill, Cambridge, MA 1996 (ανατύπωση).

   Στην ενότητα «11.2 Linked Lists» του βιβλίου αυτού υπάρχει μια αναλυτική διαπραγμάτευση των πράξεων και των αντίστοιχων αλγορίθμων σε διπλά συνδεδεμένες λίστες. Επίσης, στην επόμενη ενότητα «11.3 Implementing pointers and objects» εξηγείται πώς γίνεται υλοποίηση της διπλά συνδεδεμένης λίστας με χρήση πινάκων.

4. Main M. και Savitch W., *Data Structures and Other Objects*, Benjamin/Cummings, Redwood City, CA 1995.

Στο τμήμα της ενότητας «5.3 Linked Lists in Pascal» του βιβλίου αυτού με τίτλο *«Searching a linked list»* μπορείτε να βρείτε αλγορίθμους αναζήτησης σε μια απλά συνδεδεμένη λίστα και αντίστοιχες υλοποιήσεις σε Pascal.

5. Standish, A. T. , *Data Structures*, *Algorithm and Software Principles*, Addison-Wesley, Reading, MA 1994.

   Στην ενότητα 8.6 Dynamic Memory Allocation, και ιδιαίτερα στα τμήματά της με τίτλους «Available Space Lists» και «Garbage Collection», το βιβλίο αυτό διαπραγματεύεται αναλυτικά τα θέματα της δυναμικής διάθεσης μνήμης και της αποκομιδής άχρηστων δεδομένων.

6. Weiss, A. M., *Data Structures and Algorithm Analysis*, Benjamin/Cummings, 2nd Edition, Redwood City, CA 1995.

   Στο κεφάλαιο 3 του βιβλίου αυτού υπάρχουν διάφορες παραλλαγές αλγορίθμων για πράξεις σε απλά συνδεδεμένη λίστα σε Pascal. Επίσης, παραδείγματα εφαρμογής της συνδεδεμένης λίστας, όπως η αναπαράσταση πολυωνύμων μιας μεταβλητής.

Ειδικές λίστες

ΚΕΦΑΛΑΙΟ 4

Σκοπός

Στο κεφάλαιο αυτό παρουσιάζονται οι δύο βασικοί τύποι ειδικών λιστών, η 'στοίβα' και η 'ουρά'. Η παρουσίαση κυρίως αφορά τον ορισμό των δομών αυτών, τις πράξεις σ' αυτές και τους αντίστοιχους αλγορίθμους υλοποίησής τους.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό, θα είστε σε θέση να:

- *σχεδιάζετε τροποποιήσεις, επεκτάσεις ή συνδυασμούς των βασικών αλγορίθμων των πράξεων σε συνεχόμενες και συνδεμένες στοίβες,*
- *εξηγείτε τη χρήση της στοίβας στη λύση προβλημάτων,*
- *σχεδιάζετε τροποποιήσεις, επεκτάσεις ή συνδυασμούς των βασικών αλγορίθμων των πράξεων σε συνεχόμενες ή συνδεμένες ουρές,*
- *εξηγείτε τη χρήση της ουράς στη λύση προβλημάτων.*

Έννοιες κλειδιά

- *Ειδική λίστα*
- *Στοίβα*
- *Λογική LIFO*
- *Συνεχόμενη στοίβα*
- *Συνδεδεμένη στοίβα*
- *Κορυφή στοίβας*
- *Βάση στοίβας*
- *Εισαγωγή σε στοίβα*
- *Διαγραφή από στοίβα*
- *Κλήση υποπρογραμμάτων*
- *Εκτίμηση εκφράσεων*
- *Ουρά*

- *Λογική FIFO*
- *Συνεχόμενη ουρά*
- *Συνδεδεμένη ουρά*
- *Εμπρός άκρο*
- *Πίσω άκρο*
- *Εισαγωγή σε ουρά*
- *Διαγραφή από ουρά*
- *Κυκλική ουρά*
- *Διπλή ουρά*
- *Ουρά προτεραιότητας*
- *Λογική HPIFO*

Ενότητες Κεφαλαίου 4

4.1 Στοίβα

4.2 Εφαρμογές στοίβας

4.3 Ουρά

4.4 Ειδικοί τύποι ουράς

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό εξετάζουμε τις ειδικές λίστες, μια άλλη κατηγορία ανώτερων δομών δεδομένων. Στην πρώτη ενότητα παρουσιάζεται η 'στοίβα', ένας τύπος ειδικής λίστας. Δίνεται ο ορισμός της, περιγράφονται οι τρόποι αναπαράστασής της, συνεχόμενη και συνδεδεμένη, και εξηγούνται οι αλγόριθμοι για τις δύο κυριότερες πράξεις σ' αυτήν, την εισαγωγή και τη διαγραφή.

Στη δεύτερη ενότητα παρουσιάζονται δύο εφαρμογές της στοίβας στην επιστήμη των υπολογιστών, η χρήση της στην κλήση υποπρογραμμάτων και στην εκτίμηση αριθμητικών εκφράσεων.

Στην τρίτη ενότητα ασχολούμαστε με τον άλλο τύπο ειδικής λίστας, την 'ουρά'. Δίνεται ο ορισμός της και παρουσιάζονται οι τρόποι αναπαράστασής της, συνεχόμενη και συνδεδεμένη, και οι αλγόριθμοι για την εισαγωγή και διαγραφή στοιχείων/κόμβων.

Τέλος, στην τέταρτη ενότητα παρουσιάζονται συνοπτικά τρεις ειδικοί τύποι ουράς, η 'κυκλική ουρά', η 'διπλή ουρά' και η 'ουρά προτεραιότητας'.

4.1 Στοίβα

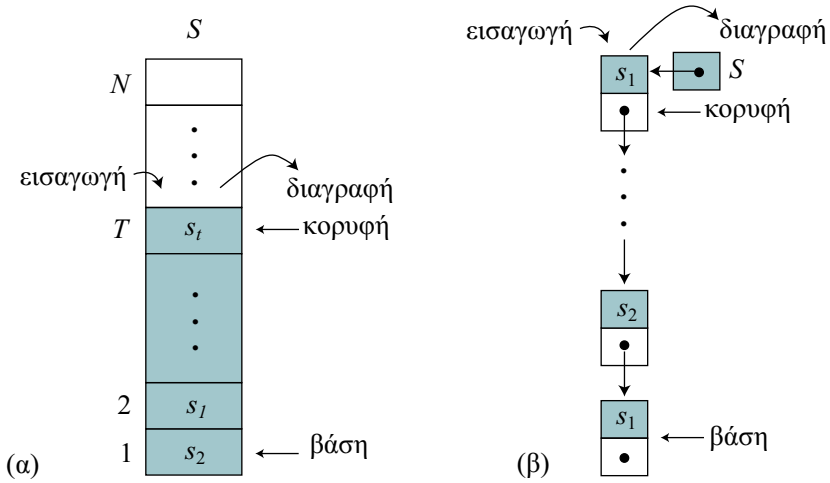
4.1.1 Ορισμός και αναπαράσταση

Μια *στοίβα* (*stack*) είναι ένας ειδικός τύπος λίστας, δηλαδή μια λίστα που έχει ορισμένους περιορισμούς ως προς την εφαρμογή των διαφόρων πράξεων σ' αυτή. Πιο συγκεκριμένα, μια στοίβα επιτρέπει εισαγωγές και διαγραφές στοιχείων/κόμβων μόνο στο ένα άκρο της, που ονομάζεται *κορυφή* (*top*) της στοίβας και χαρακτηρίζεται ως το 'ελεύθερο' άκρο της. Το άλλο άκρο, το μη ελεύθερο, ονομάζεται *βάση* (*bottom*). ■

■ Μια **στοίβα** είναι μια λίστα στην οποία εισαγωγές και διαγραφές στοιχείων ή κόμβων μπορούν να γίνουν μόνο στο ένα άκρο της (κορυφή).

Δηλαδή, σε κάθε στιγμή σε μια στοίβα μπορούμε είτε να εισάγουμε ένα στοιχείο στην κορυφή της είτε να διαγράψουμε (ή εξάγουμε) ένα στοιχείο από την κορυφή της. Στην τελευταία περίπτωση, το στοιχείο που εισήχθη τελευταίο είναι αυτό που διαγράφεται. Μια τέτοιου είδους λογική ονομάζεται λογική LIFO (Last-In First-Out). Από την καθημερινή μας ζωή, ένα σύνηθες παράδειγμα στοίβας, είναι αυτό της στοίβας πιάτων σ' ένα εστιατόριο, όπου τα πιάτα που πλένονται και τοποθετούνται πρώτα, βρίσκονται στον πάτο της στοίβας και αυτό που τοποθετήθηκε τελευταίο είναι αυτό που θα πάρει πρώτο ο σερβιτόρος για σερβίρισμα.

Η αναπαράσταση μιας στοίβας γίνεται είτε με τον τρόπο της συνεχόμενης αναπαράστασης είτε με αυτόν της συνδεδεμένης αναπαράστασης. Έτσι, διακρίνουμε μεταξύ *συνεχόμενης στοίβας* (*contiguous stack*) και *συνδεδεμένης στοίβας* (*linked stack*). Στην πρώτη περίπτωση, η στοίβα υλοποιείται μέσω ενός πίνακα, ενώ στη δεύτερη, μέσω συνδέσμων ή δεικτών. Μια συνεχόμενη στοίβα σε μια γλώσσα υψηλού επιπέδου μπορεί να οριστεί σαν μια εγγραφή με δύο πεδία, που το ένα είναι ένας πίνακας που αποθηκεύονται τα στοιχεία της και το άλλο ένας δείκτης στην κορυφή της. Μια συνδεδεμένη στοίβα μπορεί να οριστεί σαν μια μεταβλητή δείκτη που αποθηκεύει τον δείκτη στην κορυφή της στοίβας. Κάθε κόμβος της στοίβας ορίζεται σαν μια εγγραφή με δύο πεδία, που το ένα περιέχει το στοιχείο και το άλλο τον δείκτη στον επόμενο κόμβο. Στο Σχήμα 4.1 απεικονίζονται οι δύο τρόποι αναπαράστασης μιας στοίβας.



Σχήμα 4.1

(α) Συνεχόμενη αναπαράσταση, (β) συνδεδεμένη αναπαράσταση στοίβας

4.1.2 Πράξεις σε στοίβα

Όπως είναι προφανές, δύο είναι οι κυριότερες πράξεις που μας ενδιαφέρουν σε μια στοίβα, η 'εισαγωγή' και η 'διαγραφή' (ή 'εξαγωγή') ενός στοιχείου/κόμβου[1]. Στη συνέχεια, παρουσιάζονται οι αλγόριθμοι για τις δύο αυτές πράξεις και για τους δύο τρόπους αναπαράστασης της στοίβας.

Συνεχόμενη Αναπαράσταση

Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου από μια συνεχόμενη στοίβα παρουσιάζονται στα παρακάτω πλαίσια.

ΑΛΓΟΡΙΘΜΟΣ 4.1: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ένας πίνακας (S), το μέγεθός του (N), ο δείκτης κορυφής της αντίστοιχης στοίβας (T) και η προς εισαγωγή τιμή (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα υπερχείλισης στην περίπτωση γεμάτης στοίβας, αλλιώς τίποτα. Εσωτερικά, καταχωρείται η τιμή στη στοίβα σε περίπτωση επιτυχίας.

```
SS-EISAGWGH (S, N, T, X)
1   if T = N                          {Έλεγχος γεμάτης στοίβας}
2     then print 'ΥΠΕΡΧΕΙΛΙΣΗ'        {Μήνυμα υπερχείλισης}
3     else T ← T+1                    {Ενημέρωση κορυφής}
4          S[T] ← X                   {Καταχώρηση στοιχείου}
    endif
```

[1] Στην ξένη βιβλιογραφία χρησιμοποιούνται συνήθως οι όροι, 'push' (ώθηση ή προώθηση) για την 'εισαγωγή' και 'pop' (απώθηση) ή 'pull' (εξαγωγή) για τη 'διαγραφή'.

ΑΛΓΟΡΙΘΜΟΣ 4.2: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ένας πίνακας (S) και ο δείκτης κορυφής της αντίστοιχης στοίβας (T).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα στην περίπτωση άδειας στοίβας, αλλιώς το εξαγχθέν στοιχείο. Εσωτερικά, διαγράφεται η κορυφή της στοίβας σε περίπτωση επιτυχίας.

```
SS-DIAGRAFH (S, T)
1   if T = 0                          {Έλεγχος άδειας στοίβας}
2       then print 'ΑΔΕΙΑ ΣΤΟΙΒΑ'      {Μήνυμα άδειας στοίβας}
3       else print S[T]               {Επιστροφή στοιχείου}
4            T ← T-1                  {Ενημέρωση κορυφής}
    endif
```

Η λογική των δύο αλγορίθμων είναι απλή. Στον Αλγόριθμο 4.1 (SS-EISAGWGH), μετά τον έλεγχο για το αν η στοίβα είναι γεμάτη και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) αυξάνεται ο δείκτης κορυφής της στοίβας κατά ένα, οπότε «δείχνει» την επόμενη κενή θέση στον πίνακα (βήμα 3) και κατόπιν καταχωρείται το στοιχείο στην κενή αυτή θέση (βήμα 4).

Στον Αλγόριθμο 4.2 (SS-DIAGRAFH), μετά τον έλεγχο για το αν η στοίβα είναι άδεια και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) επιστρέφει την τιμή του στοιχείου της κορυφής (βήμα 3) και κατόπιν μειώνεται ο δείκτης κορυφής της στοίβας κατά ένα, οπότε «δείχνει» την προηγούμενη θέση στη στοίβα. Η επιστροφή της τιμής γίνεται, διότι συνήθως ένα στοιχείο διαγράφεται από μια στοίβα για να χρησιμοποιηθεί κάπου.

Συνδεδεμένη Αναπαράσταση

Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου από μια απλά συνδεδεμένη στοίβα παρουσιάζονται στα παρακάτω πλαίσια.

ΑΛΓΟΡΙΘΜΟΣ 4.3: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ο δείκτης στον προς εισαγωγή κόμβο (P) και ο δείκτης στην κορυφή μιας συνδεδεμένης λίστας (S).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα στην περίπτωση ανυπαρξίας του προς εισαγωγή κόμβου, αλλιώς τίποτα. Εσωτερικά, εισάγεται ο νέος κόμβος στην στοίβα, σε περίπτωση επιτυχίας.

```
DS-EISAGWGH (P, S)
1    if P=NIL                                  {Έλεγχος δείκτη}
2      then print 'ΑΝΥΠΑΡΚΤΟΣ ΚΟΜΒΟΣ'          {Μήνυμα λάθους}
3      else DEIKTHS(KOMBOS(P)) ← S              {Καταχώρηση δείκτη}
4           S ← P                               {Ενημέρωση δείκτη}
     endif
```

ΑΛΓΟΡΙΘΜΟΣ 4.4: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος: Ο δείκτης στην κορυφή μιας συνδεδεμένης λίστας (S).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα σε περίπτωση άδειας στοίβας, αλλιώς το στοιχείο του διαγραφέντος κόμβου. Εσωτερικά, διαγράφεται η κορυφή της στοίβας σε περίπτωση επιτυχίας.

```
DS-DIAGRAFH (S)
1    if S = NIL                                {Έλεγχος κενής στοίβας}
2      then print 'ΑΔΕΙΑ ΣΤΟΙΒΑ'                {Μήνυμα κενής στοίβας}
3      else print STOIXEIO(KOMBOS(S))           {Επιστροφή στοιχείου κορυφής}
4           S ← DEIKTHS(KOMBOS(S))              {Ενημέρωση δείκτη κορυφής}
     endif
```

Η λογική και αυτών των δύο αλγορίθμων είναι απλή. Στον Αλγόριθμο 4.3 (DS-EISAGWGH), μετά τον έλεγχο για το αν ο δείκτης στον προς εισαγωγή κόμβο είναι έγκυρος και την επιστροφή κατάλληλου μηνύματος (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) γίνεται εναλλαγή των δεικτών. Ο δείκτης του προς εισαγωγή κόμβου παίρνει την τιμή του S, ώστε να «δείχνει» στον προηγούμενο κορυφαίο κόμβο (βήμα 3), ενώ ο δείκτης κορυφής παίρνει την τιμή του P, ώστε να δείχνει στο νέο κόμβο που είναι η νέα κορυφή.

Στον Αλγόριθμο 4.4 (DS-DIAGRAFH), μετά τον έλεγχο για το αν η στοίβα είναι άδεια και την επιστροφή κατάλληλου μηνύματος (βήμα-

τα 1-2), στον κυρίως αλγόριθμο (βήματα 3-4) επιστρέφει την τιμή του στοιχείου της κορυφής (βήμα 3) και κατόπιν ενημερώνεται ο δείκτης κορυφής της στοίβας, οπότε «δείχνει» στον προηγούμενο κόμβο στη στοίβα που τώρα γίνεται η νέα κορυφή.

Παράδειγμα 4.1

Διαγραφή Κ πρώτων στοιχείων συνεχόμενης στοίβας

Θεωρούμε μια συνεχόμενη στοίβα. Το ζητούμενο είναι να σχεδιαστεί ένας αλγόριθμος που να διαγράφει (εξάγει) τα Κ πρώτα στοιχεία της στοίβας και να επιστρέφει το Κ-οστό στοιχείο. Αν δεν υπάρχουν Κ στοιχεία στη στοίβα, τότε να επιστρέφει το τελευταίο διαγραφέν στοιχείο και μήνυμα άδειας στοίβας. Ο αλγόριθμος παρουσιάζεται στο παρακάτω πλαίσιο ως Αλγόριθμος Π4.1 (SS-DIAGRAFH-KP).

Το βασικό βήμα (βήμα 3) και η ενημέρωση του μετρητή (βήμα 4) βρίσκονται μέσα στη διάταξη επανάληψης while (βήματα 2-4). Η επανάληψη συνεχίζεται, εφόσον ο δείκτης κορυφής (Τ) παραμένει θετικός και ο μετρητής (Ι) όχι μεγαλύτερος του Κ. Σταματά επομένως, όταν είτε Τ = 0 είτε Ι > Κ (πιο συγκεκριμένα Ι = Κ+1). Αυτές είναι και οι δύο περιπτώσεις στις οποίες βασίζεται η σύνθετη διάταξη if (βήματα 5-8) που ακολουθεί. Αν συμβαίνει η δεύτερη περίπτωση (επιτυχία) (βήμα 5), τότε επιστρέφει το Κ-οστό διαγραφέν στοιχείο (βήμα 6). Αν όχι, τότε επιστρέφει το τελευταίο διαγραφέν στοιχείο και το μήνυμα άδειας στοίβας.

ΑΛΓΟΡΙΘΜΟΣ Π4.1: ΔΙΑΓΡΑΦΗ Κ ΣΤΟΙΧΕΙΩΝ ΣΥΝΕΧΟΜΕΝΗΣ ΣΤΟΙΒΑΣ

Είσοδος: Ένας πίνακας (S), ο δείκτης κορυφής της αντίστοιχης στοίβας (T) και ένας ακέραιος αριθμός (K).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα σε περίπτωση άδειας στοίβας ή στοίβας με λιγότερα από K στοιχεία, αλλιώς το τελευταίο από τα διαγραφέντα στοιχεία. Εσωτερικά, διαγράφονται τα K πρώτα ή όλα στοιχεία της στοίβας, σε περίπτωση επιτυχίας.

```
SS-DIAGRAFH-KP (S, T, K)
1    I ← 1                          {Αρχικοποίηση μετρητή}
2    while (T > 0) and (I ≤ K)      {Συνθήκη επανάληψης}
3      T ← T-1                      {Ενημέρωση δείκτη κορυφής}
4      I ← I+1                      {Ενημέρωση μετρητή}
     endwhile
5    if I = K +1                    {Έλεγχος επιτυχίας}
6      then print S[T+1]            {Επιστροφή K-οστού στοιχείου}
7      else print S[1]              {Επιστροφή τελευταίου στοιχείου}
8           print 'ΑΔΕΙΑ ΣΤΟΙΒΑ'    {Μήνυμα άδειας στοίβας}
     endif
```

Άσκηση Αυτοαξιολόγησης 4.1

Δίνεται μια μη κενή συνδεμένη στοίβα και ένας ακέραιος αριθμός K. Να σχεδιαστεί αλγόριθμος που να διαγράφει τα K πρώτα στοιχεία της στοίβας και να επιστρέφει το K-οστό στοιχείο. Σε περίπτωση που δεν υπάρχουν K στοιχεία στη στοίβα διαγράφει τα υπάρχοντα και επιστρέφει το τελευταίο διαγραφέν στοιχείο και μήνυμα άδειας στοίβας.

4.2 Εφαρμογές στοίβας

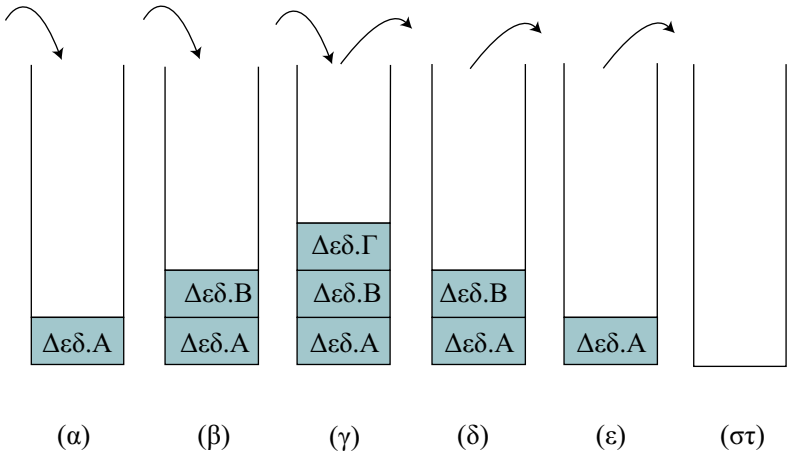
4.2.1 Κλήση υποπρογραμμάτων

Μια εφαρμογή της στοίβας στην επιστήμη των υπολογιστών σχετίζεται με την κλήση υποπρογραμμάτων. Ας υποθέσουμε ότι έχουμε ένα κυρίως πρόγραμμα Α και τρία υποπρογράμματα, τα Β, Γ και Δ, και ότι το Α καλεί το Β, το Β καλεί το Γ και το Γ καλεί το Δ κατά την εκτέλεσή του. Δηλαδή, για την εκτέλεση του Α απαιτείται η εκτέλεση όλων των υποπρογραμμάτων και επομένως, ενώ ξεκινά πρώτο τελειώνει

τελευταίο. Το Β για να εκτελεστεί χρειάζεται η εκτέλεση των Γ και Δ και επομένως, ενώ ξεκινά δεύτερο, τελειώνει προτελευταίο κ.ο.κ. Το Δ που ξεκινά τελευταίο, τελειώνει πρώτο. Αυτή η κατάσταση, προφανώς, συνιστά μια λογική LIFO και επομένως ευνοεί τη χρήση μιας στοίβας για τον χειρισμό των κλήσεων των υποπρογραμμάτων.

Για το σκοπό αυτό χρησιμοποιείται μια στοίβα στην οποία καταχωρούνται οι διευθύνσεις των περιοχών της μνήμης όπου κρατούνται τα δεδομένα (η κατάσταση) κάθε (υπο)προγράμματος, όταν εγκαταλείπεται προσωρινά για να εκτελεστεί κάποιο άλλο υποπρόγραμμα. Τα δεδομένα αυτά, που συχνά ονομάζονται *εγγραφή ενεργοποίησης* (*activation record*) ή *πλαίσιο στοίβας* (*stack frame*), περιλαμβάνουν όλα τα απαραίτητα στοιχεία για να επανέλθει η εκτέλεση του (υπο)προγράμματος στην κατάσταση που ήταν όταν διακόπηκε.

Στο Σχήμα 4.2 απεικονίζονται οι καταστάσεις της στοίβας κατά την εκτέλεση του προγράμματος Α. Πρώτο ξεκινά το πρόγραμμα Α και σε κάποια στιγμή καλεί το υποπρόγραμμα Β. Τότε, τα δεδομένα του Α κατ' εκείνη τη στιγμή, τοποθετούνται σε μια προσωρινή περιοχή της μνήμης και η διεύθυνση της περιοχής αυτής εισάγεται στη στοίβα (Σχήμα 4.2α) και ξεκινά η εκτέλεση του Β. Σε κάποια στιγμή, το Β καλεί το υποπρόγραμμα Γ και τότε τα δεδομένα του Β τοποθετούνται στη μνήμη και η αντίστοιχη διεύθυνση στη στοίβα (Σχήμα 4.2β) κ.ο.κ. μέχρι να κληθεί το Δ από το Γ. Μόλις τελειώσει το Δ, εξάγεται (διαγράφεται) η διεύθυνση των δεδομένων του Γ από τη στοίβα (Σχήμα 4.2δ), ανακαλείται το περιεχόμενό της και συνεχίζεται η εκτέλεση του υποπρογράμματος Γ από εκεί που σταμάτησε κ.ο.κ. μέχρι να τελειώσει και η εκτέλεση του Β, οπότε εξάγεται και η διεύθυνση των δεδομένων του Α και η στοίβα μένει κενή (Σχήμα 4.2στ).



Σχήμα 4.2

Στοίβα για την κλήση υποπρογραμμάτων

4.2.2 Εκτίμηση αριθμητικών εκφράσεων

Μια άλλη εφαρμογή της στοίβας στην επιστήμη των υπολογιστών αφορά την εκτίμηση αριθμητικών εκφράσεων. Ο συνήθης τρόπος (συμβολισμός) με τον οποίο γράφουμε τις αριθμητικές εκφράσεις ονομάζεται *ενδοθεματική μορφή* (*infix notation*). Για παράδειγμα, τέτοιας μορφής είναι οι εκφράσεις (α+β), (α+β)*γ, α*β+γ*δ, στις οποίες οι αριθμητικοί *τελεστές* (*operators*), (π.χ. +, *, -, /) βρίσκονται ανάμεσα στα στοιχεία στα οποία εφαρμόζονται, δηλαδή ανάμεσα στους *τελεστέους* (*operands*).

Εκτός όμως της ενδοθεματικής μορφής χρησιμοποιούνται και άλλες μορφές γραφής αριθμητικών εκφράσεων. Μια τέτοια είναι η λεγόμενη *προθεματική μορφή* (*prefix notation*) ή *Πολωνική μορφή* (*Polish notation*)[2], στην οποία οι τελεστές προηγούνται των τελεστέων και δεν απαιτούνται παρενθέσεις. Οι προηγούμενες εκφράσεις σε προθεματική μορφή γράφονται, +αβ, *+αβγ, +*αβ*γδ.

Μια τρίτη μορφή είναι η *μεταθεματική μορφή* (*postfix notation*) ή *αντίστροφη Πολωνική μορφή* (*reverse Polish notation*), στην οποία οι τελεστές γράφονται μετά από τους τελεστέους (αριθμούς). Π.χ., οι παραπάνω αριθμητικές εκφράσεις σε μεταθεματική μορφή γράφονται, αβ+, αβ+γ*, αβ*γδ*+.

[2] Η μορφή αυτή επινοήθηκε από τον Πολωνό μαθηματικό Jan Lukasiewicz το 1951, απ' όπου και η ονομασία 'Πολωνική'

Οι μορφές αυτές βρίσκουν εφαρμογή στα μεταφραστικά προγράμματα (μεταγλωττιστές, διερμηνευτές) των γλωσσών προγραμματισμού υψηλού επιπέδου. Μερικά μεταφραστικά προγράμματα χρησιμοποιούν εσωτερικά την μεταθεματική μορφή. Έτσι, ο υπολογισμός της τιμής μιας αριθμητικής έκφρασης γίνεται σε δύο στάδια. Στο πρώτο γίνεται μετατροπή της ενδοθεματικής μορφής σε μεταθεματική, ενώ στο δεύτερο στάδιο γίνεται υπολογισμός της τιμής (εκτίμηση) της έκφρασης από την μεταθεματική της μορφή. Και οι δύο αλγόριθμοι που υλοποιούν τα δύο αυτά στάδια χρησιμοποιούν σαν βάση μια στοίβα. Στη συνέχεια, θα αναφερθούμε στο δεύτερο αλγόριθμο, αυτόν που υπολογίζει την τιμή μιας παράστασης από τη μεταθεματική της μορφή.

Πίνακας 4.1

Βήμα	Στοιχείο	Στοίβα	Υπολογισμοί
1	6	6	
2	3	3 6	
3	4	4 3 6	4 * 3 = 12
4	*	12 6	12 + 6 = 18
5	+	18	
6	2	2 18	18 / 2 = 9
7	/	9	

Θα περιγράψουμε τα βήματα του αλγορίθμου για μια συγκεκριμένη αριθμητική έκφραση, την 6 3 4 * + 2 / (δηλαδή την α β γ * + δ / με α = 6, β = 3, γ = 4 και δ = 2), που είναι η μεταθεματική μορφή της (6 + 3*4)/2. Τα βήματα και οι αντίστοιχες ενέργειες φαίνονται στον Πίνακα 4.1, όπου βάση της στοίβας είναι ο δεξιότερα ευρισκόμενος αριθμός (3η στήλη). Ο αλγόριθμος βασίζεται στην εξής λογική: Σαρώνεται η μεταθεματική έκφραση και (α) όταν συναντάται αριθμός, τότε προωθείται (εισάγεται) στη στοίβα, (β) όταν συναντάται τελεστής τότε εξάγουμε (διαγράφουμε) τα δύο τελευταία στοιχεία (αριθμούς) της στοίβας, εφαρμόζουμε την πράξη του τελεστή σ' αυτά, υπολογίζουμε το αποτέλεσμα και το εισάγουμε στη στοίβα. Όταν έχουμε διέλθει όλα τα στοιχεία της έκφρασης τότε αυτό που μένει στη στοίβα είναι το αποτέλεσμα, η τιμή της αριθμητικής έκφρασης.

Άσκηση Αυτοαξιολόγησης 4.2

Σε μια αριθμητική έκφραση, συχνά χρησιμοποιούμε διάφορους τύπους παρενθέσεων, όπως π.χ., στην έκφραση {α + (β - [γ + δ] /(χ - ψ))}*(φ - (ω - (κ - [λ - μ]))). Εξηγήστε πώς χρησιμοποιώντας μια στοίβα μπορούμε να εξετάσουμε αν υπάρχει ο ίδιος αριθμός αριστερών και δεξιών παρενθέσεων και αν για κάθε δεξιά υπάρχει μια αριστερή του ίδιου τύπου. Η εξήγηση (αλγόριθμος) θα είναι η λύση του προβλήματος σε φυσική γλώσσα.

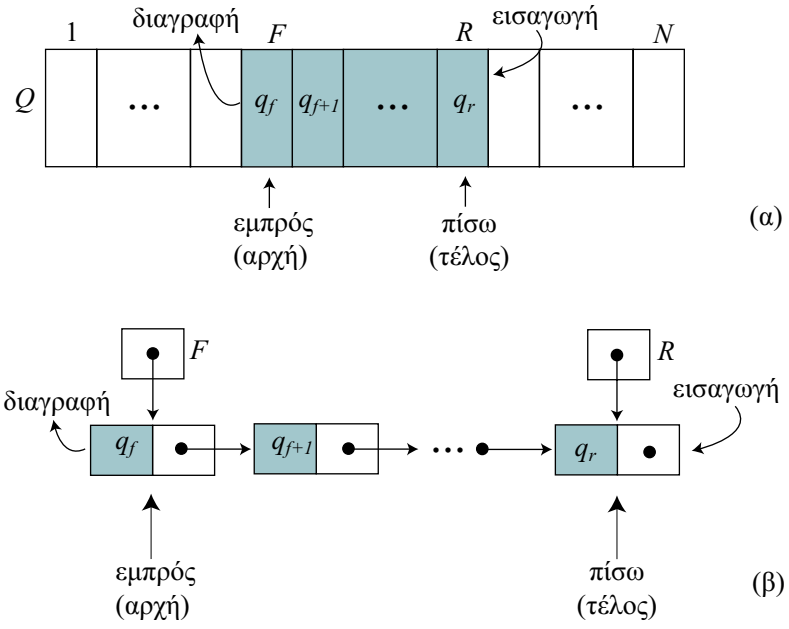
4.3 Ουρά

4.3.1 Ορισμός και αναπαράσταση

Στην καθημερινή μας ζωή υπάρχουν καταστάσεις που δεν μπορούν να περιγραφούν με βάση την έννοια της στοίβας. Τέτοιες καταστάσεις είναι οι ουρές αναμονής σε μια τράπεζα, στη στάση του λεωφορείου, για τις θέσεις μια αεροπορικής πτήσης κλπ. Στις περιπτώσεις αυτές, αυτός που προσήλθε πρώτος στην ουρά εξυπηρετείται και πρώτος. Ακολουθείται δηλαδή μια διαφορετική λογική από εκείνη στις στοίβες, που ονομάζεται FIFO (First-In First-Out). Η δομή δεδομένων που μπορεί να αναπαραστήσει αυτή τη λογική είναι ένας ειδικός τύπος λίστας που ονομάζεται *ουρά* (*queue*). Οι περιορισμοί στην ουρά, σε σχέση με μια γενική λίστα, είναι οι εξής: α) Εισαγωγές στοιχείων μπορούν να γίνονται μόνο από το ένα άκρο της, που ονομάζεται *πίσω* (*rear*) ή *τέλος* και β) διαγραφές μπορούν να γίνονται μόνο από το άλλο άκρο της, που ονομάζεται *εμπρός* (*front*) ή *αρχή*. ■

■ Μια **ουρά** είναι μια λίστα, στην οποία μπορούν να γίνονται εισαγωγές στοιχείων μόνο στο ένα άκρο της (πίσω ή τέλος) και διαγραφές στοιχείων μόνο από το άλλο άκρο της (εμπρός ή αρχή).

Η αναπαράσταση μιας ουράς μπορεί να γίνει, όπως και στην περίπτωση της στοίβας, με δύο τρόπους, είτε με τον συνεχόμενο είτε με τον συνδεδεμένο, δηλαδή είτε με πίνακα είτε με δείκτες, οπότε διακρίνουμε δύο τύπους ουράς, *συνεχόμενη ουρά* (*contiguous queue*) και *συνδεδεμένη ουρά* (*linked queue*). Στο Σχήμα 4.3 απεικονίζεται μια ουρά και με τους δύο τρόπους αναπαράστασης.



Σχήμα 4.3

(α) Συνεχόμενη αναπαράσταση, (β) συνδεδεμένη αναπαράσταση ουράς

Μια συνεχόμενη ουρά σε μια γλώσσα υψηλού επιπέδου, μπορεί να οριστεί σαν μια εγγραφή με τρία πεδία: δύο μετρητές που αντιπροσωπεύουν την αρχή και το τέλος της και ένας πίνακας για την αποθήκευση των στοιχείων της. Μια συνδεδεμένη ουρά μπορεί να οριστεί σαν μια εγγραφή με δύο πεδία που είναι μεταβλητές δείκτη. Η μια δείχνει στην αρχή και η άλλη στο τέλος της ουράς. Ο κάθε κόμβος ορίζεται όπως και σε μια συνδεδεμένη λίστα.

4.3.2 Πράξεις σε ουρά

Όπως στη στοίβα, έτσι και στην ουρά δύο είναι οι κυριότερες πράξεις, η εισαγωγή και η διαγραφή[3]. Οι αντίστοιχοι αλγόριθμοι και για τις δύο αναπαραστάσεις παρουσιάζονται στη συνέχεια.

Συνεχόμενη Αναπαράσταση

Στη συνεχόμενη αναπαράσταση η ‘άδεια ουρά’ ορίζεται από τη συνθήκη F=R=0, ενώ η ουρά με ένα στοιχείο από τη συνθήκη F=R=1. Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου παρουσιάζονται στα παρακάτω πλαίσια ως Αλγόριθμοι 4.5 και 4.6.

[3] Συχνά στην ξένη βιβλιογραφία χρησιμοποιούνται οι όροι ‘enqueue’ και ‘dequeue’ (ή 'serve') για τη ‘εισαγωγή’ και ‘διαγραφή’ αντίστοιχα.

ΑΛΓΟΡΙΘΜΟΣ 4.5: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΟΥΡΑ

Είσοδος: Ένας πίνακας (Q), το μέγεθός του (N), ο δείκτης αρχής (F) και τέλους (R) της αντίστοιχης ουράς και η προς εισαγωγή τιμή (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα γεμάτης ουράς, αλλιώς τίποτα. Εσωτερικά, καταχωρείται η τιμή στο τέλος της ουράς, σε περίπτωση επιτυχίας.

```
SQ-EISAGWGH (Q, N, F, R, X)
1    if R > N                          {Έλεγχος γεμάτης ουράς}
2      then print 'ΓΕΜΑΤΗ ΟΥΡΑ'         {Μήνυμα γεμάτης ουράς}
3      else R ← R+1                    {Ενημέρωση δείκτη τέλους}
4           Q[R] ← X                   {Καταχώρηση τιμής}
5           if F = 0                   {Έλεγχος άδειας ουράς}
6             then F ← 1               {Ενημέρωση δείκτη αρχής}
            endif
     endif
```

Η λογική του Αλγορίθμου 4.5 (SQ-EISAGWGH) είναι η εξής: Μετά τον έλεγχο για γεμάτη ουρά (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-6), κατ' αρχήν (βήμα 3) αυξάνεται ο δείκτης τέλους της ουράς κατά ένα, ώστε να δείχνει στην επόμενη κενή θέση του πίνακα, και κατόπιν καταχωρείται η τιμή X στη θέση αυτή (βήμα 4). Στη συνέχεια, ελέγχουμε αν η ουρά ήταν κενή (βήμα 5) και ενημερώνουμε και τον δείκτη αρχής (βήμα 6).

Στον Αλγόριθμο 4.6 (SQ-DIAGRAFH) έχουμε τα εξής: Μετά τον έλεγχο για άδεια ουρά (βήματα 1-2), στον κυρίως αλγόριθμο (βήματα 3-6), κατ' αρχήν (βήμα 3) επιστρέφει την αρχή της ουράς (για όποια χρήση). Κατόπιν (βήμα 4), αυξάνεται ο δείκτης αρχής της ουράς κατά ένα, ώστε να δείχνει στο επόμενο στοιχείο του πίνακα (της ουράς). Στη συνέχεια, ελέγχουμε αν η ουρά γίνεται άδεια (βήμα 5), δηλαδή αν υπήρχε μόνο ένα στοιχείο πριν τη διαγραφή (F = R = 1), και ενημερώνουμε τους δείκτες αρχής και τέλους (βήμα 6).

ΑΛΓΟΡΙΘΜΟΣ 4.6: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΟΥΡΑ

Είσοδος: Ένας πίνακας (Q), ο δείκτης αρχής (F) και τέλους (R) της αντίστοιχης ουράς.
Έξοδος: Εξωτερικά, επιστρέφει μήνυμα άδειας ουράς, αλλιώς το διαγραφέν στοιχείο. Εσωτερικά, διαγράφεται το στοιχείο από την αρχή της ουράς, σε περίπτωση επιτυχίας.

```
SQ-DIAGRAFH (Q, F, R)
1   if F = 0                          {Έλεγχος άδειας ουράς}
2     then print 'ΑΔΕΙΑ ΟΥΡΑ'          {Μήνυμα άδειας ουράς}
3     else print Q[F]                 {Επιστροφή διαγραφέντος}
4          F ← F+1                    {Ενημέρωση δείκτη αρχής}
5          if F > R                   {Έλεγχος άδειας ουράς}
6            then F ← 0, R ← 0        {Ενημέρωση δεικτών}
           endif
    endif
```

Συνδεδεμένη Αναπαράσταση

Στη συνδεδεμένη αναπαράσταση η 'άδεια ουρά' ορίζεται από τη συνθήκη $F = R =$ N I L, ενώ η ουρά με ένα στοιχείο από τη συνθήκη $F = R \neq$ NIL. Οι αλγόριθμοι για την εισαγωγή και διαγραφή ενός στοιχείου παρουσιάζονται στα παρακάτω πλαίσια ως Αλγόριθμοι 4.7 και 4.8.

ΑΛΓΟΡΙΘΜΟΣ 4.7: ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΟΥΡΑ

Είσοδος: Ο δείκτης αρχής (F) και τέλους (R) μιας ουράς και ο δείκτης (P) στον προς εισαγωγή κόμβο.
Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, εισάγεται ο κόμβος στο τέλος της ουράς.

```
DQ-EISAGWGH (F, R, P)
1   if F=NIL                          {Έλεγχος άδειας ουράς}
2     then F ← P                      {Ενημέρωση δείκτη αρχής}
3     else DEIKTHS(KOMBOS(R)) ← P     {Ενημέρωση τελευταίου κόμβου}
4          DEIKTHS(KOMBOS(P)) ← NIL   {Δημιουργία τελευταίου κόμβου}
    endif
5   R ← P                             {Ενημέρωση δείκτη τέλους}
```

Η λογική του Αλγορίθμου 4.7 (DQ-EISAGWGH) έχει ως ακολούθως: Κατ' αρχήν, εξετάζεται αν η ουρά είναι άδεια (βήμα 1). Αν είναι, τότε απλώς ο δείκτης αρχής παίρνει την τιμή του δείκτη στον προς εισαγωγή κόμβο (βήμα 2). Αλλιώς, ο δείκτης του τελευταίου κόμβου της ουράς παίρνει την τιμή του P (βήμα 3), οπότε «δείχνει» στον προς εισαγωγή κόμβο, που γίνεται ο τελευταίος της ουράς (βήμα 4). Τέλος, σε κάθε περίπτωση, ενημερώνεται ο δείκτης τέλους (βήμα 5).

ΑΛΓΟΡΙΘΜΟΣ 4.8: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΟΥΡΑ

Είσοδος: Ο δείκτης αρχής (F) και τέλους (R) μιας ουράς.
Έξοδος: Εξωτερικά, επιστρέφει μήνυμα στην περίπτωση άδειας ουράς, αλλιώς το διαγραφέν στοιχείο. Εσωτερικά, διαγράφεται το στοιχείο από την αρχή της ουράς, σε περίπτωση επιτυχίας.

```
DQ-DIAGRAFH (F, R, X)
1    if F = NIL                              {Έλεγχος άδειας ουράς}
2       then print 'ΑΔΕΙΑ ΟΥΡΑ'               {Μήνυμα άδειας ουράς}
3       else print STOIXEIO(KOMBOS(F))        {Επιστροφή στοιχείου}
4            F ← DEIKTHS(KOMBOS(F))           {Ενημέρωση δείκτη αρχής}
5            if F = NIL                       {Έλεγχος άδειας ουράς}
6               then R ← NIL                  {Ενημέρωση δείκτη τέλους}
             endif
    endif
```

Στον Αλγόριθμο 4.8 (DQ-DIAGRAFH), κατ' αρχήν εξετάζεται αν η ουρά είναι κενή (βήμα 1) και επιστρέφεται κατάλληλο μήνυμα (βήμα 2). Αλλιώς, επιστρέφει την τιμή του στοιχείου του πρώτου κόμβου της ουράς (βήμα 3) και ενημερώνεται ο δείκτης F, ώστε να «δείχνει» στο δεύτερο κόμβο, που τώρα γίνεται πρώτος (βήμα 4). Τέλος, ελέγχουμε αν με τη διαγραφή άδειασε η ουρά (βήμα 5), οπότε ενημερώνουμε τον δείκτη R.

Παράδειγμα 4.2

Άδειασμα συνδεδεμένης ουράς σε άδεια συνεχόμενη στοίβα

Θεωρούμε μια απλά συνδεδεμένη ουρά και ζητείται να σχεδιαστεί αλγόριθμος που να την αδειάζει και να τοποθετεί τα στοιχεία των κόμβων της, σε μια άδεια συνεχόμενη στοίβα.

Η λογική του ζητούμενου αλγορίθμου είναι σχετικά απλή. Γίνονται διαδοχικές διαγραφές των κόμβων της ουράς και κάθε φορά, τοποθετούμε το στοιχείο τού προς διαγραφή κόμβου στη συνεχόμενη στοίβα. Επομένως, η βάση για τη σχεδίαση είναι ο βασικός αλγόριθμος διαγραφής (Αλγόριθμος 4.8). Ο ζητούμενος αλγόριθμος δίνεται στο παρακάτω πλαίσιο ως Αλγόριθμος Π4.2 (DQ-ADEIASMA-SS).

Στο βήμα 1 τίθεται η τιμή του δείκτη κορυφής της στοίβας στο μηδέν (άδεια στοίβα). Στη συνέχεια, ενεργοποιείται η επαναληπτική διαδικασία διαγραφής από την ουρά και καταχώρησης στη στοίβα μέσω της διάταξης while (βήματα 2-5). Σε κάθε επανάληψη, ο δείκτης κορυφής αυξάνει κατά ένα, ώστε να δείχνει στην επόμενη κενή θέση της στοίβας (βήμα 3), καταχωρείται το στοιχείο του τρέχοντος κόμβου αρχής της ουράς στη στοίβα (βήμα 4) και ενημερώνεται ο δείκτης αρχής της ουράς, ώστε να δείχνει στον επόμενο κόμβο (βήμα 5). Τέλος, ενημερώνεται και ο δείκτης τέλους της ουράς (άδεια ουρά) (βήμα 6). Επειδή θεωρούμε στοίβα με ικανό μέγεθος πίνακα, δεν τίθεται θέμα ελέγχου υπερχείλισης της στοίβας.

ΑΛΓΟΡΙΘΜΟΣ Π4.2: ΑΔΕΙΑΣΜΑ ΣΥΝΔΕΔΕΜΕΝΗΣ ΟΥΡΑΣ ΣΕ ΣΤΟΙΒΑ

Είσοδος: Οι δείκτες αρχής (F) και τέλους (R) μιας συνδεδεμένης ουράς, ένας πίνακας (S) και ο δείκτης κορυφής της αντίστοιχης στοίβας (T).

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, αδειάζει η ουρά και γεμίζει μερικώς η στοίβα.

```
DQ-ADEIASMA-SS (F, R, S, T)
1    T ← 0                                  {Αρχικοποίηση δείκτη κορυφής}
2    while F ≠ NIL                          {Συνθήκη ελέγχου επανάληψης}
3      T ← T+1                              {Ενημέρωση δείκτη κορυφής}
4      S[T] ← STOIXEIO(KOMBOS(F))           {Καταχώρηση στη στοίβα}
5      F ← DEIKTHS(KOMBOS(F))               {Ενημέρωση δείκτη αρχής}
     endwhile
6    R ← NIL                                {Ενημέρωση δείκτη τέλους}
```

Άσκηση Αυτοαξιολόγησης 4.3

Δίνεται μια απλή συνεχόμενη ουρά και ζητείται να σχεδιαστεί αλγόριθμος που να αναστρέφει τη σειρά των στοιχείων της. (Αν δυσκολεύεστε να ξεκινήσετε, δείτε την υπόδειξη πριν την απάντηση της άσκησης αυτής, στο τέλος του βιβλίου.)

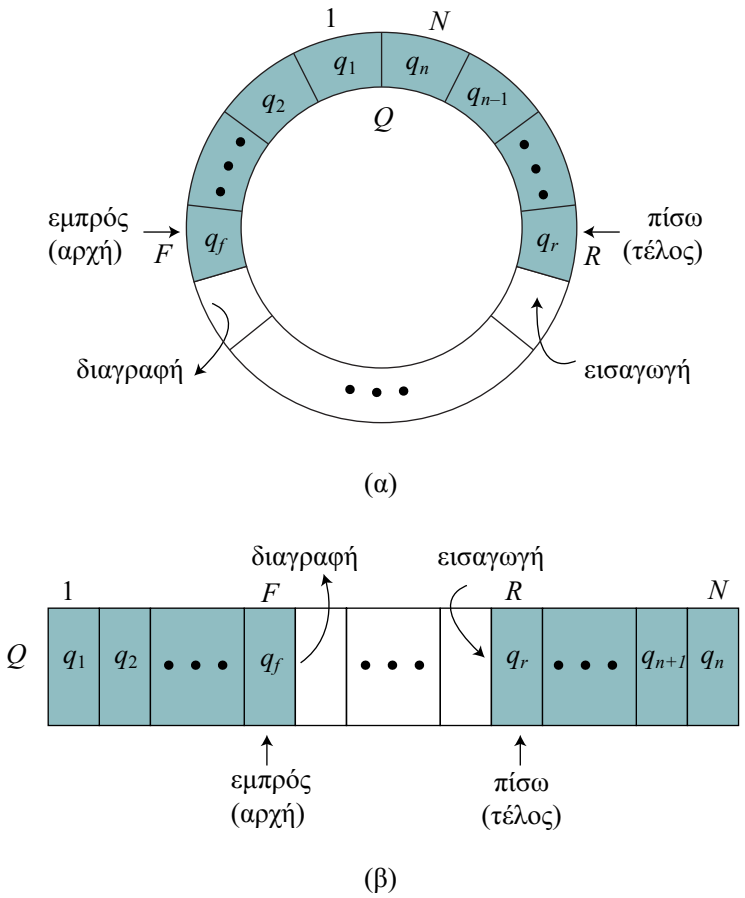
4.4 Ειδικοί τύποι ουράς

4.4.1 Κυκλική ουρά

Η υλοποίηση μιας συνεχόμενης ουράς με απλό πίνακα παρουσιάζει ορισμένα προβλήματα. Κατ' αρχήν, ένα πρόβλημα είναι ο καθορισμός του μεγέθους του πίνακα, όπως και στις γενικές λίστες. Επίσης, καθώς διαγράφουμε στοιχεία παραμένουν αχρησιμοποίητες θέσεις του πίνακα. Τέλος, όταν γίνει R = N, τότε δεν μπορούμε να εισάγουμε άλλα στοιχεία, έστω και αν υπάρχουν κενές θέσεις στο εμπρός μέρος του πίνακα λόγω διαγραφών.

Μια λύση στο πρόβλημα αυτό είναι, σε κάθε τέτοια περίπτωση, να μετακινούμε τα στοιχεία του πίνακα στην αρχή του και να επαναθέτουμε τα F και R. Η λύση αυτή όμως είναι χρονοβόρα. Μια άλλη, καλύτερη λύση, είναι να χρησιμοποιήσουμε *κυκλικό πίνακα* (*circular array*) για την υλοποίηση της ουράς, οπότε δημιουργείται μια *κυκλική ουρά* (*circular queue*). Σ' ένα κυκλικό πίνακα το τελευταίο (πρώτο) στοιχείο του έχει σαν επόμενο (προηγούμενο) το πρώτο (τελευταίο) στοιχείο του. Η σχηματική παράσταση μιας κυκλικής ουράς και του αντίστοιχου κυκλικού πίνακα απεικονίζονται στο Σχήμα 4.4.

Σε μια τέτοια ουρά, στην περίπτωση που R = N και θέλουμε να εισάγουμε ένα στοιχείο (Αλγόριθμος 4.5), αντί να αυξήσουμε το R κατά ένα (R←R + 1), επαναθέτουμε το R στην τιμή 'ένα' (R ← 1) και καταχωρούμε το στοιχείο στη νέα θέση (Q[R] ← X). Όταν F = N και θέλουμε να διαγράψουμε ένα στοιχείο (Αλγόριθμος 4.6), αντί να αυξήσουμε το F κατά ένα (F ← F + 1), επαναθέτουμε το F στην τιμή 'ένα' (F ← 1).



Σχήμα 4.4

(α) Σχηματική παράσταση κυκλικής ουράς, (β) αντίστοιχος κυκλικός πίνακας

4.4.2 Διπλή ουρά

Η *διπλή ουρά* (*double-ended queue* ή *deque*) είναι μια ουρά στην οποία επιτρέπονται εισαγωγές και εξαγωγές στοιχείων/κόμβων και από τα δύο άκρα. Ένας συνήθης τρόπος υλοποίησης μιας τέτοιας ουράς είναι μέσω ενός κυκλικού πίνακα.

4.4.3 Ουρά προτεραιότητας

Εκτός από τις συνήθεις ουρές που υπακούουν στη λογική FIFO, στην καθημερινή ζωή υπάρχουν ουρές που υπακούουν σε άλλη λογική. Τέτοια ουρά είναι η ουρά των ασθενών στο τμήμα επειγόντων περιστατικών ενός νοσοκομείου, όπου οι ασθενείς δεν εξυπηρετούνται με κριτήριο το χρόνο άφιξης, αλλά τη σοβαρότητα του περιστατικού.

Κάτι ανάλογο συμβαίνει και στα υπολογιστικά συστήματα, όπου στη διαχείριση κάποιων συμβάντων που σχηματίζουν μια ουρά, προηγούνται αυτά που έχουν σπουδαιότερη σημασία για την ομαλή λειτουργία του συστήματος, δηλαδή αυτά που έχουν μεγαλύτερη προτεραιότητα. Αυτό το μοντέλο διαχείρισης υπακούει σε μια άλλη λογική, που ονομάζεται λογική HPIFO (Highest-Priority-In First-Out).

Η αντίστοιχη δομή δεδομένων ονομάζεται *ουρά προτεραιότητας* (*priority queue*). Στην ουρά προτεραιότητας σημαντικό ρόλο δεν παίζει η σειρά εισαγωγής, όπως στη στοίβα και την ουρά, αλλά ο βαθμός προτεραιότητας. Ένας τρόπος αναπαράστασης μιας ουράς προτεραιότητας είναι μέσω μιας διατεταγμένης συνδεδεμένης λίστας. Στην περίπτωση αυτή, κάθε κόμβος έχει τρία τμήματα. Το πρώτο περιέχει την τιμή του στοιχείου του κόμβου, το δεύτερο περιέχει τον βαθμό προτεραιότητας του στοιχείου και το τρίτο τον δείκτη στον επόμενο κόμβο. Ο βαθμός προτεραιότητας, συνήθως, εκφράζεται με ένα ακέραιο αριθμό. Όσο μικρότερος είναι ο αριθμός αυτός τόσο μεγαλύτερη είναι η προτεραιότητα του αντίστοιχου στοιχείου. Οι κόμβοι επομένως, είναι διατεταγμένοι κατ' αύξοντα βαθμό προτεραιότητας (δηλαδή κατά φθίνουσα προτεραιότητα), ώστε πρώτος κόμβος να είναι πάντα αυτός με τη μεγαλύτερη προτεραιότητα. Δύο κόμβοι με την ίδια προτεραιότητα είναι διατεταγμένοι σύμφωνα με τη σειρά εισαγωγής.

Η διαδικασία διαγραφής (εξαγωγής) στοιχείου/κόμβου από μια ουρά προτεραιότητας γίνεται όπως και στην απλή ουρά. Αντίθετα, η εισαγωγή ενός στοιχείου/κόμβου απαιτεί πρώτα αναζήτηση της σωστής θέσης και μετά την καταχώρηση του στοιχείου.

Σύνοψη

Οι ***ειδικές λίστες*** είναι λίστες που θέτουν περιορισμούς όσον αφορά τις πράξεις σ' αυτές. Η ***στοίβα*** είναι ένας τύπος ειδικής λίστας στην οποία επιτρέπονται εισαγωγές και διαγραφές (εξαγωγές) μόνο από το ένα άκρο της, που ονομάζεται ***κορυφή***. Η ***ουρά***, ένας άλλος τύπος ειδικής λίστας, επιτρέπει εισαγωγές από το ένα άκρο, που ονομάζεται ***πίσω*** ή ***τέλος***, και διαγραφές (εξαγωγές) από το άλλο, που ονομάζεται ***εμπρός*** ή ***αρχή***.

Η στοίβα υπακούει στη ***λογική LIFO***, δηλαδή όποιο στοιχείο εισά-

γεται τελευταίο, αυτό διαγράφεται πρώτο. Η ουρά υπακούει στη ***λογική FIFO***, δηλαδή όποιο στοιχείο εισάγεται πρώτο, αυτό και εξάγεται πρώτο.

Η στοίβα και η ουρά μπορούν να αναπαρασταθούν είτε με συνεχόμενη είτε με συνδεδεμένη αναπαράσταση και βρίσκουν διάφορες εφαρμογές στην επιστήμη των Η/Υ. Κυριότερες πράξεις στις δομές αυτές είναι η εισαγωγή και η διαγραφή.

Η ***κυκλική ουρά*** είναι ένας ειδικός τύπος συνεχόμενης ουράς που λύνει ορισμένα προβλήματα της απλής συνεχόμενης ουράς. Ένας άλλος ειδικός τύπος ουράς είναι η ***ουρά προτεραιότητας*** που ακολουθεί τη ***λογική HPIFO***, δηλαδή εξάγεται πρώτο το στοιχείο με τη μεγαλύτερη προτεραιότητα, ανεξαρτήτως σειράς εισόδου του.

Οδηγός για περαιτέρω μελέτη

1. Μανωλόπουλος Ι., *Δομές Δεδομένων*, Τόμ. Α', ART of TEXT, 2η έκδοση, Θεσσαλονίκη 1992.

   Στην υποενότητα «3.3.2 Υπολογισμός Αριθμητικών Εκφράσεων» του βιβλίου αυτού, μπορείτε να βρείτε μια αναλυτική παρουσίαση της εφαρμογής της στοίβας στο πρόβλημα που υποδηλώνει ο τίτλος της υποενότητας και για τα δύο στάδια της διαδικασίας.

2. Main M. και Savitch W., *Data Structures and Other Objects*, Benjamin/Cummings, Redwood City, CA 1995.

   Στα κεφάλαια 6 και 7 του βιβλίου αυτού με τίτλους «Stacks» και «Queues» αντίστοιχα, παρουσιάζονται πολύ αναλυτικά οι στοίβες και οι ουρές με περιγραφές αλγορίθμων, διαφόρων εφαρμογών και υλοποιήσεων σε γλώσσα Pascal. Οι εφαρμογές περιλαμβάνουν: έλεγχο παρενθέσεων, εκτίμηση αριθμητικών εκφράσεων από διάφορες μορφές, αναδρομική κλήση, όσον αφορά τη στοίβα, και αναγνώριση παλίνδρομων φράσεων, προσομοίωση ουράς σε πλυντήριο αυτοκινήτων, όσον αφορά την ουρά.

3. Standish, A. T., *Data Structures*, *Algorithm and Software Principles*, Addison-Wesley, Reading, MA 1994.

   Στο κεφάλαιο 7 του βιβλίου αυτού, με τίτλο «Linear Data Structures-Stacks and Queues», παρουσιάζονται πολύ αναλυτικά

οι στοίβες και οι ουρές. Περιλαμβάνονται τρόποι αναπαράστασης και υλοποίησης, με παραδείγματα σε γλώσσα Pascal καθώς και ποικίλες εφαρμογές των δομών αυτών. Οι εφαρμογές που περιλαμβάνονται είναι παρόμοιες με αυτές της προηγούμενης βιβλιογραφικής αναφοράς.

Δέντρα

ΚΕΦΑΛΑΙΟ 5

Σκοπός

Στο κεφάλαιο αυτό παρουσιάζονται κατά βάση τα δυαδικά δέντρα, η σπουδαιότερη κατηγορία δέντρων. Η παρουσίαση αφορά τους τρόπους υλοποίησης τους, τις πράξεις σ' αυτά και τους αντίστοιχους αλγορίθμους.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό, θα είστε σε θέση να:

- *αναγράφετε τη σειρά επίσκεψης των κόμβων ενός δυαδικού δέντρου ανάλογα με τον τρόπο διαπέρασης,*
- *περιγράφετε τις αλλαγές κατά την εισαγωγή και τη διαγραφή στοιχείου σε ένα δυαδικό δέντρο αναζήτησης,*
- *περιγράφετε τις αλλαγές κατά την εισαγωγή και τη διαγραφή στοιχείου σε ένα δέντρο-σωρό,*
- *σχηματίζετε ένα δέντρο δυαδικής αναζήτησης ή ένα σωρό, όταν δίνονται τα στοιχεία του,*
- *διακρίνετε το είδος του δέντρου από την περιγραφή του.*

Έννοιες κλειδιά

- *Γενικό δέντρο*
- *Ρίζα*
- *Κόμβος*
- *Εσωτερικός κόμβος*
- *Τερματικός κόμβος*
- *Ακμή*
- *Κενό δέντρο*
- *Γονέας*
- *Παιδί*
- *Αριστερό παιδί*
- *Δεξί παιδί*
- *Υποδέντρο*
- *Αριστερό υποδέντρο*
- *Δεξί υποδέντρο*
- *Διαδρομή*
- *Επίπεδο ή βάθος κόμβου*
- *Ύψος δέντρου*
- *Πλήρες δέντρο*
- *Προδιατεταγμένη διαπέραση*
- *Ενδοδιατεταγμένη διαπέραση*

- *Μεταδιατεταγμένη διαπέραση*
- *Δυαδικό δέντρο αναζήτησης*
- *Δέντρο-σωρός ή σωρός*
- *Ισοζυγισμένο δέντρο*
- *Υψοζυγισμένο δέντρο*
- *Βαροζυγισμένο δέντρο*

Ενότητες Κεφαλαίου 5

5.1 Γενικά στοιχεία και ορισμοί

5.2 Δυαδικά δέντρα

5.3 Δυαδικά δέντρα αναζητήσης

5.4 Δέντρα - σωροί

5.5 Άλλες κατηγορίες δέντρων

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό παρουσιάζουμε τα δέντρα, ίσως τη σπουδαιότερη κατηγορία ανώτερων δομών δεδομένων. Στην πρώτη ενότητα δίνεται η περιγραφή ενός γενικού δέντρου, καθώς και ορισμένων βασικών του στοιχείων.

Στη δεύτερη ενότητα εξετάζουμε τα δυαδικά δέντρα, τη σπουδαιότερη κατηγορία δέντρων. Δίνεται ο ορισμός ενός δυαδικού δέντρου, περιγράφονται οι τρόποι αναπαράστασής του και οι διάφοροι τρόποι διαπέρασης, καθώς και ο αλγόριθμος για την προδιατεταγμένη διαπέραση.

Στην τρίτη ενότητα ασχολούμαστε με μια παραλλαγή των δυαδικών δέντρων, τα δυαδικά δέντρα αναζήτησης. Δίνεται ο ορισμός τους και οι αλγόριθμοι για τις πράξεις της αναζήτησης, εισαγωγής και διαγραφής ενός στοιχείου (κόμβου).

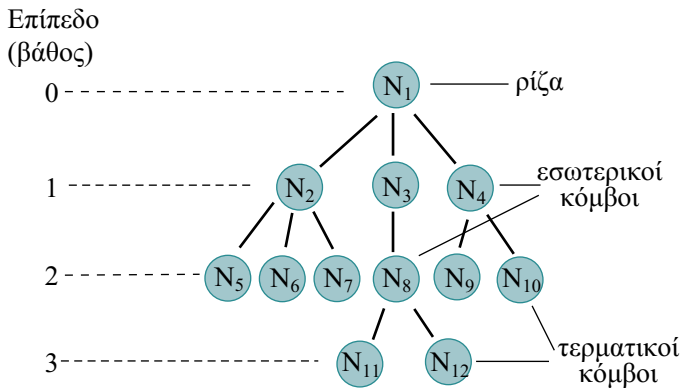
Στην τέταρτη ενότητα παρουσιάζουμε μια άλλη παραλλαγή των δυαδικών δέντρων, τα δέντρα-σωροί ή απλώς σωροί. Δίνεται ο ορισμός τους και περιγράφονται οι αλγόριθμοι για την εισαγωγή στοιχείου (κόμβου) σε ένα σωρό και τη διαγραφή της ρίζας ενός σωρού.

Τέλος, στην πέμπτη ενότητα αναφερόμαστε πολύ συνοπτικά σε διάφορους άλλους τύπους δέντρων, όπως τα δέντρα AVL και τα κόκκινα-μαύρα δέντρα, στο πλαίσιο της έννοιας των ισοζυγισμένων δέντρων.

5.1 Γενικά στοιχεία και ορισμοί

Τα δέντρα είναι από τις σπουδαιότερες δομές δεδομένων και ανήκουν στην κατηγορία των μη γραμμικών δομών. Ένα δέντρο είναι μια συλλογή από στοιχεία (κόμβους) του ίδιου τύπου που συνδέονται (σχετίζονται) μεταξύ τους κατά τρόπο πιο πολύπλοκο απ' ό,τι στους πίνακες και στις λίστες, που είναι γραμμικές δομές. Η συνήθης απεικόνιση του τρόπου αυτού συσχέτισης των κόμβων μοιάζει με δέντρο. Στο Σχήμα 5.1 παρουσιάζεται ένα δέντρο γενικής μορφής.

Κάθε δέντρο αποτελείται από *κόμβους* (*nodes*) και *ακμές* (*edges*). Οι ακμές είναι ευθύγραμμα τμήματα που συνδέουν τους κόμβους μεταξύ τους. Σε κάθε δέντρο υπάρχει ένας και μοναδικός κόμβος, που ονομάζεται *ρίζα* (*root*) του δέντρου. Από τη ρίζα μόνο ξεκινούν ακμές. Δεν υπάρχει ακμή που να καταλήγει στη ρίζα. Οι κόμβοι στους οποίους μόνο καταλήγουν ακμές ονομάζονται *τερματικοί κόμβοι* (*terminal nodes*) ή *φύλλα* (*leaves*). Οι υπόλοιποι κόμβοι, στους οποίους καταλήγουν και από τους οποίους ξεκινούν ακμές, ονομάζονται *εσωτερικοί* (*internal*) ή *μη τερματικοί* (*non terminal*) κόμβοι. Το *κενό δέντρο* (*null tree*) δεν έχει κανέναν κόμβο και καμία ακμή. Για παράδειγμα, στο δέντρο του Σχήματος 5.1 ο κόμβος N_1 είναι η ρίζα του δέντρου, οι N_2, N_3, N_4 και N_8 είναι εσωτερικοί κόμβοι και οι υπόλοιποι τερματικοί κόμβοι. Τέλος, *υποδέντρο* (*subtree*) ενός δέντρου είναι κάθε δέντρο που σχηματίζεται αν θεωρήσουμε ως ρίζα έναν οποιοδήποτε κόμβο του δέντρου.



Σχήμα 5.1
Ένα γενικό δέντρο

Σ' ένα δέντρο ορίζουμε τις παρακάτω «οικογενειακές» σχέσεις μεταξύ των κόμβων του. Ένας κόμβος N_x ονομάζεται *γονέας* (*parent*) ενός άλλου κόμβου N_y, αν από τον N_x ξεκινά μια ακμή που καταλήγει στον

N_y. Ο N_y τότε ονομάζεται *παιδί* (*child*) του N_x. Τα φύλλα ενός δέντρου δεν έχουν παιδιά. Π.χ. στο Σχήμα 5.1 ο κόμβος N_4 είναι γονέας των N_9, N_{10} και οι N_9, N_{10} είναι παιδιά του. Τα παιδιά ενός κόμβου, π.χ. τα N_9, N_{10} είναι μεταξύ τους *αδέρφια* (*siblings*). Κατ' επέκταση, έχουμε και τους όρους *απόγονος* (*descendant*) και *πρόγονος* (*ancestor*). Π.χ. ο κόμβος N_{12} είναι απόγονος του N_3, διότι είναι παιδί του παιδιού του (δηλαδή του N_8). Αντίστοιχα, ο κόμβος N_3 είναι πρόγονος του N_{12}.

Επίσης, σ' ένα δέντρο ορίζουμε και τις έννοιες που παρουσιάζονται στον Πίνακα 5.1. Τα παραδείγματα αναφέρονται στο δέντρο του Σχήματος 5.1.

Πίνακας 5.1

Ορισμός	Παράδειγμα - Παρατηρήσεις
Διαδρομή (*path*) από έναν κόμβο N_x σε έναν άλλο κόμβο N_y είναι μια ακολουθία κόμβων, όπου πρώτος είναι ο N_x, τελευταίος ο N_y και κάθε άλλος κόμβος είναι παιδί του προηγουμένου.	• Η διαδρομή από τον κόμβο N_1 στον κόμβο N_{12} είναι η N_1-N_3-N_8-N_{12}. ➢ Σε ένα δέντρο υπάρχει μια και μοναδική διαδρομή από τη ρίζα σε οποιονδήποτε κόμβο του.
Μήκος (*length*) μιας διαδρομής είναι ο αριθμός των ακμών που περιλαμβάνονται σ' αυτήν.	• Το μήκος της παραπάνω διαδρομής είναι τρία (3), διότι περιλαμβάνει τρεις ακμές.
Επίπεδο (*level*) ή *βάθος* (*depth*) ενός κόμβου είναι το μήκος της μοναδικής διαδρομής από τη ρίζα στον κόμβο αυτό.	• Το επίπεδο (βάθος) του κόμβου N_8 είναι δύο (2), διότι η διαδρομή από τη ρίζα στον κόμβο έχει μήκος δύο (2). ➢ Οι κόμβοι ενός δέντρου είναι οργανωμένοι σε επίπεδα. Η ρίζα έχει βάθος μηδέν (0), δηλαδή βρίσκεται στο επίπεδο μηδέν (0).
Ύψος (*height*) ενός δέντρου είναι το μέγιστο βάθος των (τερματικών) κόμβων του.	• Το ύψος του δέντρου του σχήματος είναι τρία (3), διότι αυτό είναι το μέγιστο βάθος των κόμβων του. ➢ Το ύψος του κενού δέντρου είναι –1.
Βαθμός (*degree*) ενός *κόμβου* είναι ο αριθμός των παιδιών του.	• Ο βαθμός του κόμβου N_2 είναι τρία (3), ενώ του N_3 είναι ένα (1). ➢ Ο βαθμός ενός τερματικού κόμβου είναι μηδέν (0).

Βαθμός ενός *δέντρου* είναι ο μέγιστος των βαθμών των κόμβων του.

• Ο βαθμός του δέντρου του σχήματος είναι τρία (3).

➢ Δέντρα βαθμού *n* ονομάζονται *n*–δικά δέντρα. Έτσι έχουμε δυαδικά (binary), τριαδικά (ternary) και τετραδικά (quadtree) δέντρα.

5.2 Δυαδικά δέντρα

5.2.1 Ορισμοί και αναπαράσταση

Ένα δυαδικό δέντρο είναι ένα δέντρο με βαθμό δύο (2) για κάθε κόμβο του. Για τον ορισμό ενός δυαδικού δέντρου χρησιμοποιείται ο διπλανός αναδρομικός ορισμός [1]. ■

■ Ένα **δυαδικό δέντρο** είναι είτε το κενό δέντρο είτε αποτελείται από ένα στοιχείο (κόμβο), που ονομάζεται *ρίζα* του δέντρου, και από δύο υποδέντρα, που ονομάζονται *αριστερό* και *δεξιό υποδέντρο*, με στοιχεία (κόμβους) του ίδιου τύπου με τη ρίζα, που είναι δυαδικά δέντρα.

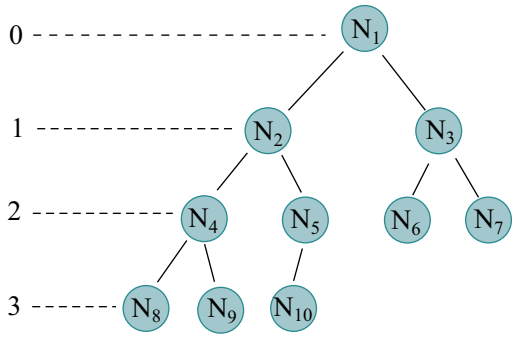
Δηλαδή κάθε κόμβος σ' ένα δυαδικό δέντρο έχει, κατά μέγιστο, δύο παιδιά, το *αριστερό παιδί* και το *δεξιό παιδί*, που είναι αντίστοιχα οι ρίζες του αριστερού και δεξιού υποδέντρου του κόμβου. Αν κάποιο από τα δύο παιδιά είναι το κενό δέντρο, τότε λέμε ότι το αντίστοιχο παιδί δεν υπάρχει ή είναι κενό.

Ένα δυαδικό δέντρο απεικονίζεται στο Σχήμα 5.2. Το αριστερό παιδί του κόμβου N_1 είναι ο κόμβος N_2 και το δεξιό παιδί ο κόμβος N_3. Το αριστερό παιδί του N_5 είναι ο N_{10}, αλλά το δεξιό του παιδί είναι το κενό δέντρο. Οι τερματικοί κόμβοι έχουν ως αριστερό και δεξιό παιδί τους το κενό δέντρο.

Στο επίπεδο 0 (ρίζα) ο μέγιστος αριθμός κόμβων είναι $2^0 = 1$, στο επίπεδο 1 είναι $2^1 = 2$, στο επίπεδο 2 είναι $2^2 = 4$ κ.ο.κ., στο επίπεδο (βάθος) k είναι 2^k. Επομένως, ένα δυαδικό δέντρο ύψους h μπορεί να έχει κατά μέγιστο $2^0 + 2^1 + 2^2 + \ldots + 2^h = \sum_{k=0}^{h} 2^k = 2^{h+1} - 1$ κόμβους [2].

[1] Αναδρομικός λέγεται ένας ορισμός που χρησιμοποιεί την οριζόμενη έννοια για τον ορισμό της.

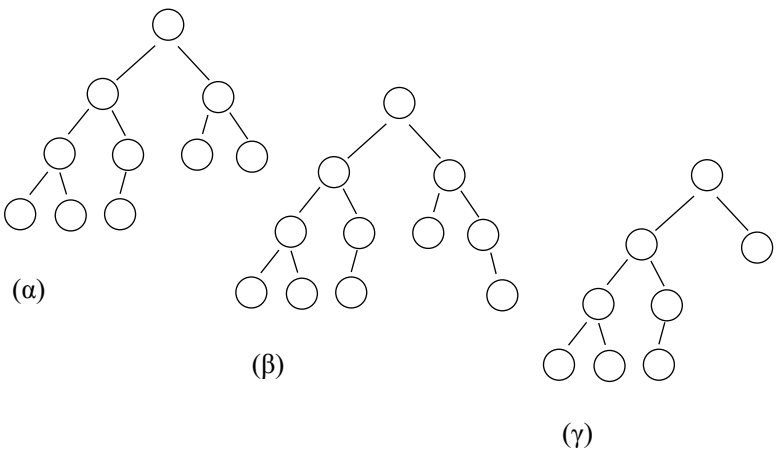
[2] Προκύπτει από τη μαθηματική ταυτότητα $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$.



Σχήμα 5.2

Ένα δυαδικό δέντρο

Ένα δυαδικό δέντρο λέγεται *πλήρες* (complete), αν σε όλα τα επίπεδά του, εκτός ίσως από το τελευταίο, έχει το μέγιστο δυνατό αριθμό κόμβων και όλοι οι κόμβοι στο τελευταίο επίπεδο βρίσκονται όσο το δυνατό στα αριστερά του δέντρου. Επομένως, υπάρχει ένα μοναδικό πλήρες δέντρο με ακριβώς *n* κόμβους. Π.χ. από τα δέντρα του Σχήματος 5.3 μόνο το (α) είναι πλήρες. Το (β) δεν είναι πλήρες, διότι οι κόμβοι στο τελευταίο επίπεδο (τερματικοί) δε βρίσκονται όσο το δυνατόν αριστερότερα. Το (γ) δεν είναι επίσης πλήρες, διότι στο δεύτερο επίπεδο, που δεν είναι το τελευταίο, δεν έχει το μέγιστο δυνατό αριθμό κόμβων.



Σχήμα 5.3

Δυαδικά δέντρα: (α) πλήρες, (β) και (γ) μη πλήρη

Στη βιβλιογραφία, ορισμένες φορές αναφέρονται σ' αυτό τον τύπο δέντρου ως *σχεδόν πλήρες* (*almost complete*) δέντρο και στο δέντρο

που έχει συμπληρωμένα όλα τα επίπεδα ως πλήρες δέντρο. Εδώ δε θα ακολουθήσουμε αυτή τη διάκριση.

Για το ύψος h ενός πλήρους δέντρου n κόμβων ισχύει (πάνω όριο) η σχέση:

$$h \leq \log n.$$

Η απόδειξη έχει ως ακολούθως. Ο ελάχιστος αριθμός κόμβων σε ένα πλήρες δέντρο ύψους h είναι

$$(2^0+2^1+2^2+ \ldots +2^{h-1}) + 1,$$

δηλαδή συμπληρωμένα τα επίπεδα μέχρι ύψος $(h\text{-}1)$ συν έναν κόμβο από το τελευταίο επίπεδο. Αλλά $(2^0+2^1+2^2+ \ldots +2^{h-1}) = 2^h\text{-}1$, οπότε ο ελάχιστος αριθμός κόμβων γίνεται τελικά $(2^h\text{-}1) + 1 = 2^h$. Δηλαδή για ένα δέντρο n στοιχείων, είναι $n \geq 2^h \Leftrightarrow \log n \geq \log(2^h) \Leftrightarrow h \leq \log n$.

Επίσης, ισχύει (κάτω όριο) η σχέση:

$$h \geq \log(n+1)\text{-}1.$$

Η απόδειξη έχει ως ακολούθως: Αφού ο μέγιστος αριθμός κόμβων ενός πλήρους δέντρου ύψους h είναι $2^{h+1}\text{-}1$, για ένα δέντρο n στοιχείων θα ισχύει $n \leq 2^{h+1} - 1 \Leftrightarrow n + 1 \leq 2^{h+1} \Leftrightarrow \log(n + 1) \leq \log(2^{h+1})$ $\Leftrightarrow h + 1 \geq \log(n + 1) \Leftrightarrow h \geq \log(n + 1) - 1$.

Όπως και οι προηγούμενες δομές, έτσι και ένα δυαδικό δέντρο μπορεί να αναπαρασταθεί χρησιμοποιώντας είτε συνεχόμενη είτε συνδεδεμένη αναπαράσταση. Κοινές απαιτήσεις και στις δύο περιπτώσεις είναι α) να υπάρχει άμεση προσπέλαση της ρίζας του δέντρου και β) δεδομένου ενός κόμβου να υπάρχει άμεση προσπέλαση στα παιδιά του.

Συνεχόμενη Αναπαράσταση

Στη συνεχόμενη αναπαράσταση χρησιμοποιούμε έναν πίνακα, έστω T, για την αναπαράσταση ενός δυαδικού δέντρου. Για να είναι επαρκής ο πίνακας T για ένα δυαδικό δέντρο ύψους h πρέπει να έχει $n = (2^{h+1} – 1)$ στοιχεία, όσα δηλαδή και ο μέγιστος αριθμός στοιχείων του δέντρου. Για ένα πλήρες δυαδικό δέντρο, η αποθήκευση των κόμβων του στον πίνακα γίνεται όπως περιγράφεται στη συνέχεια.

Η ρίζα του δέντρου αποθηκεύεται στο πρώτο στοιχείο του πίνακα (T_1) και οι υπόλοιποι κόμβοι (στοιχεία) αποθηκεύονται στα επόμενα στοι-

χεία του πίνακα, με τη σειρά ανά επίπεδο, ξεκινώντας από τον πρώτο αριστερά κόμβο κάθε επιπέδου. Στην περίπτωση ενός πλήρους δυαδικού δέντρου που έχει αποθηκευτεί με αυτό τον τρόπο, για κάθε κόμβο που βρίσκεται στη θέση i του πίνακα, δηλαδή παριστάνεται από το στοιχείο T_i , ισχύουν τα εξής:

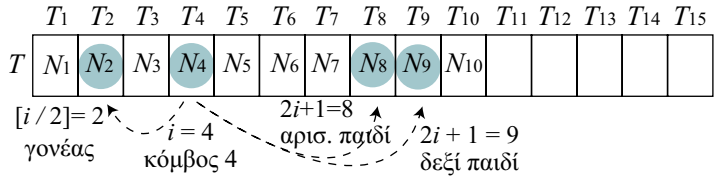
(1) Αν $2i > n$, όπου n ο αριθμός των πραγματικών κόμβων του δέντρου, τότε το T_i είναι τερματικός κόμβος (φύλλο) του δέντρου.

(2) Ο γονέας του T_i είναι το $T_{[i/2]}$, αν $i > 1$, όπου '[ ]' σημαίνει ακέραιο μέρος. Αν $i = 1$, πρόκειται για τη ρίζα του δέντρου.

(3) Το αριστερό παιδί του T_i είναι το T_{2i}, αν $2i \leq n$, αλλιώς δεν έχει αριστερό παιδί.

(4) Το δεξιό παιδί του T_i είναι το T_{2i+1}, αν $2i + 1 \leq n$, αλλιώς δεν έχει δεξιό παιδί.

Στο Σχήμα 5.4 φαίνεται η συνεχόμενη αναπαράσταση του πλήρους δυαδικού δέντρου του Σχήματος 5.2, όπου σημειώνονται ο γονέας και τα παιδιά του κόμβου N_4. Στη συνέχεια θα παραλείπουμε τις κενές θέσεις του πίνακα στη συνεχόμενη αναπαράσταση ενός δέντρου, για απλούστερη απεικόνιση.



Σχήμα 5.4

Συνεχόμενη αναπαράσταση δυαδικού δέντρου

Η συνεχόμενη αναπαράσταση μπορεί να εφαρμοστεί και σε μη πλήρες δέντρο. Τότε, βέβαια, θα υπάρχουν ενδιάμεσες κενές θέσεις στον πίνακα αναπαράστασης.

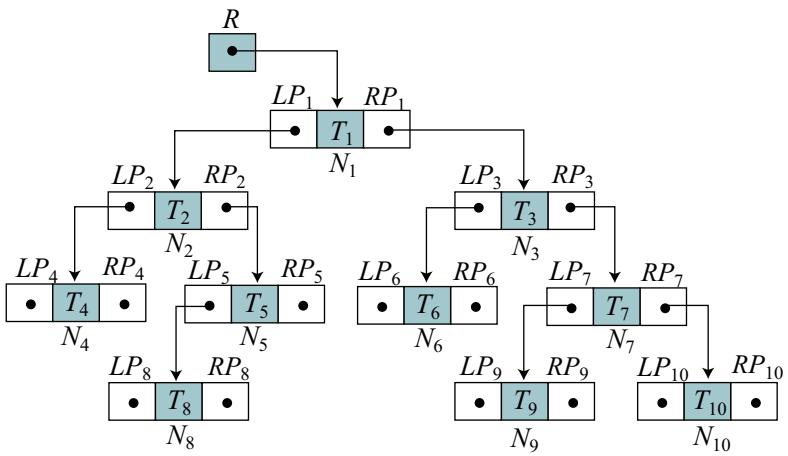
Συνδεδεμένη Αναπαράσταση

Στη συνδεδεμένη αναπαράσταση κάθε κόμβος N_i αποτελείται από τρία μέρη. Το αριστερό μέρος LP_i είναι ένας δείκτης στο αριστερό παιδί του κόμβου, που είναι η ρίζα του αριστερού υποδέντρου, ενώ το δεξιό μέρος RP_i είναι ένας δείκτης στο δεξιό παιδί του κόμβου, που είναι η

ρίζα του δεξιού υποδέντρου. Στο Σχήμα 5.5 φαίνεται η σχηματική παράσταση της συνδεδεμένης αναπαράστασης ενός δυαδικού δέντρου. Ο δείκτης *R* «δείχνει» στη ρίζα του δέντρου και ονομάζεται *δείκτης ρίζας* (*root pointer*).

Αν κάποιος κόμβος δεν έχει αριστερό ή και δεξιό παιδί (υποδέντρο), τότε ο αντίστοιχος δείκτης έχει τιμή NIL, δηλαδή είναι ο κενός δείκτης. Σ' ένα κενό δέντρο ο δείκτης *R* είναι ο κενός δείκτης.

Ο ορισμός ενός συνδεδεμένου δυαδικού δέντρου στην Pascal και την C γίνεται όπως και αυτός μιας απλής συνδεδεμένης λίστας (Παραδείγματα 3.2α και 3.2β), με τη διαφορά ότι η εγγραφή που παριστάνει έναν κόμβο έχει δύο πεδία τύπου δείκτη (π.χ. ldeiktis, rdeiktis) αντί για ένα (deiktis), και ο δείκτης ρίζας ορίζεται ως μια μεταβλητή δείκτη *R*.



Σχήμα 5.5

Συνδεδεμένη αναπαράσταση δυαδικού δέντρου

5.2.2 Διαπεράσεις

Διαπέραση ενός δυαδικού δέντρου σημαίνει την επίσκεψη κάθε κόμβου του μια φορά και κατά μια συγκεκριμένη σειρά. Διακρίνουμε τρεις βασικούς τρόπους διαπέρασης ενός δυαδικού δέντρου, την *προδιατεταγμένη* (*preorder*), την *ενδοδιατεταγμένη* (*inorder*) και τη *μεταδιατεταγμένη* (*postorder*) διαπέραση. Οι τρεις αυτοί τρόποι διαφέρουν στο ότι εκτελούν σ' ένα δέντρο τις διαδικασίες

(1) επίσκεψη ρίζας,

(2) επίσκεψη αριστερού υποδέντρου,

(3) επίσκεψη δεξιού υποδέντρου,

με διαφορετική σειρά, η οποία ακολουθείται και στη διαπέραση των υποδέντρων του δέντρου κ.ο.κ.

Έτσι, η προδιατεταγμένη διαπέραση ακολουθεί την παραπάνω σειρά (1)-(2)-(3), η ενδοδιατεταγμένη διαπέραση την (2)-(1)-(3) και η μεταδιατεταγμένη τη σειρά (2)-(3)-(1). Παρατηρήστε ότι το βασικό χαρακτηριστικό που διακρίνει τους τρείς τρόπους διαπέρασης είναι η σειρά επίσκεψης της ρίζας. Δηλαδή στην προδιατεταγμένη διαπέραση η επίσκεψη της ρίζας γίνεται πριν από αυτές των δύο υποδέντρων, στην ενδοδιατεταγμένη γίνεται μεταξύ των επισκέψεων στα δύο υποδέντρα και στη μεταδιατεταγμένη μετά τις επισκέψεις των υποδέντρων. Επίσης, παρατηρήστε ότι η επίσκεψη στο αριστερό υποδέντρο προηγείται πάντα της επίσκεψης στο δεξί. Για παράδειγμα, κατά την εφαρμογή της προδιατεταγμένης διαπέρασης στο δέντρο του Σχήματος 5.2, οι κόμβοι διαπερνώνται με την εξής σειρά N_1, N_2, N_3, …, N_{10}, ενώ στη μεταδιατεταγμένη διαπέραση με τη σειρά N_8, N_9, N_4, N_{10}, N_5, N_2, N_6, N_7, N_3, N_1.

Η υλοποίηση της διαπέρασης ενός δέντρου μπορεί να γίνει με διάφορες τεχνικές. Δύο βασικές τεχνικές είναι α) με χρήση *στοίβας* και β) με χρήση *αναδρομής* (*recursion*). Η αναδρομή είναι μια γενική τεχνική, με την οποία όμως δεν ασχολούμαστε σ' αυτό το βιβλίο. Επίσης, χρησιμοποιούμε συνδεμένη αναπαράσταση του δυαδικού δέντρου, που είναι πιο κατάλληλη, από ό,τι η συνεχόμενη, για την υλοποίηση των τρόπων διαπέρασης. Στη συνέχεια παρουσιάζουμε τον αλγόριθμο (Αλγόριθμος 5.1) για την υλοποίηση της προδιατεταγμένης διαπέρασης ενός δυαδικού δέντρου με χρήση μιας συνεχόμενης στοίβας.

Η βασική διαδικασία έχει ως εξής:

(α) Προχωρούμε προς τα κάτω ακολουθώντας πρώτα την πιο αριστερά ευρισκόμενη διαδρομή του δέντρου με ρίζα τη δοθείσα, εφαρμόζοντας την προτιθέμενη επεξεργασία στο στοιχείο κάθε κόμβου που συναντάμε στη διαδρομή και τοποθετούμε κάθε δεξιό παιδί, όπου υπάρχει, στη στοίβα. Σταματούμε όταν συναντήσουμε κόμβο χωρίς αριστερό παιδί.

(β) Εξάγουμε το κορυφαίο στοιχείο της στοίβας και επαναλαμβάνουμε το (α) θεωρώντας το στοιχείο αυτό ως ρίζα.

Στον Αλγόριθμο 5.1 (T-PROD-DIAPERASH) χρησιμοποιούμε μια συνεχόμενη στοίβα (S, T) και μια μεταβλητή δείκτη (P) ως βοηθητικά στοιχεία. Η στοίβα χρησιμοποιείται για την αποθήκευση των δεικτών των (δεξιών) υποδέντρων, ενώ η P για την αποθήκευση του τρέχοντος δείκτη του (αριστερού) υποδέντρου. Επίσης, το STOIXEIO(KOMBOS(P)) παριστάνει το στοιχείο του κόμβου, και τα ADEIKTHS(KOMBOS(P)), DDEIKTHS(KOMBOS(P)) τον αριστερό και δεξιό δείκτη του κόμβου στον οποίο δείχνει ο δείκτης P.

ΑΛΓΟΡΙΘΜΟΣ 5.1: ΠΡΟΔΙΑΤΕΤΑΓΜΕΝΗ ΔΙΑΠΕΡΑΣΗ

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνονται μεταβολές στις τιμές των στοιχείων του δέντρου.

```
T-PROD-DIAPERASH (R)
1    T ← 0, P ← R {Αρχικοποιήσεις}
2    while P ≠ NIL                                {Συνθήκη επανάληψης}
3      PROCESS(STOIXEIO(KOMBOS(P)))               {Εφαρμογή επεξεργασίας}
4      if DDEIKTHS(KOMBOS(P)) ≠ NIL               {Έλεγχος δεξιού υποδέντρου}
5        then T ← T + 1                           {Ενημέρωση δείκτη κορυφής}
6             S[T] ← DDEIKTHS(KOMBOS(P))          {Καταχώριση στη στοίβα}
       endif
7      if ADEIKTHS(KOMBOS(P)) ≠ NIL               {Έλεγχος αριστερού υποδέντρου}
8        then P ← ADEIKTHS(KOMBOS(P))             {Ενημέρωση δείκτη}
9        else P ← S[T]                            {Εξαγωγή κορυφής στοίβας}
10            T ← T - 1                           {Ενημέρωση δείκτη κορυφής}
       endif
     endwhile
```

Στο βήμα 1 αρχικοποιείται ο δείκτης κορυφής της στοίβας (άδεια στοίβα) και ο δείκτης P (στην τιμή του δείκτη ρίζας R). Ο κυρίως αλγόριθμος (βήματα 2-10) είναι μια επαναληπτική διάταξη while. Καταρχήν (βήμα 3) εφαρμόζεται η διαδικασία PROCESS στο στοιχείο του τρέχοντος κόμβου (δηλαδή της ρίζας) του τρέχοντος (υπο)δέντρου.

Στη συνέχεια αποθηκεύεται στη στοίβα ο δείκτης του τρέχοντος κόμβου που δείχνει στο δεξιό υποδέντρο του, αν υπάρχει (διάταξη if, βήματα 4-6). Κατόπιν ο τρέχων δείκτης παίρνει την τιμή του δείκτη στο αριστερό υποδέντρο του τρέχοντος κόμβου, αν υπάρχει, ή την τιμή της κορυφής της στοίβας – (δηλαδή του δείκτη σε κάποιο δεξιό υποδέντρο που αποθηκεύτηκε πιο πρόσφατα) – (διάταξη if, βήματα 7-10). Η επανάληψη σταματά όταν ο τρέχων δείκτης P πάρει την τιμή NIL, δηλαδή όταν δεν υπάρχουν άλλοι κόμβοι για επίσκεψη.

Άσκηση Αυτοαξιολόγησης 5.1

Ένα δυαδικό δέντρο αναζήτησης έχει εννέα (9) κόμβους. Η προδιατεταγμένη διαπέραση έχει ως αποτέλεσμα την επίσκεψη των στοιχείων (κόμβων) του με τη σειρά Z, A, E, K, Γ, Δ, H, Θ, B, ενώ η ενδοδιατεταγμένη διαπέραση με τη σειρά E, A, Γ, K, Z, H, Δ, B, Θ. Να σχεδιάσετε το δυαδικό δέντρο.

5.3 Δυαδικά δεντρα αναζήτησης

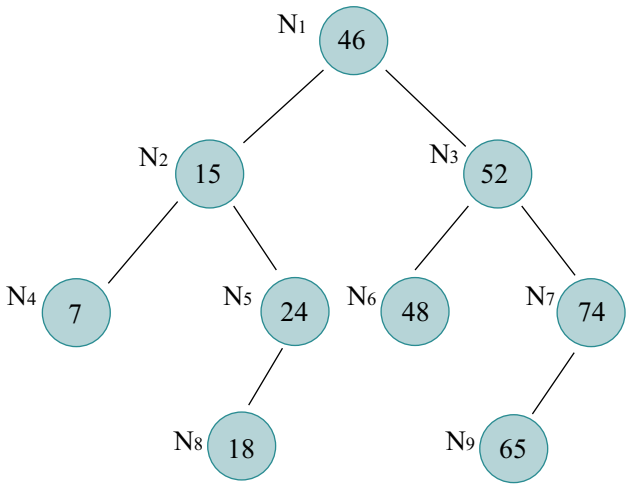
5.3.1 Ορισμός

Το *δυαδικό δέντρο αναζήτησης* (binary search tree) ή *διατεταγμένο δυαδικό δέντρο* (binary sorted tree) είναι μια από τις σπουδαιότερες δομές δεδομένων στην επιστήμη των υπολογιστών, διότι συνδυάζει μικρό χρόνο αναζήτησης και εύκολη εισαγωγή και διαγραφή στοιχείων. Ο τύπος των στοιχείων των κόμβων ενός δυαδικού δέντρου αναζήτησης πρέπει να είναι τέτοιου τύπου που να επιτρέπει τη διάταξη των στοιχείων (π.χ. ακέραιοι αριθμοί, χαρακτήρες). ■

■ Ένα **δυαδικό δέντρο αναζήτησης** είναι ένα δυαδικό δέντρο με διατεταγμένους κόμβους έτσι ώστε η τιμή του στοιχείου κάθε κόμβου να είναι μεγαλύτερη από τις τιμές όλων των στοιχείων των κόμβων του αριστερού του υποδέντρου και μικρότερη από αυτές των στοιχείων του δεξιού του υποδέντρου.

Για παράδειγμα, το δέντρο του Σχήματος 5.6 είναι ένα δυαδικό δέντρο αναζήτησης που τα στοιχεία του είναι ακέραιοι αριθμοί. Παρατηρήστε ότι η τιμή της ρίζας του δέντρου (46) είναι μεγαλύτερη από την τιμή κάθε στοιχείου του αριστερού υποδέντρου (15, 7, 24, 18) και μικρότερη από την τιμή κάθε στοιχείου του δεξιού υποδέντρου (52, 48, 74, 65). Το ίδιο συμβαίνει και με κάθε εσωτερικό κόμβο. Π.χ. η τιμή του στοιχείου του κόμβου N_2 (15) είναι μεγαλύτερη από αυτές του αριστερού υποδέντρου (7) και μικρότερη από αυτές του δεξιού υποδέντρου (24, 18).

Όπως εύκολα μπορεί να διαπιστώσει κάποιος, η ενδοδιατεταγμένη διαπέραση ενός δυαδικού δέντρου αναζήτησης επισκέπτεται τα στοιχεία (κόμβους) του δέντρου με τη σειρά που θα είχαν αν τα διατάσσαμε κατ' αύξουσα τάξη (7, 15, 18, 24, 46, 48, 52, 65, 74, για το δέντρο του Σχήματος 5.6).



Σχήμα 5.6

Δυαδικό δέντρο αναζήτησης με ακέραια στοιχεία

Ας σημειωθεί ότι, στην πράξη, οι κόμβοι ενός δέντρου είναι συνήθως εγγραφές, δηλαδή έχουμε μια μεικτή δομή, που ονομάζεται *δέντρο εγγραφών*. Στην περίπτωση αυτή, αυτό το οποίο αναφερόμαστε εδώ ως το 'στοιχείο' ενός κόμβου είναι το 'κλειδί' των εγγραφών (ανατρέξτε στην Ενότητα «2.3.2 Πίνακες Εγγραφών» για τον ορισμό της έννοιας 'κλειδί').

5.3.2 Αναζήτηση

Μια από τις βασικές πράξεις σ' ένα δυαδικό δέντρο αναζήτησης είναι η αναζήτηση ενός στοιχείου. Η βασική διαδικασία αναζήτησης έχει ως εξής:

(1) Συγκρίνουμε τη ρίζα του δέντρου με το υπό αναζήτηση στοιχείο και

(α) αν η τιμή του στοιχείου είναι μικρότερη, τότε προχωρούμε στο αριστερό παιδί της ρίζας,

(β) αν είναι μεγαλύτερη, τότε προχωρούμε στο δεξιό παιδί της ρίζας.

(2) Το βήμα (1) επαναλαμβάνεται μέχρις ότου

(α) συναντήσουμε το προς αναζήτηση στοιχείο (επιτυχία) ή

(β) συναντήσουμε το κενό (υπο)δέντρο (αποτυχία).

Ο βασικός αλγόριθμος αναζήτησης παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 5.2α (T-BASIKH-ANAZHTHSH). Τον ονομάζουμε 'βασικό' αλγόριθμο, διότι χρησιμοποιείται ως βάση για τους επόμενους αλγορίθμους. Ένας από αυτούς είναι ο ολοκληρωμένος αλγόριθμος αναζήτησης, που παρουσιάζεται στη συνέχεια ως Αλγόριθμος 5.2.

ΑΛΓΟΡΙΘΜΟΣ 5.2α: ΒΑΣΙΚΗ ΑΝΑΖΗΤΗΣΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου αναζήτησης και ένας δείκτης (P) στον προς αναζήτηση κόμβο (στοιχείο).

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται ενημέρωση κάποιων μεταβλητών.

```
T-BASIKH-ANAZHTHSH(R, P)
1    PX ← R, PXG ← NIL, EURESH ← FALSE          {Αρχικοποίηση μεταβλητών}
2    X ← STOIXEIO(KOMBOS(P))                    {Προς αναζήτηση στοιχείο}
3    while (not EURESH) and (PX ≠ NIL)          {Συνθήκη τερματισμού}
4       if X < STOIXEIO(KOMBOS(PX))             {Σύγκριση στοιχείων}
5          then PXG ← PX                        {Ενημέρωση δείκτη}
6               PX ← ADEIKTHS(KOMBOS(PX))       {Ενημέρωση δείκτη}
7          else if X > STOIXEIO(KOMBOS(PX))     {Σύγκριση στοιχείων}
8                  then PXG ← PX                {Ενημέρωση δείκτη}
9                       PX ← DDEIKTHS(KOMBOS(PX))  {Ενημέρωση δείκτη}
10                 else EURESH ← TRUE           {Ενημέρωση μεταβλητής}
                   endif
       endif
    endwhile
```

Στον Αλγόριθμο 5.2α χρησιμοποιούνται τέσσερις βοηθητικές μεταβλητές: PX, PXG, EURESH και X. Η PX είναι ένας δείκτης που δείχνει πάντα στο τρέχον στοιχείο (κόμβο) του δέντρου. Η PXG είναι ένας δείκτης που δείχνει πάντα στο γονέα τού τρέχοντος στοιχείου (κόμβου). Η EURESH είναι μια λογική μεταβλητή που γίνεται αλη-

θής, όταν το προς αναζήτηση στοιχείο βρεθεί, και τέλος η Χ αντιπροσωπεύει το τρέχον στοιχείο.

Καταρχήν γίνονται αρχικοποιήσεις των βοηθητικών μεταβλητών (βήματα 1, 2). Στη συνέχεια ξεκινά μια επαναληπτική διαδικασία μέσω μιας διάταξης while (βήματα 3-10) που υλοποιεί τη διαδικασία της αναζήτησης, όπως αναφέρθηκε πιο πάνω. Η επανάληψη τερματίζεται, όταν συναντήσουμε το κενό (υπο)δέντρο (δηλαδή γίνει PX = NIL, βήμα 6 ή βήμα 9) ή βρεθεί το στοιχείο (δηλαδή γίνει EURESH = TRUE, βήμα 10). Μετά το τέλος της εκτέλεσης του αλγορίθμου, αν έχει βρεθεί το στοιχείο, η μεταβλητή EURESH έχει την τιμή 'TRUE', ο δείκτης PX δείχνει στον κόμβο του στοιχείου και ο δείκτης PXG στο γονέα του, εκτός αν ο κόμβος είναι η ρίζα, οπότε PXG = NIL. Αν δεν έχει βρεθεί, η EURESH έχει τιμή FALSE, ο PX τιμή NIL και ο PXG «δείχνει» τον κόμβο όπου σταμάτησε η αναζήτηση.

ΑΛΓΟΡΙΘΜΟΣ 5.2: ΑΝΑΖΗΤΗΣΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου αναζήτησης και ένας δείκτης (P) στον κόμβο τού προς αναζήτηση στοιχείου.

Έξοδος: Εξωτερικά επιστρέφει, αντίστοιχα, μήνυμα για τις περιπτώσεις εύρεσης και μη εύρεσης του στοιχείου.

```
T-ANAZHTHSH(R, P)
1    T-BASIKH-ANAZHTHSH(R, P)          {Βασική αναζήτηση}
2    if EURESH                         {Έλεγχος αποτελέσματος}
3      then print 'ΤΟ ΣΤΟΙΧΕΙΟ ΒΡΕΘΗΚΕ'   {Μήνυμα εύρεσης στοιχείου}
4      else print 'ΤΟ ΣΤΟΙΧΕΙΟ ΔΕ ΒΡΕΘΗΚΕ'   {Μήνυμα αποτυχίας}
     endif
```

Στον ολοκληρωμένο αλγόριθμο αναζήτησης, Αλγόριθμος 5.2 (T-ANAZHTHSH), εκτελείται καταρχήν η βασική αναζήτηση (βήμα 1). Ο βασικός αλγόριθμος αναζήτησης χρησιμοποιείται εδώ ως μερικός αλγόριθμος τύπου διαδικασίας (ανατρέξτε στην Υποενότητα «1.2.2 Περιγραφή της Γλώσσας» για τους τύπους μερικών αλγόριθμων). Σε τέτοιες περιπτώσεις θεωρούμε ότι οι μεταβλητές του μερικού αλγορίθμου είναι και μεταβλητές του κυρίως αλγορίθμου. Δηλαδή οι τιμές που παίρνουν οι μεταβλητές αυτές κατά την εκτέλεση του μερικού αλγορίθμου διατηρούνται και μετά το πέρας της εκτέλεσής του και

μπορούν να χρησιμοποιηθούν στον κυρίως αλγόριθμο.

Στη συνέχεια γίνεται έλεγχος του αποτελέσματος (διάταξη if, βήματα 2-4) και εκτυπώνεται το αντίστοιχο μήνυμα εύρεσης (βήμα 3) ή αποτυχίας (βήμα 4). Η τιμή της μεταβλητής EURESH έχει προκύψει από την εφαρμογή του μερικού αλγορίθμου στο βήμα 1.

5.3.3 Εισαγωγή

Η διαδικασία για την εισαγωγή ενός στοιχείου είναι η ίδια με αυτή της αναζήτησης, με τις ακόλουθες αλλαγές. Πρώτον, χρησιμοποιούμε τη φράση «προς εισαγωγή» αντί της «υπό αναζήτηση». Δεύτερον, στην περίπτωση (2α) απλώς διαπιστώνεται η ύπαρξη του στοιχείου και δεν έχει νόημα η εισαγωγή του. Τρίτον, στην περίπτωση (2β), δηλαδή όταν συναντήσουμε κενό υποδέντρο, δεν επιστρέφει αποτυχία, αλλά τότε γίνεται η εισαγωγή του στοιχείου ως ρίζα του υποδέντρου. Ο αλγόριθμος για την εισαγωγή στοιχείου σ' ένα συνδεδεμένο δυαδικό δέντρο αναζήτησης παρουσιάζεται στο παρακάτω πλαίσιο ως Αλγόριθμος 5.3 (T-EISAGWGH).

Στον Αλγόριθμο 5.3, καταρχήν (βήμα 1), καλείται ο μερικός αλγόριθμος βασικής αναζήτησης (Αλγόριθμος 5.2α). Στη συνέχεια χρησιμοποιείται μια σύνθετη διάταξη if (βήματα 2-8) για την ανίχνευση των τριών δυνατών περιπτώσεων που μπορεί να συναντήσουμε. Αν το δοθέν δέντρο είναι το κενό δέντρο (1η περίπτωση, βήμα 2), τότε ο κόμβος του στοιχείου γίνεται η ρίζα του δέντρου (βήμα 3). Αλλιώς, αν το στοιχείο δεν υπάρχει (2η περίπτωση, βήμα 4), τότε γίνεται εισαγωγή του στοιχείου ως ρίζας του αντίστοιχου (αριστερού ή δεξιού) κενού υποδέντρου που ανιχνεύτηκε με το βασικό αλγόριθμο αναζήτησης (βήματα 5-7). Τέλος, αν το στοιχείο υπάρχει (3η περίπτωση), δεν τίθεται θέμα εισαγωγής του, αλλά εκτυπώνεται αντίστοιχο μήνυμα (βήμα 8). Για παράδειγμα, η εφαρμογή του Αλγορίθμου 5.3 για την εισαγωγή του στοιχείου '22' στο δυαδικό δέντρο του Σχήματος 5.6 θα έχει ως αποτέλεσμα την τοποθέτηση του στοιχείου ως δεξιού παιδιού του κόμβου N_8, αφού ακολουθηθεί η διαδρομή N_1-N_2-N_5-N_8.

ΑΛΓΟΡΙΘΜΟΣ 5.3: ΕΙΣΑΓΩΓΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου αναζήτησης και ένας δείκτης (P) στον κόμβο τού προς εισαγωγή στοιχείου.

Έξοδος: Εξωτερικά επιστρέφει αντίστοιχο μήνυμα σε περίπτωση ύπαρξης του στοιχείου, αλλιώς τίποτα. Εσωτερικά γίνεται εισαγωγή του κόμβου του στοιχείου στο δέντρο.

```
T-EISAGWGH(R, P)
1    T-BASIKH-ANAZHTHSH(R, P)                          {Βασική αναζήτηση}
2    if (PX = NIL) and (PI = NIL)                      {Έλεγχος κενού δέντρου}
3      then R ← P                                      {Εισαγωγή στη ρίζα}
4      else if EURESH = FALSE                          {Έλεγχος εύρεσης στοιχείου}
5             then if X < STOIXEIO(KOMBOS(PI))         {Έλεγχος θέσης εισαγωγής}
6                    then ADEIKTHS(KOMBOS(PI)) ← P     {Εισαγωγή κόμβου αριστερά}
7                    else DDEIKTHS(KOMBOS(PI)) ← P     {Εισαγωγή κόμβου δεξιά}
                  endif
8             else print 'ΤΟ ΣΤΟΙΧΕΙΟ ΥΠΑΡΧΕΙ'         {Μήνυμα ύπαρξης στοιχείου}
           endif
     endif
```

5.3.4 Διαγραφή

Η διαγραφή ενός στοιχείου από ένα δυαδικό δέντρο αναζήτησης είναι πιο πολύπλοκη από την εισαγωγή. Η αναζήτηση τού προς διαγραφή στοιχείου γίνεται όπως και στην περίπτωση της εισαγωγής. Αν δε βρεθεί το στοιχείο, τότε επιστρέφει μήνυμα αποτυχίας. Αν βρεθεί ο κόμβος N_x που έχει το στοιχείο, τότε ακολουθείται η εξής διαδικασία (τρεις περιπτώσεις):

(1) Αν ο N_x δεν έχει παιδιά, τότε διαγράφεται ο N_x και τη θέση του καταλαμβάνει το κενό δέντρο.

(2) Αν ο N_x έχει μόνο ένα παιδί, τότε διαγράφεται ο N_x και τη θέση του καταλαμβάνει το μοναδικό παιδί του.

(3) Αν ο N_x έχει δύο παιδιά, τότε

(α) βρίσκεται ο κόμβος που είναι ο αμέσως επόμενος από τον N_x στην, κατ' αύξουσα σειρά, ακολουθία των κόμβων (ενδοδιατεταγ-

μένη διαπέραση), έστω N_y, σύμφωνα με τα βήματα (1) και (2),

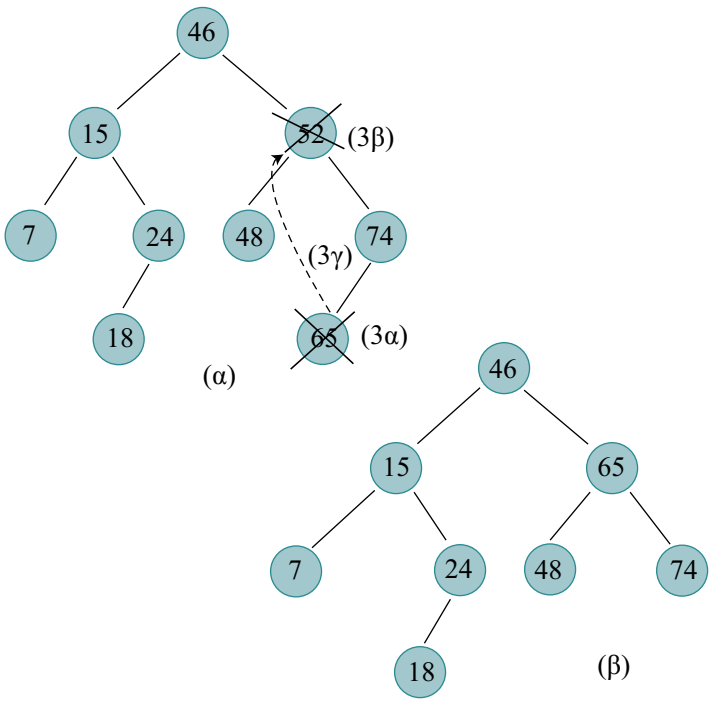
(β) διαγράφεται ο κόμβος N_x και

(γ) τη θέση του καταλαμβάνει ο N_y.

Πρέπει να σημειωθεί ότι, όπως είναι εύκολο να διαπιστωθεί, ο N_y (περίπτωση 3), δεν έχει ποτέ δύο παιδιά, ενώ όταν έχει ένα, αυτό είναι πάντα το δεξιό παιδί. Διότι, αν είχε δύο παιδιά, ή ένα και αριστερό, τότε το αριστερό παιδί του θα ήταν πριν από αυτόν, σύμφωνα με την ενδοδιατεταγμένη διαπέραση. Έτσι, στην περίπτωση (3α) δεν χρειάζεται ποτέ αναδρομική εφαρμογή του βήματος (3) για τη διαγραφή του N_y.

Παράδειγμα 5.1. *Διαγραφή στοιχείου (κόμβου) από δυαδικό δέντρο αναζήτησης*

Θεωρούμε το δυαδικό δέντρο αναζήτησης του Σχήματος 5.6, από το οποίο θέλουμε να διαγράψουμε το στοιχείο (κόμβο) '52'. Σημειώνεται ότι με '52' θα συμβολίζουμε το στοιχείο (κόμβο) και με 52 την τιμή του. Η διαδικασία διαγραφής του στοιχείου '52' απεικονίζεται στα παρακάτω σχήματα.



Στο Σχήμα (α) σημειώνονται τα βήματα της διαδικασίας διαγραφής. Καταρχήν ο επόμενος του '52' στη σειρά, δηλαδή ο '65', διαγράφεται (3α) και τη θέση του καταλαμβάνει το κενό δέντρο. Κατόπιν διαγράφεται και ο '52' (3β) και, τέλος, ο '65' καταλαμβάνει τη θέση του (3γ). Η τελική κατάσταση του δέντρου απεικονίζεται στο Σχήμα (β).

Για λόγους καλύτερης κατανόησης, θα παρουσιάσουμε τον αλγόριθμο διαγραφής με τη βοήθεια δύο μερικών αλγορίθμων. Ο πρώτος, που αναφέρεται στις παραπάνω περιπτώσεις (1) και (2), δηλαδή τις περιπτώσεις που ο προς διαγραφή κόμβος έχει το πολύ ένα παιδί, είναι ο Αλγόριθμος 5.4α (T-DIAGRAFH-A).

ΑΛΓΟΡΙΘΜΟΣ 5.4α: ΜΕΡΙΚΗ ΔΙΑΓΡΑΦΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ - Α

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου αναζήτησης, ένας δείκτης (P) στον προς διαγραφή κόμβο και ένας δείκτης (PG) στο γονέα του κόμβου.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται διαγραφή του κόμβου από το δέντρο.

```
T-DIAGRAFH-A (R, P, PG)

1    if (ADEIKTHS(KOMBOS(P)) = NIL) and
2      (DDEIKTHS(KOMBOS(P)) = NIL)                {Έλεγχος πλήθους παιδιών}
       then PZ ← NIL                              {Ενημέρωση δείκτη}
3      else if ADEIKTHS(KOMBOS(P)) = NIL          {Έλεγχος αριστερού στοιχείου}
4              then PZ ← DDEIKTHS(KOMBOS(P))      {Ενημέρωση δείκτη}
5              else PZ ← ADEIKTHS(KOMBOS(P))      {Ενημέρωση δείκτη}
           endif
     endif
6    if PG ≠ NIL                                  {Έλεγχος διαγραφής ρίζας}
7      then if P = ADEIKTHS(KOMBOS(PG))           {Έλεγχος αριστερού παιδιού}
8              then ADEIKTHS(KOMBOS(PG)) ← PZ     {Ενημέρωση αριστερού δείκτη}
9              else  DDEIKTHS(KOMBOS(PG)) ← PZ    {Ενημέρωση δεξιού δείκτη}
           endif
10     else R ← PZ                                {Ενημέρωση δείκτη ρίζας}
     endif
```

Στον αλγόριθμο αυτό χρησιμοποιούμε τη βοηθητική μεταβλητή PZ, που είναι τύπου δείκτη και αντιπροσωπεύει το δείκτη στον κόμβο που θα αντικαταστήσει τον προς διαγραφή. Ο αλγόριθμος αποτελείται από δύο σύνθετες διατάξεις if. Η πρώτη (βήματα 1-5) αποδίδει στην PZ την κατάλληλη τιμή, ανάλογα με το αν ο προς διαγραφή κόμβος δεν έχει καθόλου παιδιά (βήμα 2) ή αν έχει μόνο δεξί (βήμα 4) ή μόνο αριστερό (βήμα 5) παιδί. Η δεύτερη σύνθετη διάταξη if (βήματα 6-10) ενημερώνει τον αντίστοιχο δείκτη του γονέα τού προς διαγραφή κόμβου ώστε να δείχνει στον κόμβο που θα τον αντικαταστήσει, ανάλογα με το αν ο προς διαγραφή κόμβος είναι το αριστερό (βήμα 8) ή το δεξιό (βήμα 9) παιδί του ή είναι η ρίζα του δέντρου (βήμα 10).

Ο δεύτερος (μερικός) αλγόριθμος, που αναφέρεται στην περίπτωση (3), δηλαδή όταν ο προς διαγραφή κόμβος έχει δύο παιδιά, είναι ο Αλγόριθμος 5.4β (T-DIAGRAFH-B). Όπως φαίνεται, χρησιμοποιεί ως μερικό αλγόριθμο τον αλγόριθμο 5.4α.

Στον Αλγόριθμο 5.4β χρησιμοποιούμε δύο βοηθητικές μεταβλητές, τις PY και PYG, που είναι τύπου δείκτη και αντιπροσωπεύουν το δείκτη στον αμέσως επόμενο (στην ενδοδιατεταγμένη σειρά) από τον προς διαγραφή κόμβο και στο γονέα του αντίστοιχα. Ο Αλγόριθμος 5.4β αποτελείται από τρία τμήματα. Το πρώτο τμήμα (βήματα 1-5), που ως βάση έχει μια διάταξη while, ασχολείται με την εύρεση του επόμενου κόμβου και του γονέα του. Η ιδέα εδώ είναι ότι προχωρούμε, καταρχήν, στο δεξιό παιδί τού προς διαγραφή κόμβου (βήμα 2) και κατόπιν συνεχώς στα αριστερά παιδιά (βήμα 5), μέχρις ότου συναντήσουμε κόμβο με κενό αριστερό υποδέντρο (βήμα 3). Σε κάθε μετακίνηση κρατούμε τον προηγούμενο δείκτη στη μεταβλητή PYG (βήματα 1, 4), αφού αντιπροσωπεύει το δείκτη στο γονέα του τρέχοντος κόμβου.

Στο δεύτερο τμήμα γίνεται κλήση του πρώτου (μερικού) αλγορίθμου για τη διαγραφή του επόμενου κόμβου (PY). Προσέξτε ότι η κλήση του μερικού αλγορίθμου γίνεται με μεταβλητές εισόδου τις R, PY, και PYG, που είναι διαφορετικές από αυτές που χρησιμοποιούνται στον ορισμό του αλγορίθμου (R, P, PG). Στην περίπτωση αυτή οι R, PY, και PYG παίρνουν τη θέση των R, P, και PG κατά την εκτέλεση του αλγορίθμου.

ΑΛΓΟΡΙΘΜΟΣ 5.4β: ΜΕΡΙΚΗ ΔΙΑΓΡΑΦΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ - Β

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου αναζήτησης, ένας δείκτης (P) στον προς διαγραφή κόμβο και ένας δείκτης (PG) στο γονέα του κόμβου.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται διαγραφή του κόμβου από το δέντρο.

```
T-DIAGRAFH-B (R, P, PG)
1    PYG ← P                                          {Αρχικοποίηση δείκτη γονέα}
2    PY ← DDEIKTHS(KOMBOS(P))                         {Αρχικοποίηση δείκτη κόμβου}
3    while ADEIKTHS(KOMBOS(P)) ≠ NIL                  {Συνθήκη επανάληψης}
4      PYG ← PY                                       {Ενημέρωση δείκτη γονέα}
5      PY← ADEIKTHS(KOMBOS(PY))                       {Ενημέρωση δείκτη κόμβου}
     endwhile
6    T-DIAGRAFH-A (R, PY, PYG)                        {Κλήση μερικού αλγορίθμου}
7    if PG ≠ NIL                                      {Έλεγχος διαγραφής ρίζας}
8      then if P = ADEIKTHS(KOMBOS(PG))               {Έλεγχος αριστερού παιδιού}
9              then ADEIKTHS(KOMBOS(PG)) ← PY         {Ενημέρωση αριστερού δείκτη}
10             else DDEIKTHS(KOMBOS(PG)) ← PY         {Ενημέρωση δεξιού δείκτη}
           endif
       else R ← PY                                    {Ενημέρωση δείκτη ρίζας}
     endif
12   ADEIKTHS(KOMBOS(PY)) ← ADEIKTHS(KOMBOS(P))       {Ενημέρωση αριστερού δείκτη}
DDEIKTHS(KOMBOS(PY)) ← DDEIKTHS(KOMBOS(P))           {Ενημέρωση δεξιού δείκτη}
```

Στο τρίτο τμήμα γίνεται διαγραφή του κόμβου (P) και αντικατάστασή του με τον επόμενο (PY). Ως βάση χρησιμοποιείται μια διάταξη if (βήματα 7-11), όπου ο αντίστοιχος δείκτης του γονέα τού προς διαγραφή κόμβου ενημερώνεται ώστε να δείχνει στον επόμενο κόμβο (βήματα 9-10). Στην περίπτωση που ο προς διαγραφή κόμβος είναι η ρίζα του δέντρου (PG = NIL), τότε ενημερώνεται ο δείκτης ρίζας (βήμα 11). Τέλος, ενημερώνονται και οι δείκτες του επόμενου κόμβου ώστε να δείχνουν εκεί που έδειχναν οι αντίστοιχοι δείκτες του διαγραφέντος κόμβου (βήματα 12-13).

Ο ολοκληρωμένος αλγόριθμος διαγραφής ενός στοιχείου (κόμβου) παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 5.4 (T-DIAGRAFH).

Στον αλγόριθμο καταρχήν εκτελείται η βασική αναζήτηση (βήμα 1) για την εύρεση τού προς διαγραφή κόμβου. Αν δε βρεθεί (βήμα 2), τότε επι-

στρέφει αντίστοιχο μήνυμα (βήμα 3). Αν βρεθεί, τότε εξετάζεται ο αριθμός των παιδιών του. Αν έχει δύο παιδιά (βήμα 4), τότε καλείται ο μερικός αλγόριθμος 5.4β (βήμα 5), αλλιώς ο μερικός αλγόριθμος 5.4α (βήμα 6).

ΑΛΓΟΡΙΘΜΟΣ 5.4: ΔΙΑΓΡΑΦΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ

Είσοδος: Ένας δείκτης (R) στη ρίζα ενός δυαδικού δέντρου αναζήτησης και ένας δείκτης (P) στον προς διαγραφή κόμβο (στοιχείο).

Έξοδος: Εξωτερικά επιστρέφει αντίστοιχο μήνυμα όταν το στοιχείο δεν υπάρχει, αλλιώς τίποτα. Εσωτερικά γίνεται διαγραφή του κόμβου από το δέντρο όταν το στοιχείο υπάρχει.

```
T-DIAGRAFH(R, P)
1    T-BASIKH-ANAZHTHSH(R, P)                    {Βασική αναζήτηση}
2    if EURESH = FALSE                           {Έλεγχος ύπαρξης}
3       then print 'ΤΟ ΣΤΟΙΧΕΙΟ ΔΕΝ ΥΠΑΡΧΕΙ'     {Μήνυμα ανυπαρξίας}
4       else if (ADEIKTHS(KOMBOS(PX)) = NIL) and
                (DDEIKTHS(KOMBOS(PX)) = NIL)    {Έλεγχος παιδιών}
5               then T-DIAGRAFH-B(R, PX, PXG)   {Κλήση μερικού αλγορίθμου}
6               else T-DIAGRAFH-A(R, PX, PXG)   {Κλήση μερικού  αλγορίθμου}
            endif
    endif
```

5.3.5 Πολυπλοκότητα

Η πολυπλοκότητα των πράξεων σε δυαδικά δέντρα αναζήτησης εξαρτάται βασικά από την πολυπλοκότητα της αναζήτησης, όπως εύκολα μπορεί να διακρίνει κανείς εξετάζοντας τους αλγορίθμους εισαγωγής και διαγραφής στοιχείου.

Δεδομένου ενός δυαδικού δέντρου αναζήτησης ύψους h, η αναζήτηση ενός στοιχείου, στη χειρότερη περίπτωση, δηλαδή σε περίπτωση αποτυχίας ή επιτυχίας όπου το στοιχείο βρίσκεται σε φύλλο του τελευταίου επιπέδου, απαιτεί τόσες συγκρίσεις όσες το ύψος του δέντρου συν μία (Αλγόριθμος 5.2α). Το ύψος ενός δυαδικού δέντρου αναζήτησης με n στοιχεία εξαρτάται από τον τρόπο δημιουργίας του, δηλαδή από τη σειρά με την οποία εισάγονται τα στοιχεία του κατά τη δημιουργία του. Έτσι, για n στοιχεία μπορούμε να δημιουργήσουμε $n!$ διαφορετικά

δέντρα, όσοι δηλαδή και οι συνδυασμοί των n στοιχείων ανά n, τα οποία έχουν, εν γένει, διαφορετικά ύψη. Μια ακραία περίπτωση είναι όλα τα στοιχεία να τοποθετηθούν έτσι ώστε να έχουμε ένα δέντρο με ένα μονοπάτι, όπου, π.χ., κάθε κόμβος είναι αριστερό παιδί του προηγουμένου. Στην περίπτωση αυτή το ύψος του δέντρου είναι ίσο με τον αριθμό των στοιχείων (κόμβων) n. Η άλλη ακραία περίπτωση είναι να είναι ένα πλήρες δέντρο, οπότε ισχύει η σχέση για το κάτω όριο της Υποενότητας «5.2.1 Ορισμοί και Αναπαράσταση». Έτσι είναι

$$\log(n + 1) - 1 \leq h \leq n$$

και, επομένως, ο απαιτούμενος αριθμός συγκρίσεων $C(n)$ περιορίζεται από τη σχέση

$$\log(n+1) \leq C(n) \leq n+1.$$

Ένα «καλό» δέντρο είναι αυτό στο οποίο το $C(n)$ είναι κοντά στο αριστερό άκρο της ανισότητας, άρα η πολυπλοκότητα χειρότερης περίπτωσης της αναζήτησης για ένα «καλό» δέντρο θα είναι $f(n) = O(\log n)$, ενώ για ένα «κακό» δέντρο, δηλαδή για $C(n)$ κοντά στο δεξιό άκρο της ανισότητας, θα είναι $f(n) = O(n)$. Αποδεικνύεται δε ότι και για ένα «μέσο» δέντρο η πολυπλοκότητα είναι $f(n) = O(\log n)$.

Συγκρίνοντας ένα δυαδικό δέντρο με έναν πίνακα n στοιχείων, διαπιστώνουμε ότι έχουν την ίδια πολυπλοκότητα αναζήτησης, αλλά η εισαγωγή και διαγραφή στοιχείου στην περίπτωση του πίνακα έχουν μεγαλύτερη πολυπλοκότητα, λόγω των απαιτούμενων μετακινήσεων. Συγκρίνοντάς το με μια συνδεδεμένη λίστα n κόμβων, διαπιστώνουμε ότι έχουν το ίδιο εύκολη διαγραφή και εισαγωγή στοιχείων, αλλά η αναζήτηση σε μια συνδεδεμένη λίστα έχει πολυπλοκότητα $f(n) = O(n)$, δηλαδή μεγαλύτερης τάξης. Από αυτές τις συγκρίσεις φαίνεται καθαρά πόσο πλεονεκτικότερη δομή είναι, εν γένει, το δυαδικό δέντρο αναζήτησης σε σχέση με τον πίνακα και τη λίστα.

Άσκηση Αυτοαξιολόγησης 5.2

Υποθέστε ότι έχουμε την ακόλουθη λίστα από γράμματα: Ι, Ρ, Δ, Θ, Τ, Ε, Μ, Η, Π, Α, Κ, Σ.

α) Χρησιμοποιώντας τη λογική του αλγορίθμου εισαγωγής, σχηματίστε το δυαδικό δέντρο αναζήτησης που θα προκύψει, με στοιχεία (κόμβους) τα γράμματα της λίστας, αν τα στοιχεία εισάγονται με την παραπάνω σειρά.

β) Εφαρμόστε τη λογική του αλγορίθμου διαγραφής για να διαγρά-

ψετε το στοιχείο (κόμβο) 'Δ' και σχεδιάστε το δέντρο που θα προκύψει, αφού περιγράψετε τις μεταβολές που έγιναν.

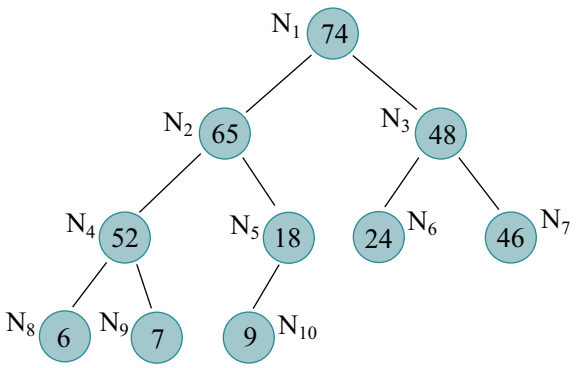
Σημειώστε ότι οι αλφαβητικοί χαρακτήρες είναι διατάξιμοι σύμφωνα με τη σειρά τους στην αλφάβητο.

5.4 Δέντρα

5.4.1 Ορισμός και αναπαράσταση

■ Ένα **δέντρο-σωρός** ή απλώς **σωρός** είναι ένα πλήρες δυαδικό δέντρο με διατεταγμένους τους κόμβους του, έτσι ώστε η τιμή του στοιχείου κάθε κόμβου, να είναι μεγαλύτερη ή ίση από τις τιμές των στοιχείων των παιδιών του.

Στην υποενότητα αυτή θα ασχοληθούμε με έναν άλλο τύπο δυαδικού δέντρου, το *δέντρο-σωρός* (*heap tree*) ή απλώς *σωρός* (*heap*). Ο σωρός χρησιμοποιείται σ' ένα γνωστό αλγόριθμο διάταξης, που λέγεται διάταξη σωρού (heapsort), με τον οποίο θα ασχοληθούμε στο επόμενο κεφάλαιο. Όπως και στα δυαδικά δέντρα αναζήτησης, έτσι και στους σωρούς ο τύπος των στοιχείων τους πρέπει να τέτοιος ώστε τα στοιχεία να είναι διατάξιμα. ■



Σχήμα 5.7
Ένα δέντρο-σωρός με στοιχεία ακέραιους αριθμούς

Στη βιβλιογραφία συναντάται μερικές φορές η διάκριση μεταξύ *σωρού μεγίστων* (*maxheap*) και *σωρού ελαχίστων* (*minheap*). Ο σωρός μεγίστων ορίζεται όπως ορίσαμε το σωρό παραπάνω, ενώ ο σωρός ελαχίστων ορίζεται κατ' αντίστοιχο τρόπο, δηλαδή το στοιχείο κάθε κόμβου είναι μικρότερο από τα στοιχεία των παιδιών του. Εδώ δε θα ακολουθήσουμε αυτή τη διάκριση.

Το δέντρο που απεικονίζεται στο Σχήμα 5.7 είναι ένα δέντρο-σωρός με στοιχεία ακέραιους αριθμούς. Αυτό συνάγεται, πρώτον, από το γεγονός ότι είναι ένα πλήρες δέντρο (για την έννοια του πλήρους δέντρου ανατρέξτε στην Υποενότητα 5.2.1 Ορισμοί και Αναπαράστα-

ση). Δεύτερον, από το γεγονός ότι το στοιχείο κάθε κόμβου (π.χ. του '65') είναι μεγαλύτερο ή ίσο από τα στοιχεία των παιδιών του ('52' και '18').

Για την υλοποίηση ενός σωρού θα χρησιμοποιήσουμε στη συνέχεια τη συνεχόμενη αναπαράσταση ενός δέντρου (ανατρέξτε στην Υποενότητα «5.2.1 Ορισμοί και Αναπαράσταση»), διότι είναι πλεονεκτικότερη της συνδεδεμένης για τις πράξεις που αφορούν τους σωρούς.

5.4.2 Εισαγωγή

Η εισαγωγή ενός στοιχείου σε ένα σωρό απαιτεί αναδιάταξη των κόμβων. Η διαδικασία που ακολουθείται είναι η εξής:

(1) Τοποθετούμε το προς εισαγωγή στοιχείο (κόμβο) στο τέλος του δέντρου έτσι ώστε να εξακολουθεί να είναι ένα πλήρες δέντρο, αλλά όχι απαραίτητα και σωρός.

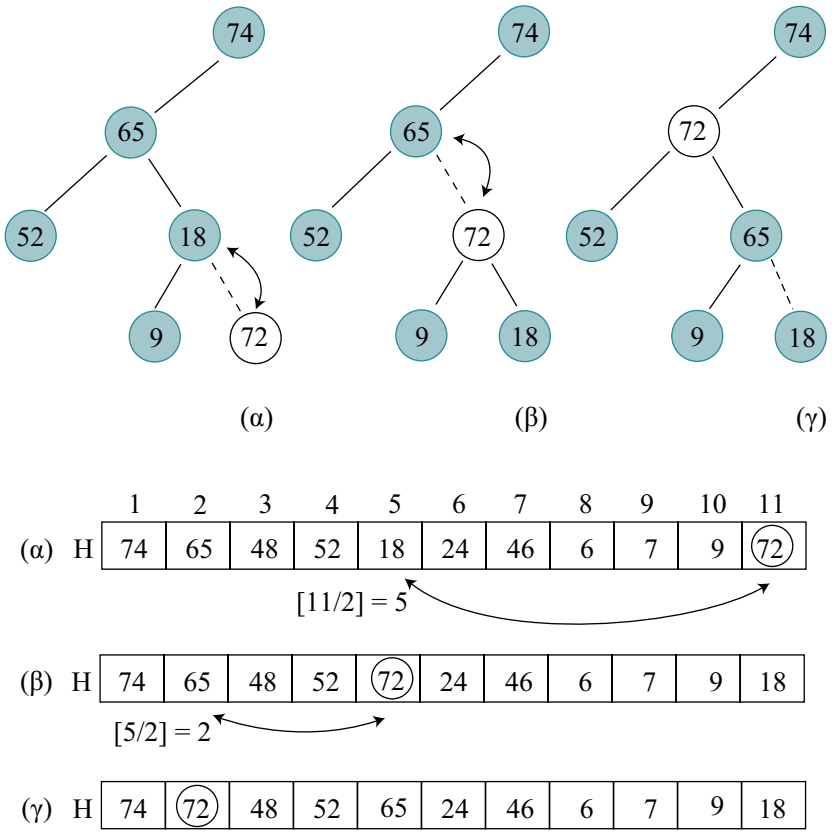
(2) Μετακινούμε το στοιχείο (κόμβο) προς τη ρίζα του δέντρου και το τοποθετούμε σε θέση ώστε να δημιουργείται σωρός, δηλαδή ο γονέας του να έχει μεγαλύτερη τιμή.

Παράδειγμα 5.2

Εισαγωγή στοιχείου σε δέντρο-σωρό

Θεωρούμε το δέντρο-σωρός του Σχήματος 5.7, στο οποίο θέλουμε να εισαγάγουμε το στοιχείο '72'. Η διαδικασία εισαγωγής απεικονίζεται γραφικά στα παρακάτω σχήματα (α), (β) και (γ), καθώς και στις αντίστοιχες αναπαραστάσεις του πίνακα που αντιπροσωπεύει το δέντρο.

Στο (α) το στοιχείο τοποθετείται στο τέλος του δέντρου (πίνακα). Επειδή 72 > 18, το '72' αλλάζει θέση με το γονέα του ('18') (Απεικονίσεις β). Επειδή και πάλι 72 > 65, μετακινείται στη θέση του γονέα του ('65') (Απεικονίσεις γ). Εκεί τελικά και τοποθετείται, διότι 72 < 74.



Ο αλγόριθμος που υλοποιεί την παραπάνω διαδικασία παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 5.5 (H-EISAGWGH).

Στον αλγόριθμο αυτό χρησιμοποιούνται τρεις βοηθητικές μεταβλητές, οι P, G και THESH. Οι μεταβλητές P και G είναι δύο δείκτες που αντιπροσωπεύουν ο μεν πρώτος την τρέχουσα θέση του μετακινούμενου στοιχείου στον πίνακα, ο δε δεύτερος τη θέση του γονέα του. Η THESH είναι μια λογική μεταβλητή που γίνεται αληθής στην περίπτωση που βρεθεί θέση, πλην της ρίζας, για την τοποθέτηση του στοιχείου.

Στο πρώτο βήμα του αλγορίθμου γίνεται αρχικοποίηση των μεταβλητών. Σημειώστε εδώ ότι το μέγεθος του πίνακα είναι, εν γένει, μεγαλύτερο από N, που αντιπροσωπεύει το μέγεθος του δέντρου (ανατρέξτε στην Υποενότητα 5.2.1 Ορισμοί και Αναπαράσταση, τμήμα Συνεχόμενη Αναπαράσταση). Η αύξηση του N κατά ένα υλοποιεί ουσιαστικά την προσθήκη του στοιχείου στο τέλος του πίνακα (δέντρου). Στη συνέχεια, η διάταξη while (βήματα 2-6) υλοποιεί τη

μετακίνηση του στοιχείου προς τη ρίζα. Η μετακίνηση γίνεται με το να παίρνει κάθε φορά το στοιχείο τη θέση του γονέα του (βήμα 6). Η μετακίνηση σταματά (βήμα 2) όταν το στοιχείο βρεθεί σε θέση που είναι μεγαλύτερο από τα παιδιά του (διάταξη if, βήματα 4-5). Τέλος, με μια διάταξη if ανιχνεύεται αν βρέθηκε θέση για το στοιχείο (βήμα 7). Αν δε βρέθηκε, τότε τοποθετείται στη ρίζα (βήμα 8).

ΑΛΓΟΡΙΘΜΟΣ 5.5: ΕΙΣΑΓΩΓΗ ΣΕ ΣΩΡΟ

Είσοδος: Ένας πίνακας (H) που αναπαριστά το δέντρο-σωρό, ο αριθμός κόμβων του δέντρου (N) και το προς εισαγωγή στοιχείο (X).

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται εισαγωγή του στοιχείου στο δέντρο.

```
H-EISAGWGH(H, N, X)
1   N ← N+1, P ← N, THESH ← FALSE          {Αρχικοποίηση μεταβλητών}
2   while (P<1) and (not THESH)            {Συνθήκη επανάληψης}
3      G ← [P/2]                           {Ενημέρωση γονέα}
4      if X ≤ H[G]                         {Έλεγχος σωστής θέσης}
5         then H[P] ←X, THESH ← TRUE       {Τοποθέτηση στοιχείου}
       endif
6      H[P] ← H[G], P ← G                  {Εναλλαγή στοιχείου-γονέα}
    endwhile
7   if THESH=FALSE                         {Έλεγχος αποτελέσματος}
8      then H[1] ←X                        {Τοποθέτηση στη ρίζα}
    endif
```

5.4.3 Διαγραφή

Αυτό που μας ενδιαφέρει στην πράξη διαγραφής ενός στοιχείου (κόμβου) από ένα σωρό είναι η περίπτωση διαγραφής της ρίζας του, διότι αυτό χρειάζεται και στον αλγόριθμο διάταξης σωρού. Η διαδικασία αυτή έχει ως εξής:

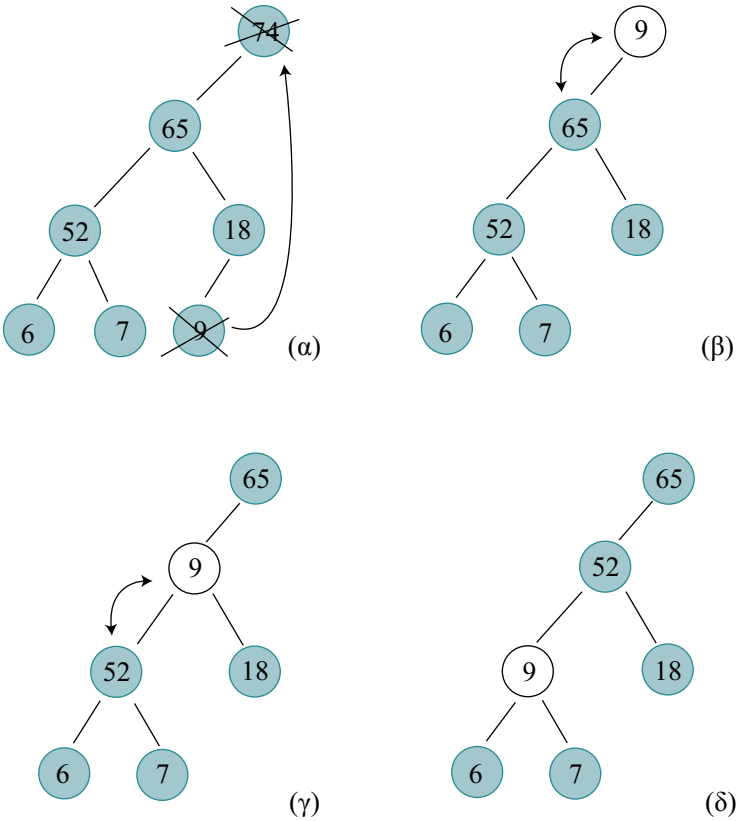
(1) Κρατάμε τη ρίζα σε μια μεταβλητή,

(2) Αντικαθιστούμε τη ρίζα με το τελευταίο στοιχείο (κόμβο) του δέντρου, έτσι ώστε το δέντρο να παραμείνει πλήρες, όχι όμως απαραίτητα σωρός,

(3) Μετακινούμε το στοιχείο (κόμβο) προς τα κάτω μέχρις ότου τοποθετηθεί στην κατάλληλη θέση, έτσι ώστε να δημιουργηθεί σωρός.

Παράδειγμα 5.3 *Διαγραφή ρίζας σωρού*

Θεωρούμε το δέντρο-σωρό του Σχήματος 5.8, από το οποίο θέλουμε να διαγράψουμε τη ρίζα του. Η διαδικασία διαγραφής απεικονίζεται στα παρακάτω σχήματα (όπου χάρη οικονομίας χώρου χρησιμοποιείται μέρος του δέντρου).



Στο (α) γίνεται διαγραφή της ρίζας και διαγραφή και μεταφορά του τελευταίου στοιχείου (κόμβου), δηλαδή του '9', στη θέση της ρίζας ('74'), οπότε καταλήγουμε στο δέντρο του Σχήματος (β). Επειδή 65 > 48 (ο κόμβος '48' δε φαίνεται στο σχήμα) και 9 < 65, απαιτείται εναλλαγή των '9' και '65'. Έτσι, καταλήγουμε στο Σχήμα (γ). Τώρα, επειδή 52 > 18 και 9 < 52, απαιτείται νέα εναλλαγή, των '9' και '52' αυτή τη φορά. Έτσι, καταλήγουμε στο (δ), όπου 9 > 6 και 9 > 7, οπότε αυτή είναι και η τελική θέση του '9'.

Ο αλγόριθμος που υλοποιεί τη διαδικασία αυτή παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 5.6 (H-DIAGRAFH).

Στον αλγόριθμο αυτό χρησιμοποιούμε πέντε βοηθητικές μεταβλητές, τέσσερις ακέραιες (P, Z, AD, DD) και μια λογική (SUCCESS). Η P περιέχει την τρέχουσα θέση του μετακινούμενου (τελευταίου) στοιχείου, ενώ η Z το ίδιο το στοιχείο. Οι AD και DD περιέχουν τις θέσεις του αριστερού και δεξιού παιδιού, αντίστοιχα, του στοιχείου στην τρέχουσα θέση P. Η SUCCESS γίνεται αληθής, όταν βρεθεί κατάλληλη θέση για το μετακινούμενο στοιχείο.

Καταρχήν γίνεται αποθήκευση της ρίζας του σωρού στη μεταβλητή X, του τελευταίου στοιχείου στη Z και ελάττωση του μεγέθους του δέντρου κατά ένα (σύνθετο βήμα 1), δηλαδή γίνεται διαγραφή της ρίζας και του τελευταίου στοιχείου του σωρού. Στη συνέχεια γίνεται αρχικοποίηση των μεταβλητών P, AD, DD και SUCCESS (βήμα 2). Κατόπιν ξεκινά το κυρίως μέρος του αλγορίθμου, που είναι μια διάταξη επανάληψης while (βήματα 3-9). Η διάταξη επανάληψης περιέχει βασικά ένα σύνθετο if (βήματα 4-8). Το πρώτο τμήμα του (βήματα 4-5) αποτελεί έλεγχο της συνθήκης τερματισμού, αν δηλαδή η τρέχουσα θέση είναι τέτοια που το μετακινούμενο στοιχείο είναι μεγαλύτερο ή ίσο από τα παιδιά του (βήμα 4), οπότε τοποθετείται εκεί και ενημερώνεται και η λογική μεταβλητή (βήμα 5). Το δεύτερο τμήμα της σύνθετης διάταξης if (βήματα 6-8) αφορά τη μετακίνηση του στοιχείου, αφού η παρούσα θέση του δεν είναι η κατάλληλη. Ανάλογα με το ποιο από τα τρέχοντα παιδιά είναι μεγαλύτερο (το αριστερό ή το δεξιό, βήμα 6), γίνεται η μετακίνηση του στοιχείου και ενημερώνεται και ο δείκτης P (βήμα 7 ή 8). Στο βήμα 9 γίνεται επανυπολογισμός των AD και DD πριν από την επόμενη επανάληψη (μετακίνηση). Οι επαναλήψεις τερματίζονται είτε όταν βρεθεί η κατάλληλη θέση για το μετακινούμενο στοιχείο είτε όταν η τιμή του δείκτη AD γίνει ίση ή μεγαλύτερη του N. Αν συμβαίνει το δεύτερο, ξεχωρίζουμε την περίπτωση να είναι AD = N, δηλαδή όταν το μετακινούμενο στοιχείο είναι γονέας του εναπομείναντος ενός στοιχείου, οπότε, αν ταυτόχρονα είναι και μικρότερο του εναπομείναντος στοιχείου (βήμα 10), γίνεται εναλλαγή των στοιχείων αυτών (βήμα 11). Αλλιώς, το μετακινούμενο στοιχείο τοποθετείται στην τρέχουσα θέση του δείκτη P (βήμα 12).

ΑΛΓΟΡΙΘΜΟΣ 5.6: ΔΙΑΓΡΑΦΗ ΣΕ ΣΩΡΟ

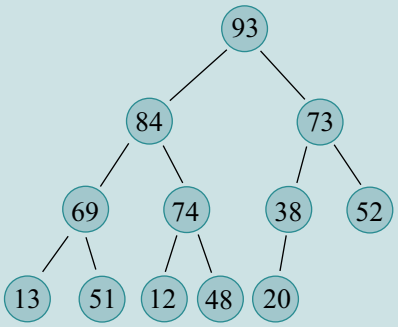
Είσοδος: Ένας πίνακας (H) που αναπαριστά το δέντρο-σωρό, ο αριθμός κόμβων του δέντρου (N).
Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται διαγραφή της ρίζας του δέντρου που καταχωρίζεται σε μια μεταβλητή (X).

```
H-DIAGRAFH(H, N)
1    X ← H[1], Z ← H[N], N ← N-1                    {Διαγραφή ρίζας και στοιχείου}
2    P ← 1, AD ← 2, DD ← 3, SUCCESS ← FALSE         {Αρχικοποίηση μεταβλητών}
3    while (AD < N) and (not SUCCESS)               {Συνθήκη επανάληψης}
4      if (Z ≥ H[AD]) and (Z ≥ H[DD])               {Έλεγχος σωστής θέσης}
5        then H[P] ←Z, SUCCESS ← TRUE               {Τοποθέτηση στοιχείου}
6        else if H[DD] ← H[AD]                      {Έλεγχος μεγαλύτερου παιδιού}
7                then H[P] ← H[AD], P ← AD          {Μετακίνηση αριστερά}
8                else H[P] ← H[DD], P ← DD          {Μετακίνηση δεξιά}
               endif
       endif
9      AD ← 2*P, DD ← 2*P+1                         {Ενημέρωση δεικτών}
     endwhile
10   if (AD = N) and (Z < H[AD])                    {Έλεγχος ακραίας περίπτωσης}
11     then H[P] ← H[AD], P ← AD                    {Εναλλαγή στοιχείων}
12   H[P] ← Z
```

Άσκηση Αυτοαξιολόγησης 5.3

(α) Ελέγξτε αν το παρακάτω δέντρο είναι ένα δέντρο-σωρός και δώστε τη συνεχόμενη αναπαράστασή του.

(β) Περιγράψτε τις αλλαγές που γίνονται στο δέντρο κατά τη διαδικασία της εισαγωγής του στοιχείου ‘75’.



5.5 Αλλες κατηγορίες δεντρών

Όπως αναφέραμε στην Υποενότητα «5.3.5 Πολυπλοκότητα», η πολυπλοκότητα αναζήτησης ενός στοιχείου σ' ένα δυαδικό δέντρο αναζήτησης εξαρτάται από το ύψος του, το οποίο με τη σειρά του εξαρτάται από τον τρόπο κατασκευής του δέντρου. Αυτό σημαίνει ότι, αν κατασκευάσουμε ένα δέντρο του οποίου οι τερματικοί κόμβοι έχουν μεγάλες διαφορές βάθους, τότε δημιουργούμε δέντρο με μεγάλο ύψος. Αντίθετα, αν οι τερματικοί κόμβοι δεν έχουν μεγάλες διαφορές βάθους, τότε δημιουργούμε δέντρο με μικρό ύψος. Στη δεύτερη περίπτωση, τα δέντρα που δημιουργούνται ονομάζονται *ισοζυγισμένα δέντρα* (*balanced trees*). Τα ισοζυγισμένα δέντρα εγγυώνται πολυπλοκότητα $f(n) = O(\log n)$. Διακρίνουμε δύο κατηγορίες ισοζυγισμένων δέντρων: τα *υψοζυγισμένα δέντρα* ή *ισοζυγισμένα δέντρα κατά ύψος* (*height balanced trees*) και τα *βαροζυγισμένα δέντρα* ή *ισοζυγισμένα δέντρα κατά βάρος* (*weight balanced trees*).

5.5.1 Υψοζυγισμένα δέντρα

Τα υψοζυγισμένα δέντρα κατασκευάζονται έτσι ώστε τα ύψη των υποδέντρων κάθε κόμβου να μη διαφέρουν πολύ μεταξύ τους. Βέβαια, η αρχική αυτή ισορροπία διαταράσσεται με κάθε πράξη εισαγωγής ή διαγραφής, οπότε απαιτείται *επαναζύγιση* (*rebalance*) του δέντρου. Δύο βασικές πράξεις που χρησιμοποιούνται σε τέτοια δέντρα για επαναζύγιση είναι η *απλή περιστροφή* (*single rotation*) και η *διπλή περιστροφή* (*double rotation*).

Εκπρόσωποι της κατηγορίας των υψοζυγισμένων δυαδικών δέντρων είναι τα *δέντρα AVL* (*AVL trees*) και τα *κόκκινα-μαύρα δέντρα* (*red-black trees*). Τα δέντρα AVL, πήραν το όνομά τους από τα αρχικά των ονομάτων των δύο Ρώσων μαθηματικών που τα εισήγαγαν (Adelson-Velskii και Landis). Η βασική ιδέα ήταν η δημιουργία δέντρων που να είναι *σχεδόν ισοζυγισμένα* (*almost balanced*) και όχι *πλήρως* ή *τέλεια ισοζυγισμένα* (*completely* ή *perfectly balanced*). Έτσι, ένα δυαδικό δέντρο είναι δέντρο AVL, αν για κάθε κόμβο του ισχύει ότι τα ύψη του αριστερού και του δεξιού του υποδέντρου είναι είτε ίσα είτε διαφέρουν κατά ένα (1). Σε κάθε κόμβο, επομένως, αποθηκεύεται μια πρόσθετη πληροφορία (πληροφορία ζύγισης) για τη διαφορά ύψους μεταξύ αριστερού και δεξιού υποδέντρου, η οποία μπορεί να είναι –1, 0 ή 1.

Στα κόκκινα-μαύρα δέντρα, οι κόμβοι διακρίνονται σε κόκκινους και μαύρους. Η ρίζα και οι τερματικοί κόμβοι (φύλλα) είναι μαύροι κόμβοι, ενώ οι εσωτερικοί κόμβοι είναι μαύροι ή κόκκινοι, με βάση τούς εξής κανόνες: (α) Μεταξύ δύο κόκκινων κόμβων παρεμβάλλεται τουλάχιστον ένας μαύρος και (β) όλες οι διαδρομές από τη ρίζα σε κάποιον τερματικό κόμβο περιέχει τον ίδιο αριθμό μαύρων κόμβων. Μ' αυτό τον τρόπο επιτυγχάνεται ο λόγος της μεγαλύτερης προς τη μικρότερη διαδρομή να είναι το πολύ δύο (2). Σε κάθε κόμβο η πληροφορία ζύγισης είναι ένα ψηφίο (0 ή 1), που παριστάνει το χρώμα του κόμβου.

5.5.2 Βαροζυγισμένα δέντρα

Τα βαροζυγισμένα δέντρα προσπαθούν να έχουν υποδέντρα περίπου ίσου βάρους, δηλαδή ίδιου αριθμού κόμβων. Χρησιμοποιούν και αυτά την απλή και διπλή περιστροφή ως βασικές πράξεις επαναζύγισης, όπως και τα υψοζυγισμένα δέντρα. Βασικός εκπρόσωπος της κατηγορίας αυτής είναι τα δέντρα BB[a].

Σύνοψη

Τα δέντρα είναι από τις σπουδαιότερες ανώτερες δομές δεδομένων. Ένα δέντρο αποτελείται από κόμβους και ακμές που συνδέουν τους κόμβους έτσι ώστε γραφικά η δομή να μπορεί να παρασταθεί σαν ένα δέντρο. Ο κόμβος της κορυφής, από τον οποίο μόνο ξεκινούν ακμές, ονομάζεται ***ρίζα***. Οι κόμβοι στη βάση, στους οποίους μόνο καταλήγουν ακμές, ονομάζονται ***τερματικοί κόμβοι*** ή ***φύλλα***. Όλοι οι υπόλοιποι ονομάζονται ***εσωτερικοί κόμβοι***. Κάθε κόμβος έχει κανένα, έναν ή περισσότερους κόμβους ***παιδιά***. Επίσης, κάθε κόμβος (πλην της ρίζας) έχει έναν κόμβο ***γονέα***. Τέλος, σε κάθε κόμβο θεωρούμενο ως ρίζα διακρίνουμε δύο υποδέντρα, το ***αριστερό υποδέντρο*** και το ***δεξιό υποδέντρο***. Το ***κενό δέντρο*** είναι ένα δέντρο χωρίς καθόλου κόμβους.

Τα ***δυαδικά δέντρα*** είναι τα πιο συχνά χρησιμοποιούμενα δέντρα, διότι συνδυάζουν απλότητα κατασκευής και χειρισμού με υψηλή αποδοτικότητα. Σ' ένα δυαδικό δέντρο κάθε κόμβος έχει το πολύ δύο παιδιά. Ένα δυαδικό δέντρο μπορεί να αναπαρασταθεί είτε με συνεχόμενη, μέσω πίνακα, είτε με συνδεδεμένη, μέσω συνδέσμων, ανα-

παράσταση. Η πράξη που μας ενδιαφέρει σ' ένα απλό δυαδικό δέντρο είναι η διαπέραση, που μπορεί να γίνει με τρεις τρόπους: ***προδιατεταγμένη***, ***ενδοδιατεταγμένη*** και ***μεταδιατεταγμένη διαπέραση***, που διαφέρουν μεταξύ τους ως προς τη σειρά επίσκεψης της ρίζας και των δύο υποδέντρων του δέντρου.

Η σπουδαιότερη κατηγορία δέντρων όμως είναι τα ***δυαδικά δέντρα αναζήτησης***. Στις περισσότερες περιπτώσεις έχουν πολυπλοκότητα $f(n) = O(\log n)$ και για τις τρεις βασικές πράξεις (αναζήτηση, εισαγωγή και διαγραφή), σε αντίθεση με τους πίνακες και τις λίστες. Σ' ένα δυαδικό δέντρο αναζήτησης το στοιχείο κάθε κόμβου είναι μεγαλύτερο από τα στοιχεία του αριστερού υποδέντρου και μικρότερο από αυτά του δεξιού. Συνήθως χρησιμοποιείται η συνδεδεμένη αναπαράσταση για την υλοποίηση ενός τέτοιου δέντρου.

Μια άλλη σημαντική κατηγορία δέντρων είναι οι ***σωροί***. Σ' ένα ***δέντρο-σωρό*** το στοιχείο κάθε κόμβου είναι μεγαλύτερο από τα στοιχεία των παιδιών του. Οι σωροί είναι δομές που χρησιμοποιούνται σ' ένα γνωστό αλγόριθμο διάταξης, τη διάταξη σωρού. Συνήθως χρησιμοποιείται η συνεχόμενη αναπαράσταση για την υλοποίηση ενός σωρού. Οι σωροί και τα δυαδικά δέντρα αναζήτησης είναι διατεταγμένα δέντρα, και γι' αυτό τα στοιχεία τους πρέπει να είναι διατάξιμα.

Το μειονέκτημα των δυαδικών δέντρων αναζήτησης είναι ότι δεν εξασφαλίζουν καλή πολυπλοκότητα σε όλες τις περιπτώσεις. Το μειονέκτημα αυτό προέρχεται από το γεγονός ότι δεν εξασφαλίζουν όλα μικρό ύψος δέντρου, από το οποίο εξαρτάται η πολυπλοκότητα χρόνου. Το μειονέκτημα αυτό έρχονται να εξαλείψουν τα ***ισοζυγισμένα δέντρα***, τα οποία καταφέρνουν να εξασφαλίζουν μικρό ύψος και, επομένως, χαμηλή πολυπλοκότητα. Η πιο σημαντική υποκατηγορία αυτών των δέντρων είναι τα ***υψοζυγισμένα***, που διατηρούν τη διαφορά ύψους των υποδέντρων κάθε κόμβου πολύ μικρή. Ένα παράδειγμα είναι τα ***δέντρα AVL***, στα οποία αυτή η διαφορά είναι το πολύ ένα. Μια άλλη υποκατηγορία είναι τα ***βαροζυγισμένα δέντρα***, που διατηρούν μικρή τη διαφορά «βάρους», δηλαδή τον αριθμό των κόμβων, μεταξύ των δύο υποδέντρων κάθε κόμβου.

Οδηγός για περαιτέρω μελέτη

1. Ι. Μανωλόπουλος, *Δομές Δεδομένων*, τόμ. Α', ART of TEXT, 2η έκδοση, 1992.

   Στο κεφάλαιο 5 του βιβλίου με τίτλο «Ισοζυγισμένα Δέντρα» γίνεται μια παρουσίαση των διαφόρων κατηγοριών και τύπων ισοζυγισμένων δέντρων, μεταξύ των οποίων και τα δέντρα AVL.

2. Α. Τσακαλίδης, *Δομές Δεδομένων*, Πανεπιστημιακές Παραδόσεις, Τμήμα Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Πατρών, 1994.

   Στην Υποενότητα «4.2.1 Ισοζυγισμένα δέντρα» του βιβλίου υπάρχει μια σχετικά συνοπτική περιγραφή για δέντρα AVL, για κόκκινα-μαύρα δέντρα και για δέντρα BB[a]. Επίσης, στην Προσθήκη 1 του βιβλίου υπάρχει μια εκτενής αναφορά στα δέντρα BB, που είναι μια παραλλαγή των κόκκινων-μαύρων δέντρων.

3. T. H. Cormen, C. E. Leiserson και R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1996 (ανατύπωση).

   Στο κεφάλαιο 14 με τίτλο «Red-Black Trees» υπάρχει μια εκτενής αναφορά στα κόκκινα-μαύρα δέντρα.

4. T. A. Standish, *Data Structures, Algorithm and Software Principles*, Addison-Wesley, 1994.

   Στην Ενότητα «9.7 Binary Search Trees» στο τμήμα «Analyzing Performance Characteristics» μπορείτε να βρείτε μια καλή ανάλυση της πολυπλοκότητας της αναζήτησης σε δυαδικά δέντρα αναζήτησης. Επίσης, στην Ενότητα «AVL Trees and Their Performance» υπάρχει μια εκτενής αναφορά σε δέντρα AVL, καθώς και ανάλυση της αποδοτικότητάς τους.

5. M. A. Weis, *Data Structures and Algorithm Analysis*, Benjamin/Cummings, 2nd Edition, 1995.

   Στο Κεφάλαιο 4 με τίτλο «Trees» μπορείτε να βρείτε πρακτικούς αλγορίθμους σε Pascal που σχετίζονται με τα δυαδικά δέντρα. Ιδιαίτερα στην Ενότητα «4.4 AVL Trees» υπάρχει εκτενής αναφορά για τα δέντρα AVL.

Διάταξη

ΚΕΦΑΛΑΙΟ 6

Σκοπός

Στο κεφάλαιο αυτό παρουσιάζονται μερικές από τις μεθόδους διάταξης στοιχείων. Η παρουσίαση αφορά τη βασική ιδέα, τον αλγόριθμο και την πολυπλοκότητα της κάθε μεθόδου.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό, θα είστε σε θέση να:

- *σχεδιάζετε παραλλαγές των αλγόριθμων διάταξης,*
- *σχεδιάζετε συνδυασμούς των αλγόριθμων διάταξης,*
- *εφαρμόζετε τους αλγορίθμους διάταξης σε συγκεκριμένες ακολουθίες δεδομένων.*

Έννοιες κλειδιά

- *Αλγόριθμος Διάταξης*
- *Εσωτερική Διάταξη*
- *Εξωτερική Διάταξη*
- *Σύγκριση*
- *Εναλλαγή*
- *Διάταξη Επιλογής*
- *Γρήγορη Διάταξη*
- *Άξονας*
- *Διαχωρισμός*
- *Διάταξη Σωρού*

Ενότητες Κεφαλαίου 6

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο αυτό ασχολούμαστε με το πρόβλημα της διάταξης μιας ακολουθίας στοιχείων, που είναι, μαζί με την αναζήτηση, από τα θεμελιώδη προβλήματα στην επιστήμη των υπολογιστών. Το πρόβλημα (ή την πράξη) της αναζήτησης το πραγματευτήκαμε κατανεμημένα στα προηγούμενα κεφάλαια. Στο κεφάλαιο αυτό εξετάζουμε αποκλειστικά το πρόβλημα (ή την πράξη) της διάταξης. Παρουσιάζουμε τρεις μεθόδους (αλγορίθμους) διάταξης, που θεωρούμε ότι είναι οι αντιπροσωπευτικότεροι.

Στην πρώτη ενότητα ορίζουμε το πρόβλημα της διάταξης, καθώς και το πλαίσιο στο οποίο θα αναφερθούμε. Επίσης, αναφερόμαστε σε ορισμένες εισαγωγικές έννοιες και συμβολισμούς απαραίτητους για τις επόμενες ενότητες.

Στη δεύτερη ενότητα παρουσιάζουμε τη μέθοδο «διάταξη επιλογής», όσον αφορά τον αλγόριθμο, την εφαρμογή και την πολυπλοκότητά της. Η διάταξη επιλογής είναι μια από τις απλούστερες, αλλά υπολογιστικά ακριβότερες μεθόδους.

Στην τρίτη ενότητα παρουσιάζουμε ίσως τη δυσκολότερη από πλευράς κατανόησης και την περισσότερο ερευνηθείσα μέθοδο, τη «γρήγορη διάταξη». Στηρίζεται σε μια διαφορετική φιλοσοφία από αυτή της διάταξης επιλογής και είναι αποδοτικότερη στη μέση περίπτωση.

Τέλος, στην τέταρτη ενότητα εξετάζουμε την αποδοτικότερη μέθοδο διάταξης, τη «διάταξη σωρού». Η μέθοδος αυτή χρησιμοποιεί ένα δέντρο, ένα σωρό, για να επιτύχει τη διάταξη των στοιχείων.

6.1 Εισαγωγικές έννοιες

Το πρόβλημα της *διάταξης* ή *ταξινόμησης* (*sorting*) αναφέρεται στην τακτοποίηση μιας δοθείσας ακολουθίας δεδομένων με κάποια ζητούμενη σειρά, αύξουσα ή φθίνουσα, προκειμένου περί αριθμητικών δεδομένων, αλφαβητική ή αντίστροφα αλφαβητική, προκειμένου περί

αλφαβητικών χαρακτήρων. Τα δεδομένα αυτά θεωρούμε ότι βρίσκονται στην κύρια μνήμη του Η/Υ, τοποθετημένα σε κάποιον πίνακα, και ότι η ζητούμενη διάταξη είναι αύξουσα ή αλφαβητική σειρά.

Επομένως, το πρόβλημα της διάταξης n στοιχείων τίθεται ως εξής: Δοθέντος ενός πίνακα A με n στοιχεία $A_1, A_1, \ldots, A_n$, να αναδιαταχθούν τα στοιχεία του έτσι ώστε:

$$A_1 \leq A_2 \leq \ldots \leq A_n.$$

Συνήθως τα δεδομένα που θέλουμε να διατάξουμε είναι εγγραφές που είναι αποθηκευμένες σε πίνακες εγγραφών. Στην περίπτωση αυτή, η διάταξη αφορά στα κλειδιά των εγγραφών.

Υπάρχουν αρκετοί μέθοδοι ή *αλγόριθμοι διάταξης* (*sorting algorithms*) που δίνουν λύση στο παραπάνω πρόβλημα. Τέτοιοι είναι οι εξής:

- διάταξη φυσαλίδας (bubble sort)
- διάταξη εισαγωγής (insertion sort)
- διάταξη επιλογής (selection sort)
- γρήγορη διάταξη (quicksort)
- διάταξη συγχώνευσης (merge sort)
- διάταξη σωρού (heapsort)
- διάταξη ρίζας (radix sort)

Επίσης υπάρχουν παραλλαγές τους. Οι αλγόριθμοι αυτοί χαρακτηρίζονται ως *αλγόριθμοι εσωτερικής διάταξης* (*internal sorting algorithms*), σε αντιδιαστολή με τους *αλγορίθμους εξωτερικής διάταξης* (*external sorting algorithms*). Οι πρώτοι αναφέρονται στη διάταξη στοιχείων αποθηκευμένων (σε πίνακα) στην κύρια μνήμη του Η/Υ, ενώ οι δεύτεροι στη διάταξη στοιχείων αποθηκευμένων (σε αρχεία) στη δευτερεύουσα μνήμη του Η/Υ. Σ' αυτό το βιβλίο δε θα ασχοληθούμε με αλγορίθμους εξωτερικής διάταξης. Από τους αλγορίθμους εσωτερικής διάταξης θα παρουσιάσουμε τρεις: διάταξη επιλογής, γρήγορη διάταξη και διάταξη σωρού, ως τους αντιπροσωπευτικότερους.

Ένα ζήτημα που ενδιαφέρει τους αλγορίθμους διάταξης, όπως και οποιονδήποτε αλγόριθμο, είναι η πολυπλοκότητα χρόνου ως συνάρτηση του αριθμού των προς διάταξη δεδομένων (στοιχείων) (ανα-

τρέξτε στην Ενότητα «1.3 Πολυπλοκότητα Αλγόριθμων» για βασικές έννοιες σχετικές με την πολυπλοκότητα αλγόριθμων). Στους αλγορίθμους διάταξης συναντάμε τις ακόλουθες βασικές υπολογιστικές πράξεις:

(α*) συγκρίσεις* (*comparisons*), για τη σύγκριση δύο στοιχείων του πίνακα,

(β) *εναλλαγές* ή *αντιμεταθέσεις* (*interchanges*), για την αμοιβαία αλλαγή των τιμών δύο στοιχείων του πίνακα,

(γ) *καταχωρίσεις* (*assignments*), για καταχώριση τιμών σε μεταβλητές ή στοιχεία του πίνακα.

Στην πράξη, ως κριτήριο πολυπλοκότητας θεωρούνται κυρίως οι συγκρίσεις και οι εναλλαγές, οπότε η πολυπλοκότητα είναι $f(n) = C(n) + I(n)$, όπου $C(n)$ ο αριθμός των συγκρίσεων και $I(n)$ ο αριθμός των εναλλαγών που πραγματοποιούνται κατά την εκτέλεση ενός αλγορίθμου. Δεδομένου όμως ότι ο αριθμός των εναλλαγών, αλλά και των καταχωρίσεων, είναι συνήθως ανάλογος του αριθμού των συγκρίσεων, για μια ποιοτική ανάλυση της πολυπλοκότητας είναι ικανοποιητικό να θεωρούμε $f(n) = C(n)$.

Επειδή στους αλγορίθμους που θα εξετάσουμε χρησιμοποιείται συχνά η εναλλαγή στοιχείων, για λόγους απλούστερης παρουσίασης εισάγουμε το συμβολισμό '↔' για την αναπαράσταση της εναλλαγής. Έτσι, μια εντολή της μορφής

$$A[I] \leftrightarrow A[J]$$

σημαίνει την αμοιβαία αλλαγή τιμών μεταξύ των A[I] και A[J] και ισοδυναμεί με τη σύνθετη εντολή

$$Y \leftarrow A[I],\ A[I] \leftarrow A[J],\ A[J] \leftarrow Y,$$

όπου Y μια βοηθητική ή ενδιάμεση, όπως συνήθως αποκαλείται, μεταβλητή.

6.2 Διάταξη επιλογής

Ο αλγόριθμος για τη διάταξη επιλογής στηρίζεται στην παρακάτω διαδικασία:

(1) Βρίσκουμε το μικρότερο στοιχείο του πίνακα και το τοποθετού-

με ως πρώτο στοιχείο του, δηλαδή το εναλλάσσουμε με το πρώτο στοιχείο του πίνακα.

(2) Βρίσκουμε το μικρότερο στοιχείο του υποπίνακα που δεν περιλαμβάνει το πρώτο τοποθετημένο στοιχείο, δηλαδή περιλαμβάνει τα υπόλοιπα n-1 στοιχεία του, και το τοποθετούμε ως δεύτερο στοιχείο του πίνακα.

(3) Βρίσκουμε το μικρότερο στοιχείο του υποπίνακα που δεν περιλαμβάνει τα δύο (2) πρώτα τοποθετημένα στοιχεία, δηλαδή περιλαμβάνει τα υπόλοιπα n-2 στοιχεία, του πίνακα και το τοποθετούμε ως τρίτο στοιχείο του πίνακα.

.
.
.

(n-1) Βρίσκουμε το μικρότερο στοιχείο του υποπίνακα που δεν περιλαμβάνει τα n-2 τοποθετημένα στοιχεία, δηλαδή περιλαμβάνει τα δύο (2) τελευταία στοιχεία, του πίνακα και το τοποθετούμε ως το n-1 στοιχείο του.

Σύμφωνα με τη διαδικασία αυτή, απαιτούνται n-1 περάσματα του πίνακα για την πλήρη διάταξή του. Σε κάθε πέρασμα τοποθετείται στη σωστή θέση και ένα στοιχείο του πίνακα. Έτσι, το τελευταίο που μένει είναι στη σωστή θέση.

Παράδειγμα 6.1

Διάταξη πίνακα με επιλογή

Θεωρούμε έναν πίνακα Α που περιέχει τα στοιχεία 65, 42, 25, 85, 14, 56 και 33. Η εφαρμογή της διαδικασίας της διάταξης επιλογής απεικονίζεται στον παρακάτω πίνακα, όπου σε κάθε πέρασμα σημειώνεται σε κύκλο το μικρότερο στοιχείο που βρέθηκε και σε τετράγωνο το στοιχείο με το οποίο θα αλλάξουν θέση.

Πέρασμα	A_1	A_2	A_3	A_4	A_5	A_6	A_7
1	[65]	42	25	85	(14)	56	33
2	**14**	[42]	(25)	85	65	56	33
3	**14**	**25**	[42]	85	65	56	(33)
4	**14**	**25**	**33**	[85]	65	56	(42)
5	**14**	**25**	**33**	**42**	[65]	(56)	85
6	**14**	**25**	**33**	**42**	**56**	([65])	85
7	**14**	**25**	**33**	**42**	**56**	**65**	**85**

Επίσης, σε κάθε πέρασμα φαίνεται σκιασμένο το τμήμα του πίνακα που είναι σε διάταξη και, επομένως, το υπόλοιπο είναι ο υποπίνακας του οποίου βρίσκουμε το μικρότερο στοιχείο. Έτσι, στο πέρασμα 1 είναι όλος ο πίνακας, στο πέρασμα 2 από το στοιχείο A_2 και πέρα, στο πέρασμα 3 από το στοιχείο A_3 και πέρα κ.ο.κ.

Όπως φαίνεται και από το παραπάνω παράδειγμα, για την υλοποίηση του αλγορίθμου απαιτείται η υλοποίηση ενός μερικού αλγορίθμου που θα βρίσκει σε κάθε πέρασμα το μικρότερο στοιχείο του αντίστοιχου (υπο)πίνακα. Ο αλγόριθμος αυτός παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 6.1α (MINSTOIXEIO). (Ο αλγόριθμος αυτός είναι παρόμοιος με τον Αλγόριθμο 1.1, που δόθηκε ως παράδειγμα στην Υποενότητα «1.2.2 Περιγραφή της Γλώσσας», όπου και επεξηγείται.)

ΑΛΓΟΡΙΘΜΟΣ 6.1α: ΕΥΡΕΣΗ ΜΙΚΡΟΤΕΡΟΥ ΣΤΟΙΧΕΙΟΥ ΠΙΝΑΚΑ

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A), ο δείκτης αρχής (NL) και ο δείκτης τέλους (NU) του υπό εξέταση (υπο)πίνακα.

Έξοδος: Αποθηκεύονται η θέση (X) και η τιμή (MIN) του μικρότερου στοιχείου του (υπο)πίνακα στις αντίστοιχες μεταβλητές.

```
MINSTOIXEIO(A, NL, NU)
1    X ← NL, MIN ← A[NL]          {Αρχικοποίηση μεταβλητών}
2    for I ← NL to NU             {Έλεγχος τέλους επανάληψης}
3       if MIN > A[I]             {Σύγκριση με τη μικρότερη τιμή}
4          then X ← I, MIN ← A[I] {Ενημέρωση των X και MIN}
           endif
     endfor
```

Ο αλγόριθμος για τη διάταξη επιλογής παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 6.1 (DIAT-EPILOGHS).

ΑΛΓΟΡΙΘΜΟΣ 6.1: ΔΙΑΤΑΞΗ ΕΠΙΛΟΓΗΣ

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A) και το πλήθος (N) των στοιχείων του.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα, εσωτερικά γίνεται διάταξη των στοιχείων του πίνακα κατά αύξουσα σειρά.

```
DIAT-EPILOGHS(A, N)
1    for K ← 1 to N-1             {Έλεγχος τέλους επανάληψης}
2       MINSTOIXEIO (A, K, N)     {Εύρεση μικρότερου στοιχείου}
3       A[K] ↔ A[X]               {Εναλλαγή τιμών στοιχείων}
     endfor
```

Στον αλγόριθμο αυτό χρησιμοποιούμε τη βοηθητική μεταβλητή K, που αντιπροσωπεύει το δείκτη στο πρώτο στοιχείο του τρέχοντος υποπίνακα σε κάθε πέρασμα του πίνακα. Ο αλγόριθμος, που είναι πολύ απλός στην υλοποίησή του, αποτελείται μόνο από μια διάταξη for (βήματα 1-3). Σε κάθε επανάληψη βρίσκεται το μικρότερο στοιχείο του τρέχοντος υποπίνακα, με κλήση του μερικού αλγορίθμου 6.1α (MINSTOIXEIO) (βήμα 2). Ο δείκτης αρχής του υποπίνακα καθορίζεται από την τιμή της μεταβλητής K (βήμα 1), ενώ ο δείκτης τέλους

είναι πάντα ίσος με Ν. Τέλος, γίνεται εναλλαγή του μικρότερου στοιχείου με το ευρισκόμενο στην πρώτη θέση του τρέχοντος υποπίνακα (βήμα 3).

Αν και ο αλγόριθμος διάταξης επιλογής είναι εύκολα κατανοητός και υλοποιήσιμος, δεν έχει εξίσου καλή πολυπλοκότητα χρόνου. Μια πρώτη παρατήρηση που αφορά την πολυπλοκότητα είναι ότι ο αριθμός των συγκρίσεων είναι ανεξάρτητος από την αρχική διάταξη των στοιχείων. Δηλαδή δεν υπάρχουν «καλές» και «κακές» αρχικές ακολουθίες στοιχείων και, επομένως, δεν υπάρχει χειρότερη ή καλύτερη περίπτωση.

Επίσης, ας σημειωθεί ότι σε κάθε κλήση του μερικού αλγορίθμου ΜΙΝΣΤΟΙΧΕΙΟ γίνονται n - Κ συγκρίσεις, όπου Κ η θέση (δείκτης) του πρώτου στοιχείου του τρέχοντος (υπο)πίνακα. Δηλαδή έχουμε n - 1 συγκρίσεις στο πρώτο πέρασμα, n - 2 στο δεύτερο κ.ο.κ. Άρα συνολικά γίνονται:

$$f(n) = C(n) = (n\text{-}1) + (n\text{-}2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} =$$

και επομένως είναι $f(n) = O(n^2)$.

6.3 Γρήγορη διάταξη

Η μέθοδος της *γρήγορης διάταξης* (*quicksort*) ακολουθεί μια διαφορετική φιλοσοφία από αυτή της διάταξης επιλογής. Η φιλοσοφία αυτή περιγράφεται από τη γνωστή φράση «διαίρει και βασίλευε» («divide and conquer»). Αυτό σημαίνει στην επιστήμη των Η/Υ ότι, για να λύσουμε ένα πρόβλημα, το σπάμε σε μικρότερα προβλήματα και, αν χρειάζεται, αυτά σε ακόμη μικρότερα κ.ο.κ., λύνουμε αυτά τα μικρότερα προβλήματα και η λύση του αρχικού προβλήματος παράγεται από τον κατάλληλο συνδυασμό των λύσεων των επιμέρους προβλημάτων.

Η διαδικασία της γρήγορης διάταξης έχει ως εξής:

(1) Επιλέγουμε ένα στοιχείο του πίνακα που θα χρησιμοποιηθεί ως *άξονας* (*pivot*) και αναδιατάσσουμε τα στοιχεία του πίνακα έτσι ώστε αριστερά του στοιχείου-άξονα να βρίσκονται στοιχεία μικρότερα από αυτό και δεξιά του να βρίσκονται στοιχεία μεγαλύτερα από αυτό. Έτσι, ο πίνακας διαχωρίζεται σε δύο υποπίνακες, αυτόν που βρίσκεται αριστερά και αυτόν που βρίσκεται δεξιά του στοιχείου - άξονα.

(2) Επαναλαμβάνουμε το βήμα (1) για καθέναν από τους δύο υποπίνακες που έχει περισσότερα του ενός στοιχεία και κατόπιν για τους υποπίνακες που θα προκύψουν από αυτούς κ.ο.κ., έως ότου καταλήξουμε σε υποπίνακες με το πολύ ένα στοιχείο. Ο αρχικός πίνακας τότε έχει διαταχτεί κατ' αύξουσα σειρά.

Υπάρχουν δύο βασικά σημεία στο πρώτο από τα παραπάνω βήματα. Το ένα είναι η επιλογή του στοιχείου-άξονα. Υπάρχουν διάφοροι τρόποι γι' αυτή την επιλογή, που στοχεύουν άλλοι στην απλότητα, άλλοι στην αποδοτικότητα και άλλοι στην ασφάλεια του αλγορίθμου. Ένας από τους απλούστερους τρόπους είναι να επιλέγουμε πάντα το πρώτο στοιχείο του πίνακα. Αν και δεν είναι από τους πρακτικά καλύτερους, είναι όμως κατάλληλος για την παρουσίαση του αλγορίθμου της γρήγορης διάταξης.

Το δεύτερο βασικό σημείο είναι η *κατάτμηση* ή *διαχωρισμός* (*partitioning*) του πίνακα σε δύο υποπίνακες. Και ο διαχωρισμός αυτός μπορεί να γίνει με διάφορους τρόπους, που σχετίζονται και με τον τρόπο επιλογής του άξονα. Η διαδικασία που θα ακολουθήσουμε περιγράφεται αμέσως παρακάτω και είναι τέτοια που βοηθά στην καλύτερη κατανόηση της μεθόδου.

Η διαδικασία του διαχωρισμού του πίνακα έχει ως εξής:

(1) Διαπερνάμε τον πίνακα ταυτόχρονα από τα δύο άκρα του προς το κέντρο του, με τη βοήθεια δύο δεικτών.

(2) Ο αριστερός δείκτης σταματά όταν συναντήσει στοιχείο μεγαλύτερο από τον άξονα, ενώ ο δεξιός όταν συναντήσει στοιχείο μικρότερο από αυτόν.

(3) Όταν σταματήσουν και οι δύο, γίνεται εναλλαγή των στοιχείων στα οποία σταμάτησαν και κατόπιν η διαπέραση συνεχίζεται.

(4) Τα (2) και (3) επαναλαμβάνονται μέχρις ότου οι δείκτες διασταυρωθούν, οπότε σταματά η διαπέραση και γίνεται εναλλαγή του άξονα με το στοιχείο που δείχνει ο δεξιός δείκτης.

Παράδειγμα 6.2α. *Διαχωρισμός πίνακα στη γρήγορη διάταξη*

Θεωρούμε έναν πίνακα Α που περιέχει τα στοιχεία 20, 78, 32, 12, 65, 15, 9 και 38. Η εφαρμογή της παραπάνω διαδικασίας απεικονίζεται στον παρακάτω πίνακα, όπου το στοιχείο-άξονας σημειώνεται σε κύκλο και οι δύο δείκτες, αριστερός και δεξιός, με τα σύμβολα ⬠ και ⬡, αντίστοιχα, που δείχνουν την κατεύθυνση της κίνησής τους και ταυτόχρονα και τα προς εναλλαγή στοιχεία.

Βήμα	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
1	(20)	⬠78	32	12	65	15	9	⬡38
2	(20)	⬠78	32	12	65	15	⬡9	38
3	(20)	9	⬠32	12	65	⬡15	78	38
4	(20)	9	15	⬡12	⬠65	32	78	38
	12	9	15	(20)	65	32	78	38
	αριστερός υποπίνακας				δεξιός υποπίνακας			

Η αρχική κατάσταση είναι όπως φαίνεται στο βήμα 1. Ο αριστερός δείκτης δείχνει στο δεύτερο στοιχείο, δηλαδή στο στοιχείο μετά τον άξονα, και ο δεξιός δείκτης στο τελευταίο. Οι πρώτες στάσεις των δύο δεικτών φαίνονται στο βήμα δύο. Ο αριστερός δείκτης σταματά στο στοιχείο ‘78’ (διότι 78 > 20) και ο δεξιός στο ‘9’ (διότι 9 < 20). Αυτά τα δύο στοιχεία εναλλάσσονται και προκύπτει η διάταξη του βήματος 3, όπου απεικονίζονται και οι δεύτερες στάσεις των δεικτών. Ο αριστερός δείκτης σταματά στο ‘32’ (διότι 32 > 20) και ο δεξιός στο ‘15’ (διότι 15 < 20). Τα στοιχεία αυτά εναλλάσσονται και προκύπτει η διάταξη του βήματος 4, όπου απεικονίζονται και οι τρίτες στάσεις των δεικτών, στα στοιχεία ‘65’ και ‘12’, αντίστοιχα. Παρατηρήστε ότι τώρα οι δείκτες έχουν διασταυρωθεί και, επομένως, δεν εναλλάσσονται τα στοιχεία στα οποία σταμάτησαν, αλλά ο άξονας με το στοιχείο ‘12’, που δείχνει ο δεξιός δείκτης. Έτσι, προκύπτει η τελική κατάσταση, όπως φαίνεται στην τελευταία γραμμή του πίνακα.

Οι παραγόμενοι υποπίνακες, εν γένει, είναι δυνατόν να έχουν κανένα (κενός πίνακας), ένα ή περισσότερα στοιχεία. Μόνο αυτοί που έχουν περισσότερα του ενός στοιχεία θεωρούνται προς περαιτέρω διαχωρισμό.

Ο αλγόριθμος που υλοποιεί την παραπάνω διαδικασία παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 6.2α (DIAT-DIAXORISMOS).

ΑΛΓΟΡΙΘΜΟΣ 6.2α: ΔΙΑΧΩΡΙΣΜΟΣ ΠΙΝΑΚΑ

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A), ο δείκτης αρχής (NL) και ο δείκτης τέλους (NU) του υπό εξέταση (υπο)πίνακα.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται αναδιάταξη των στοιχείων του πίνακα.

DIAT-DIAXORISMOS(A, NL, NU)

```
1   X ← A[NL], AD ← NL+1, DD ← NU          {Αρχικοποίηση μεταβλητών}
2   while AD < DD                          {Συνθήκη επανάληψης}
3     while (AD ≤ NU) and (A[AD] < X)      {Συνθήκη μετακίνησης αριστερού δείκτη}
4           AD ← AD + 1                    {Μετακίνηση αριστερού δείκτη}
      endwhile
5     while (DD ≥ NL) and (A[DD] > X)      {Συνθήκη μετακίνησης δεξιού δείκτη}
6           DD ← DD - 1                    {Μετακίνηση δεξιού δείκτη}
      endwhile
7     if AD < DD                           {Έλεγχος τερματισμού μετακίνησης}
8       then A[AD] ↔ A[DD]                 {Εναλλαγή στοιχείων}
      endif
    endwhile
9   A[NL] ↔ A[DD], P ← DD                  {Τοποθέτηση στοιχείου-άξονα}
```

Στον αλγόριθμο αυτό χρησιμοποιούμε τέσσερις βοηθητικές μεταβλητές (AD, DD, X, P). Οι δύο πρώτες (AD, DD) αντιπροσωπεύουν τον αριστερό και δεξιό δείκτη. Στην τρίτη (X) καταχωρίζεται κάθε φορά ο άξονας και στην τέταρτη (P) η θέση (δείκτης) του άξονα.

Στο πρώτο βήμα γίνεται αρχικοποίηση των μεταβλητών. Το κυρίως σώμα του αλγορίθμου αποτελείται από μια διάταξη επανάληψης while (βήματα 2-8), η οποία περιέχει άλλες δύο διατάξεις while και μια διάταξη συνθήκης if. Η πρώτη εσωτερική διάταξη while (βήματα 3-4)

υλοποιεί την κίνηση του αριστερού δείκτη, ενώ η δεύτερη (βήματα 5-6) την κίνηση του δεξιού δείκτη. Η διάταξη if (βήματα 7-8) πραγματοποιεί την εναλλαγή των στοιχείων που σταματούν οι δύο δείκτες. Τέλος, η εντολή στο βήμα 9 πραγματοποιεί την εναλλαγή του άξονα με το στοιχείο στο οποίο σταμάτησε ο δεξιός δείκτης και αποθηκεύει την τελική θέση του άξονα.

Η κλασική υλοποίηση του αλγορίθμου γρήγορης διάταξης γίνεται με τη χρήση αναδρομικής κλήσης, δηλαδή κλήσης του ίδιου του αλγορίθμου από το σώμα του, διότι η μέθοδος αυτή από τη φύση της ευνοεί την αναδρομή. Επειδή σ' αυτό το βιβλίο δεν ασχοληθήκαμε με την έννοια της αναδρομής, θα παρουσιάσουμε μια υλοποίηση του αλγορίθμου με τη χρήση της έννοιας της στοίβας. Οι στοίβες χρησιμοποιούνται, εν γένει, για την υλοποίηση της αναδρομής σε γλώσσες που δε διαθέτουν δυνατότητα αναδρομικής κλήσης. Στο τέλος, θα δώσουμε και την υλοποίηση με αναδρομή, μαζί με μια συνοπτική εξήγηση, για λόγους πληρότητας.

Ο αλγόριθμος για τη γρήγορη διάταξη παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 6.2 (DIAT-GRHGORH). Χρησιμοποιούμε δύο συνεχόμενες στοίβες, τις (SL, T) και (SU, T). Στις στοίβες αυτές αποθηκεύουμε κάθε φορά τα άκρα (όρια) των εκάστοτε παραγόμενων υποπινάκων που προορίζονται για περαιτέρω διαχωρισμό, δηλαδή έχουν περισσότερα του ενός στοιχεία. Τα μεν αριστερά (πάνω) άκρα τους αποθηκεύονται στην SL, τα δε δεξιά στην SU. Στη συνέχεια διαχωρίζεται ο υποπίνακας που τα άκρα του βρίσκονται στις κορυφές των δύο στοιβών, οι οποίες και διαγράφονται. Η διαδικασία τερματίζεται όταν αδειάσουν οι δύο στοίβες.

Καταρχήν γίνονται αρχικοποιήσεις των μεταβλητών (βήματα 1-4). Η συνθήκη στο βήμα 2 εξασφαλίζει ότι ο (υπο)πίνακας έχει περισσότερα του ενός στοιχεία, αλλιώς δεν τίθεται θέμα διάταξής του. Το κυρίως σώμα του αλγορίθμου συνιστά η επαναληπτική διάταξη while (βήματα 5-14), που εκτελείται εφόσον οι στοίβες δεν είναι άδειες ($T \neq 0$). Πρώτα εξάγονται οι κορυφές των δύο στοιβών (βήματα 6-7). Στη συνέχεια καλείται ο αλγόριθμος διαχωρισμού (βήμα 8), που διαχωρίζει τον πίνακα σε δύο υποπίνακες, πλην του άξονα. Κατόπιν υπάρχουν δύο διατάξεις if. Η πρώτη εξετάζει αν ο αριστερός υποπίνακας είναι διαχωρίσιμος, δηλαδή αν έχει περισσότερα από ένα στοιχεία (βήμα 9).

Αν είναι, τοποθετεί τα όριά του στις κορυφές των δύο στοιβών (βήματα 10-11). Θυμηθείτε ότι στον αλγόριθμο διαχωρισμού (DIAT-DIAXORISMOS) η μεταβλητή P αντιπροσωπεύει τη θέση του άξονα, επομένως P - 1 είναι το δεξιό άκρο του αριστερού υποπίνακα, του οποίου το αριστερό άκρο είναι το NL. Αντίστοιχα, η δεύτερη διάταξη if εξετάζει αν ο δεξιός υποπίνακας είναι προς διαχωρισμό (βήμα 12) και, αν είναι, τοποθετεί και αυτού τα άκρα στις δύο στοίβες (βήματα 13-14). Στην περίπτωση αυτή, P + 1 είναι το αριστερό άκρο του δεξιού υποπίνακα και NU το δεξιό του.

ΑΛΓΟΡΙΘΜΟΣ 6.2: ΓΡΗΓΟΡΗ ΔΙΑΤΑΞΗ

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A), ο δείκτης αρχής (NL) και ο δείκτης τέλους (NU) του υπό εξέταση (υπο)πίνακα.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται αναδιάταξη των στοιχείων του πίνακα κατά αύξουσα σειρά.

DIAT-GRHGORH (A, NL, NU)

```
1    T ← 0                                  {Θέση κενών στοιβών}
2    if NL < NU                             {Έλεγχος διαχωρίσιμου πίνακα}
3    then T ← 1                             {Αρχικοποίηση δείκτη στοιβών }
4         SL[1] ← NL, SU[1] ← NU            {Αρχικοποίηση κορυφών στοιβών}
     endif
5    while T ≠ 0                            {Συνθήκη επανάληψης}
6      NL ← SL[T], NU ← SU[T]               {Εξαγωγή κορυφών στοιβών }
7      T ← T - 1                            {Ενημέρωση δείκτη κορυφής}
8      DIAT-DIAXORISMOS(A, NL, NU)          {Διαχωρισμός πίνακα}
9      if NL < P - 1                        {Έλεγχος διαχωρίσιμου πίνακα}
10       then T ← T + 1                     {Ενημέρωση δείκτη κορυφών}
11            SL[T] ← NL, SU[T] ← P - 1     {Τοποθέτηση στις στοίβες}
       endif
12     if P + 1 < NU                        {Έλεγχος διαχωρίσιμου πίνακα}
13       then T ← T + 1                     {Ενημέρωση δείκτη κορυφών}
14            SL[T] ← P + 1, SU[T] ← NU     {Τοποθέτηση στις στοίβες}
       endif
     endwhile
```

Παράδειγμα 6.2β. *Γρήγορη διάταξη πίνακα*

Θεωρούμε τον πίνακα του Παραδείγματος 6.2α, με τα ίδια στοιχεία. Στη συνέχεια απεικονίζονται οι διαχωρισμοί υποπινάκων που λαμβάνουν χώρα κατά την εκτέλεση του Αλγόριθμου 6.3 (DIAT-GRHGORH) για τη διάταξη των στοιχείων του πίνακα. Τα κατακόρυφα βέλη υπονοούν εφαρμογή του Αλγόριθμου 6.3α (DIAT-DIAXORISMOS). Επίσης, απεικονίζονται τα περιεχόμενα των δύο στοιβών, SL και SU, μέσα σε τετραγωνικές παρενθέσεις (πρώτο αριστερά στοιχείο είναι η βάση και τελευταίο δεξιά η κορυφή τής κάθε στοίβας).

Βήμα	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	Στοίβες	
	20	78	32	12	65	15	9	38	SL:[1]	SU:[8]
1									SL:[ ]	SU:[ ]
	12	9	15	20	65	32	78	38	SL:[1, 5]	SU:[3, 8]
2									SL:[1]	SU:[3]
									SL:[1, 5]	SU:[3, 6]
3					32	38	65	78	SL:[1]	SU:[3]
									SL:[1]	SU:[3]
4					32	38			SL:[ ]	SU:[ ]
	9	12	15						SL:[ ]	SU:[ ]
	9	12	15	20	32	38	65	78		

Πριν από το πρώτο βήμα διαχωρισμού παρουσιάζεται η αρχική κατάσταση του προβλήματος. Έχουμε τον αρχικό πίνακα και στις δύο στοίβες είναι αποθηκευμένοι οι δείκτες των άκρων του (1, 8). Στο βήμα 1 διαγράφονται οι κορυφές των στοιβών και γίνεται ο πρώτος διαχωρισμός του πίνακα σε υποπίνακες. Το αποτέλεσμα είναι οι υποπίνακες, που φαίνονται στην επόμενη γραμμή και οι δείκτες των άκρων τους, (1, 3) και (5, 8), που αποθηκεύονται στις στοίβες. Στη συνέχεια (βήμα 2) διαγράφονται οι κορυφές των στοιβών που αντιπροσωπεύουν τα άκρα του πιο πρόσφατου δεξιού υποπίνακα, στον οποίο και πραγμα-

τοποιείται ο διαχωρισμός. Από το διαχωρισμό παράγεται ένας μόνο υποπίνακας, που χρειάζεται περαιτέρω διαχωρισμό, του οποίου οι δείκτες των άκρων (5, 6) τοποθετούνται στις στοίβες. Έτσι, συνεχίζεται η διαδικασία μέχρις ότου δεν παραχθεί νέος υποπίνακας προς διαχωρισμό, δηλαδή παραμείνουν άδειες οι στοίβες (μετά το βήμα 4). Παρατηρήστε ότι διατάσσονται πλήρως πρώτα οι δεξιοί πίνακες και μετά οι αριστεροί, που προκύπτουν από ένα διαχωρισμό. Έτσι, ο αριστερός υποπίνακας που παράχθηκε μετά το βήμα 1, ξεκίνησε να διατάσσεται αφού πρώτα είχε διαταχθεί πλήρως ο αντίστοιχος δεξιός (βήμα 4).

Η αναδρομική εκδοχή της μεθόδου της γρήγορης διάταξης παρουσιάζεται στο επόμενο πλαίσιο, ως Αλγόριθμος 6.3 (DIAT-GRHGORH-ANAD). Παρατηρήστε πόσο πιο απλή είναι η αναδρομική υλοποίηση και ότι έχει ακριβή αντιστοιχία με τη διαδικασία της γρήγορης διάταξης, όπως την περιγράψαμε στην αρχή της ενότητας αυτής. Το μειονέκτημα των αναδρομικών αλγόριθμων, εν γένει, είναι ότι απαιτούν πολύ μεγαλύτερο χώρο μνήμης κατά την εκτέλεσή τους.

ΑΛΓΟΡΙΘΜΟΣ 6.3: ΓΡΗΓΟΡΗ ΔΙΑΤΑΞΗ ΜΕ (ΑΝΑΔΡΟΜΗ)

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A), ο δείκτης αρχής (NL) και ο δείκτης τέλους (NU) του υπό εξέταση (υπο)πίνακα.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται αναδιάταξη των στοιχείων του πίνακα κατά αύξουσα σειρά.

```
DIAT-GRHGORH-ANAD (A, NL, NU)
1    if NL < NU                                {Έλεγχος διαχωρίσιμου}
2      then DIAT-DIAXORISMOS(A, NL, NU)        {Διαχωρισμός πίνακα}
3           DIAT-GRHGORH-ANAD (A, NL, DD-1)    {Διάταξη αριστερού πίνακα}
4           DIAT-GRHGORH-ANAD (A, DD+1, NU)    {Διάταξη δεξιού πίνακα}
     endif
```

Αφού γίνει έλεγχος ότι ο (υπο)πίνακας έχει περισσότερα του ενός στοιχεία (βήμα 1), ξεκινά η κυρίως διαδικασία (βήματα 2-4), που έχει ως εξής: Διαχωρίζουμε πρώτα τον τρέχοντα πίνακα σε δύο υποπίνακες (βήμα 2) και μετά διατάσσουμε πρώτα το αριστερό (βήμα 3) και μετά

το δεξιό (βήμα 4) υποπίνακα. Η διάταξη όμως κάθε υποπίνακα είναι κλήση στον ίδιο τον αλγόριθμο, που σημαίνει ότι ο κάθε υποπίνακας θα διαχωριστεί σε άλλους δύο και θα γίνει αναδρομική κλήση του αλγορίθμου για καθένα από τους δύο κ.ο.κ. Οι αναδρομικές κλήσεις σταματούν όταν παραχθεί(ούν) πίνακας(ες) με λιγότερα από δύο στοιχεία (βήμα 1). Τότε γίνεται ανακεφαλαίωση (resuming) των αναδρομών, που κρατούνται στη μνήμη, για την παραγωγή του αποτελέσματος.

Η πολυπλοκότητα χρόνου της γρήγορης διάταξης είναι, εν γένει, καλύτερη από αυτή της διάταξης επιλογής. Στη χειρότερη περίπτωση έχουν την ίδια τάξη πολυπλοκότητας. Η χειρότερη περίπτωση στη γρήγορη διάταξη συμβαίνει όταν ο πίνακας είναι ήδη διατεταγμένος. Τότε, επιλέγοντας το πρώτο στοιχείο ως άξονα, απαιτούνται n συγκρίσεις για να αποδειχθεί τελικά ότι πρέπει να παραμείνει στη θέση του. Αυτό σημαίνει ότι ο αριστερός υποπίνακας που παράγεται είναι κενός, ενώ ο δεξιός έχει n-1 στοιχεία. Κατ' αναλογία, για το πρώτο στοιχείο του δεξιού υποπίνακα (δηλαδή το δεύτερο του αρχικού), που επιλέγεται ως άξονας στη συνέχεια, απαιτούνται n-1 συγκρίσεις για να αποδειχθεί ότι πρέπει να παραμείνει στη θέση του κ.ο.κ, Δηλαδή συνολικά απαιτούνται

$$C(n) = n + (n\text{-}1) + \ldots + 2 + 1 = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

συγκρίσεις. Επομένως, είναι

$$f(n) = O(n^2).$$

Παρατηρήστε ότι απαιτούνται περισσότερες συγκρίσεις απ' ό,τι στη διάταξη επιλογής.

Στη μέση περίπτωση, αποδεικνύεται ότι είναι

$$f(n) = O(n \log n),$$

δηλαδή καλύτερη από αυτή της διάταξης επιλογής. Αυτό οφείλεται στο γεγονός ότι στη μέση περίπτωση θεωρούμε ότι παράγονται πάντα δύο υποπίνακες και όχι ένας, όπως συμβαίνει στη χειρότερη περίπτωση. Αυτό είναι και η επιδίωξη κατά την εφαρμογή του αλγορίθμου γρήγορης διάταξης, πράγμα το οποίο μπορεί να επιτευχθεί με την επιλογή κατάλληλου άξονα.

Ως μια ποιοτική απόδειξη της πολυπλοκότητας μέσης περίπτωσης, μπορούμε να πούμε τα εξής:

(1) ο 1 πίνακας παράγει $2 = 2^1$ υποπίνακες και θέτει $1 = 2^0$ στοιχείο στη θέση του (άξονας)

(2) οι 2 υποπίνακες παράγουν $4 = 2^2$ υποπίνακες και θέτουν $2 = 2^1$ στοιχεία στις θέσεις τους (άξονες)

(3) οι 4 υποπίνακες παράγουν $8 = 2^3$ υποπίνακες και θέτουν $4 = 2^2$ στοιχεία στις θέσεις τους (άξονες)

(4) οι 8 υποπίνακες παράγουν $16 = 2^4$ υποπίνακες και θέτουν $8 = 2^3$ στοιχεία στις θέσεις τους (άξονες)

...

(k) οι 2^{k-1} υποπίνακες παράγουν 2^k υποπίνακες και θέτουν 2^{k-1} στοιχεία στις θέσεις τους (άξονες)

Δηλαδή, για να διατάξουμε $2^0+2^1+2^2+ \ldots +2^{k-1} = \sum_{i=0}^{k-1} 2^i = 2^k-1 \approx 2^k$

(για σχετικά μεγάλο k) στοιχεία, απαιτούνται $k=\log(2^k)$ διαχωρισμοί. Επομένως, για n στοιχεία απαιτούνται $\log n$ διαχωρισμοί. Σε κάθε βήμα διαχωρισμού γίνονται το πολύ n συγκρίσεις, οπότε συνολικά προκύπτουν το πολύ $n \log n$ συγκρίσεις. Η μαθηματική ανάλυση του αλγορίθμου γρήγορης διάταξης δίνει στη μέση περίπτωση

$$f(n) \approx 1.39\, n \log n.$$

Άσκηση Αυτοαξιολόγησης 6.1

Ο Αλγόριθμος 6.2 είναι μια υλοποίηση της μεθόδου της γρήγορης διάταξης με τη χρήση δύο στοιβών. Η χρήση δύο στοιβών προτιμήθηκε, διότι δίνει πιο παραστατικά τη λειτουργία της μεθόδου. Υλοποιήστε τη μέθοδο γρήγορης διάταξης με τη χρήση μιας στοίβας. Εφαρμόστε τον αλγόριθμο στον ίδιο πίνακα με το Παράδειγμα 6.2β.

Άσκηση Αυτοαξιολόγησης 6.2

Ενώ ο αλγόριθμος γρήγορης διάταξης έχει καλύτερη πολυπλοκότητα χρόνου μέσης περίπτωσης από τον αλγόριθμο διάταξης επιλογής, έχει παρατηρηθεί ότι για μικρούς πίνακες, δηλαδή μικρή ακολουθία στοιχείων (μικρό n), ο αλγόριθμος διάταξης επιλογής έχει καλύτερα

Άσκηση Αυτοαξιολόγησης 6.2

αποτελέσματα. Τροποποιήστε τον Αλγόριθμο 6.2 έτσι ώστε, όταν το μέγεθος ενός υποπίνακα είναι μικρότερο από ένα όριο (π.χ. 16), να αναλαμβάνει τη διάταξή του ο αλγόριθμος διάταξης επιλογής. Ίσως χρειαστεί να τροποποιήσετε ελαφρώς και τον Αλγόριθμο 6.1.

6.4 Διάταξη σωρού

Η μέθοδος της *διάταξης σωρού* (*heapsort*) στηρίζεται στο γεγονός ότι διαδοχικές διαγραφές της ρίζας ενός σωρού οδηγούν σε εξαγωγή των στοιχείων του κατά φθίνουσα σειρά, επειδή η ρίζα του σωρού κάθε φορά είναι το μεγαλύτερο στοιχείο του. Γι' αυτό η διάταξη σωρού στηρίζεται στις πράξεις της εισαγωγής και διαγραφής στοιχείου από ένα σωρό για να διατάξει τα στοιχεία ενός πίνακα. (Ανατρέξτε στην Ενότητα «5.4 Δέντρα-Σωροί» για να ξαναθυμηθείτε τις πράξεις αυτές.)

Η διαδικασία της διάταξης σωρού περιλαμβάνει δύο φάσεις:

(1) Κατασκευάζουμε ένα δέντρο-σωρό από τα στοιχεία του πίνακα, εκτελώντας διαδοχικές εισαγωγές των στοιχείων αυτών.

(2) Εκτελούμε διαδοχικές διαγραφές της ρίζας του σωρού που κατασκευάστηκε στο (1) μέχρι να καταλήξουμε στο κενό δέντρο, αποθηκεύοντας τα διαγραφόμενα στοιχεία από το τέλος του πίνακα προς την αρχή του.

Ο αλγόριθμος που υλοποιεί τη διαδικασία αυτή παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 6.4 (DIAT-SOROU).

Στον αλγόριθμο αυτό μπορούμε να διακρίνουμε τις δύο φάσεις που αναφέραμε πιο πάνω. Η διάταξη for (βήματα 1-2) υλοποιεί την πρώτη φάση. Αυτό που κάνει είναι να καλεί τον (μερικό) αλγόριθμο εισαγωγής στοιχείου σε σωρό, H-EISAGWGH (Αλγόριθμος 5.5), N-1 φορές για την εισαγωγή των στοιχείων του πίνακα, πλην του πρώτου, που δε χρειάζεται, σε σωρό. Στην πρώτη κλήση εισάγεται το δεύτερο στοιχείο στο σωρό, που ήδη υπάρχει και αποτελείται από το πρώτο στοιχείο του πίνακα. Στη δεύτερη κλήση εισάγεται το τρίτο στοιχείο στο σωρό, που έχουν σχηματίσει τα δύο πρώτα κ.ο.κ. Στο τέλος των επαναλήψεων, τα στοιχεία του πίνακα έχουν διαταχθεί έτσι ώστε ο πίνακας να αποτελεί σωρό. Εδώ πρέπει να σημειωθεί ότι δεν απαιτείται η χρήση άλλου βοηθητικού πίνακα για την αποθήκευση του σωρού που

δημιουργείται, διότι ο σωρός αποθηκεύεται στις θέσεις των στοιχείων που χρησιμοποιούνται από τον αλγόριθμο εισαγωγής. Έτσι, γίνεται οικονομία μνήμης.

ΑΛΓΟΡΙΘΜΟΣ 6.4: ΔΙΑΤΑΞΗ ΣΩΡΟΥ

Είσοδος Ένας μη κενός μονοδιάστατος πίνακας (A) και το πλήθος των στοιχείων του (N).

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται αναδιάταξη των στοιχείων του πίνακα κατά αύξουσα σειρά.

```
DIAT-SOROU(A, N)
1     for I = 1 to N-1                    {Καθορισμός επαναλήψεων}
2         H-EISAGWGH(A, I, A[I+1])        {Εισαγωγή στοιχείου στο σωρό}
      endfor
3     while N > 1                         {Συνθήκη επανάληψης}
4         H-DIAGRAFH(A, N)                {Διαγραφή ρίζας σωρού}
5         A[N+1] ← X                      {Τοποθέτηση ρίζας στον πίνακα}
      endwhile
```

Τη δεύτερη φάση υλοποιεί η διάταξη while (βήματα 3-5). Στη διάταξη αυτή γίνονται διαδοχικές κλήσεις του (μερικού) αλγορίθμου διαγραφής της ρίζας σωρού, H-DIAGRAFH (Αλγόριθμος 5.6). Η επανάληψη σταματά, όταν μείνει μόνο ένα στοιχείο (ρίζα) στο σωρό (N = 1, βήμα 3). Σημειωτέον ότι η αλλαγή της τιμής του N συμβαίνει μέσα στον καλούμενο αλγόριθμο. Και εδώ δεν απαιτείται η χρήση βοηθητικού πίνακα, αφού η διαγραφόμενη ρίζα κάθε φορά καταλαμβάνει την τελευταία θέση του ενεργού πίνακα, η οποία εκκενώνεται λόγω της διαγραφής που προηγείται (βήμα 5). Έτσι, γίνεται οικονομία χώρου από τον αλγόριθμο.

Παράδειγμα 6.3 *Εφαρμογή διάταξης σωρού*

Θεωρούμε έναν πίνακα Α με στοιχεία: 45, 30, 50, 20, 75, 65. Η εφαρμογή του αλγορίθμου της διάταξης σωρού απεικονίζεται αναλυτικά στη συνέχεια.

Στην πρώτη φάση το σκιασμένο μέρος του πίνακα σε κάθε βήμα αντιπροσωπεύει το μέχρι τότε δημιουργηθέντα σωρό, ενώ στη δεύτερη φάση τα στοιχεία που έχουν τοποθετηθεί στη σωστή θέση.

Στην πρώτη φάση, το πρώτο στοιχείο του πίνακα θεωρείται ήδη τοποθετημένο, δεδομένου ότι δεν τίθεται θέμα εισαγωγής του. Έτσι, με το ξεκίνημα της πρώτης φάσης (I = 1) τοποθετείται το δεύτερο στοιχείο στη θέση του, θεωρώντας τον ήδη υπάρχοντα σωρό αποτελούμενο από το πρώτο στοιχείο μόνο (‘45’). Επειδή είναι μικρότερο, τοποθετείται ως αριστερό παιδί. Στο επόμενο βήμα επανάληψης (I = 2) τοποθετείται το τρίτο στοιχείο (‘50’) στο σωρό που σχημάτισαν τα δύο προηγούμενα και, επειδή είναι μεγαλύτερο όλων, τοποθετείται ως ρίζα του σωρού και τα προηγούμενα ως παιδιά του κ.ο.κ., μέχρις ότου τοποθετηθούν όλα τα στοιχεία.

Στη δεύτερη φάση ξεκινούν οι διαδοχικές διαγραφές ρίζας του σωρού που δημιουργήθηκε στην πρώτη φάση. Καταρχήν [N = 6(5)] διαγράφεται η ρίζα του σωρού (‘75’) και, επειδή λόγω της διαγραφής εκκενώνεται η τελευταία θέση του πίνακα, τοποθετείται εκεί (βήμα 5). Η τιμή του N εκτός της παρένθεσης είναι αυτή με την οποία καλείται ο αλγόριθμος διαγραφής στο βήμα 4, ενώ η εντός της παρένθεσης είναι αυτή που χρησιμοποιείται στο βήμα 5, διότι η τιμή του N ελαττώνεται κατά ένα μέσα στον αλγόριθμο διαγραφής. Αυτή η τελευταία τιμή είναι εκείνη που καθορίζει το τέλος των επαναλήψεων (βήμα 3). Νέα ρίζα του σωρού είναι τώρα το ‘65’ και ο σωρός παριστάνεται τώρα από τα πέντε πρώτα στοιχεία του πίνακα. Διαγράφεται η νέα ρίζα και τοποθετείται στον «ενεργό» πίνακα ως τελευταίο στοιχείο, που είναι το πέμπτο και έμεινε κενό (βήμα 5), καθώς γίνεται αναδιάταξη των στοιχείων του σωρού λόγω της διαγραφής. Η διαδικασία συνεχίζεται έως ότου διαγραφούν όλα τα στοιχεία, πλην του τελευταίου, που δε χρειάζεται, και προκύψει ο διατεταγμένος πίνακας.

1η ΦΑΣΗ						
I	A_1	A_2	A_3	A_4	A_5	A_6
	45	30	50	20	75	65
1	**45**	**30**	50	20	75	65
2	**50**	**30**	**45**	20	75	65
3	**50**	**30**	**45**	**20**	75	65
4	**75**	**50**	**45**	**20**	**30**	65
5	**75**	**50**	**65**	**20**	**30**	**45**

(σωρός)

2η ΦΑΣΗ						
N	A_1	A_2	A_3	A_4	A_5	A_6
6(5)	(75)	50	65	20	30	45
5(4)	(65)	50	45	20	30	**75**
4(3)	(50)	30	45	20	**65**	**75**
3(2)	(45)	30	20	**50**	**65**	**75**
2(1)	(30)	20	**45**	**50**	**65**	**75**
	20	**30**	**45**	**50**	**65**	**75**

(διατεταγμένος πίνακας)

Η διάταξη σωρού έχει καλύτερη πολυπλοκότητα χρόνου και από τη διάταξη επιλογής και από τη γρήγορη διάταξη. Συγκεκριμένα, έχει και στη χειρότερη και στη μέση περίπτωση πολυπλοκότητα

$$f(n) = O(n \log n).$$

Θα επιχειρήσουμε μια ποιοτική δικαιολόγηση αυτής της πολυπλοκότητας.

1η Φάση

Ο αριθμός των συγκρίσεων για να βρεθεί η κατάλληλη θέση για ένα στοιχείο που εισάγεται δεν μπορεί να υπερβεί το ύψος του σωρού. Επειδή ο σωρός είναι ένα πλήρες δέντρο, ισχύει για το ύψος του ό,τι αναφέραμε στην Υποενότητα «5.2.1 Ορισμός και Αναπαράσταση», δηλαδή $h \leq \log n$. Επομένως, ο αριθμός των συγκρίσεων για την εισαγωγή n στοιχείων θα είναι $C(n) \leq n\log n$ σε οποιαδήποτε περίπτωση. Άρα στη χειρότερη περίπτωση θα είναι $f(n) = O(n\log n)$.

2η Φάση

Παρατηρούμε ότι κατά τη διαγραφή της ρίζας ενός σωρού, για την αναδιάταξη του σωρού, απαιτούνται δύο συγκρίσεις για κάθε μετακίνηση του μετακινούμενου κόμβου ένα επίπεδο κάτω (μια μεταξύ των τρεχόντων παιδιών του και μια μεταξύ αυτού και του μεγαλύτερου παιδιού του). (Π.χ. ελέγξτε τη μετακίνηση του '9' ένα επίπεδο κάτω στο Σχήμα (γ) του Παραδείγματος 5.3.) Αφού για το ύψος του σωρού ισχύει $h \leq \log n$, όπου n ο αριθμός των στοιχείων, έπεται ότι απαιτούνται $2\log n$ συγκρίσεις για να βρεθεί η κατάλληλη θέση του στοιχείου. Επομένως, ο συνολικός αριθμός συγκρίσεων για τη διαγραφή n στοιχείων (που σημαίνει n αναδιατάξεις) θα είναι $C(n) = 2n\log n$. Άρα είναι $f(n) = O(n\log n)$.

Αφού σε κάθε φάση η πολυπλοκότητα είναι $f(n) = O(n\ \log n)$, τέτοιας τάξεως θα είναι και για το συνολικό αλγόριθμο, αφού ο συνδυασμός των δύο φάσεων είναι προσθετικός.

Άσκηση Αυτοαξιολόγησης 6.3

Να τροποποιήσετε τον αλγόριθμο διάταξης σωρού (Αλγόριθμος 6.4) έτσι ώστε να διατάσσει τα στοιχεία του πίνακα κατά φθίνουσα σειρά. Να εφαρμόσετε τον τροποποιημένο αλγόριθμο στον πίνακα του Παραδείγματος 6.3.

Σύνοψη

Το πρόβλημα (ή πράξη) της διάταξης αναφέρεται στην τακτοποίηση μιας ακολουθίας n δεδομένων (στοιχείων) κατά αύξουσα ή αλφαβητική σειρά. Τα στοιχεία θεωρούμε ότι βρίσκονται αποθηκευμένα σ'

έναν πίνακα στην κύρια μνήμη του Η/Υ. Αυτό το είδος της διάταξης λέγεται ***εσωτερική διάταξη***, σε αντίθεση με τη διάταξη στοιχείων που βρίσκονται στη δευτερεύουσα μνήμη του Η/Υ σε αρχεία, η οποία λέγεται ***εξωτερική διάταξη***.

Υπάρχουν διάφορες μέθοδοι (ή αλγόριθμοι) εσωτερικής διάταξης, όπως η διάταξη φυσαλίδας, η διάταξη παρεμβολής ή εισαγωγής, η διάταξη επιλογής, η γρήγορη διάταξη, η διάταξη συγχώνευσης, η διάταξη σωρού, καθώς και παραλλαγές αυτών. Στο κεφάλαιο αυτό παρουσιάσαμε τρεις από αυτές τις μεθόδους (αλγορίθμους), ως αντιπροσώπους.

Η ***διάταξη επιλογής*** χρειάζεται *n-1* περάσματα του πίνακα ή υποπινάκων του για τη διάταξη των n στοιχείων του. Σε κάθε πέρασμα ένα στοιχείο, το μικρότερο του τρέχοντα υποπίνακα, τοποθετείται στη σωστή θέση. Ο αλγόριθμος αυτός έχει πολυπλοκότητα $f(n) = O(n^2)$.

Η ***γρήγορη διάταξη*** θεωρεί ένα στοιχείο ως άξονα και διαχωρίζει τον πίνακα σε δύο υποπίνακες, ένα δεξιά του άξονα και έναν αριστερά του άξονα. Ο αριστερός υποπίνακας περιέχει στοιχεία μικρότερα του άξονα, ενώ ο δεξιός μεγαλύτερα. Αυτή η διαδικασία εφαρμόζεται στη συνέχεια στους δύο υποπίνακες που παράχθηκαν κ.ο.κ., έως ότου οι υποπίνακες εκφυλιστούν σε στοιχεία. Ο αλγόριθμος γρήγορης διάταξης έχει πολυπλοκότητα $f(n) = O(nlogn)$ στη μέση περίπτωση και $f(n) = O(n^2)$ στη χειρότερη περίπτωση.

Τέλος, η ***διάταξη σωρού*** χρησιμοποιεί ένα δέντρο-σωρό για τη διάταξη των στοιχείων. Περιλαμβάνει δύο φάσεις. Στην πρώτη φάση δημιουργεί ένα σωρό με διαδοχικές εισαγωγές των στοιχείων. Στη δεύτερη φάση εξάγει τα στοιχεία του σωρού με διαδοχικές διαγραφές των στοιχείων του. Έτσι, εξασφαλίζεται η εξαγωγή του μεγαλύτερου στοιχείου του τρέχοντος σωρού (ρίζας) κάθε φορά, που σε συνδυασμό με τον τρόπο τοποθέτησής τους στον πίνακα, έχει ως αποτέλεσμα τη διάταξη των στοιχείων κατά αύξουσα σειρά. Η διάταξη σωρού έχει πολυπλοκότητα $f(n) = O(nlogn)$ σε κάθε περίπτωση.

Οδηγός για περαιτέρω μελέτη

1. Ι. Μανωλόπουλος, *Δομές Δεδομένων*, τόμ. Α΄, ART of TEXT, 2η έκδοση, 1992.

Στο κεφάλαιο 9 του βιβλίου με τίτλο «Ταξινόμηση» γίνεται μια παρουσίαση των διαφόρων αλγόριθμων ταξινόμησης (διάταξης). Ιδιαίτερα στην Υποενότητα «9.12 Ταξινόμηση Αλγόριθμων Ταξινόμησης» θα βρείτε μια κατηγοριοποίηση των αλγόριθμων διάταξης με βάση διάφορα κριτήρια, που είναι πολύ χρήσιμη για την κατανόηση των διαφορών τους.

2. T. A. Standish, *Data Structures, Algorithm and Software Principles*, Addison-Wesley, 1994.

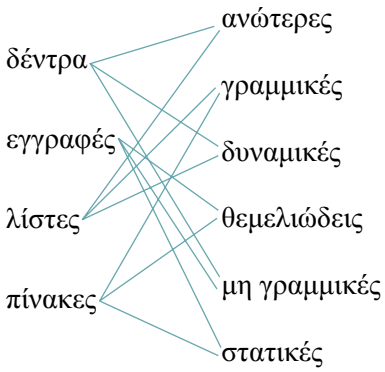
   Το κεφάλαιο 13 του βιβλίου αυτού με τίτλο «Sorting» είναι αφιερωμένο στους αλγορίθμους διάταξης με αρκετά αναλυτικό τρόπο. Ιδιαίτερα στην Ενότητα «13.1 Introduction and Motivation» υπάρχει μια ενδιαφέρουσα εισαγωγή, όπου γίνεται και μια ταξινόμηση των μεθόδων διάταξης, αρκετά χρήσιμη για την κατανόηση των διαφορών τους. Επίσης, στην Ενότητα «13.4 Divide-and-Conquer Methods», στο τμήμα «Quicksort», στο υποτμήμα «Analysis of Quicksort», υπάρχει μια λεπτομερής μαθηματική ανάλυση της πολυπλοκότητας του αλγορίθμου της γρήγορης διάταξης.

3. M. A. Weis, *Data Structures and Algorithm Analysis*, Benjamin/Cummings, 2nd Edition, 1995.

   Το κεφάλαιο 7 του βιβλίου αυτού με τίτλο «Sorting», μέχρι και την Ενότητα 7.7, παρουσιάζει τους αλγορίθμους εσωτερικής διάταξης. Εδώ μπορείτε να βρείτε υλοποιήσεις αλγόριθμων σε Pascal και πρακτικές συμβουλές γι' αυτές. Στην Υποενότητα «7.7.5 Analysis of Quicksort» υπάρχει μια ανάλυση της πολυπλοκότητας του αλγορίθμου γρήγορης διάταξης.

1.1

Απαντήσεις ασκήσεων αυτοαξιολόγησης



Αν όλες οι αντιστοιχίσεις που κάνατε είναι σωστές, μπράβο σας! Είναι βοηθητικό πριν ξεκινήσετε τη μελέτη των επόμενων κεφαλαίων να γνωρίζετε τα γενικά χαρακτηριστικά (δηλαδή τις κατηγορίες) των δομών δεδομένων που θα συναντήσετε. Είναι κάτι στο οποίο συχνά θα αναφέρεστε και το οποίο όσο προχωράτε στη μελέτη των συγκεκριμένων δομών θα γίνεται όλο και πιο ξεκάθαρο.

Αν δεν πετύχατε όλες τις αντιστοιχίσεις, μην απογοητεύεστε. Δοκιμάστε πάλι, αφού μελετήστε πιο προσεκτικά την υποενότητα «1.1.4 Κατηγορίες Δομών Δεδομένων» ή τη «Σύνοψη» του κεφαλαίου. Στο σημείο αυτό η επιτυχία είναι θέμα απομνημόνευσης, που είναι όμως βοηθητική για τη συνέχεια.

1.2

Ο αλγόριθμος δίνεται στο επόμενο πλαίσιο.

```
MESOSOROS (A, N)
1    SUM ← 0                          {Αρχικοποίηση μεταβλητών}
2    for K ← 1 to N                   {Καθορισμός αριθμού επαναλήψεων}
3      SUM ← SUM + A[K]               {Ενημέρωση αθροίσματος στοιχείων}
     endfor
4    MO ← SUM/N                       {Υπολογισμός μέσου όρου}
5    print MO                         {Επιστροφή αποτελέσματος}
```

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Αν ταυτίζεται με τον δικό σας, πλην των ονομάτων των μεταβλητών, συγχαρητήρια! Είναι ένα καλό ξεκίνημα, ιδιαίτερα αν δεν είχατε ανάλογη εμπειρία. Βέβαια είναι ένας απλός αλγόριθμος, όμως αυτό δεν μειώνει την επιτυχία σας.

Αν δεν ταυτίζεται με τον δικό σας, τότε ή χρησιμοποιήσατε διάταξη while, οπότε ξαναγράψτε τον χρησιμοποιώντας διάταξη for, ή κάπου έχετε κάνει λάθος. Μην απογοητεύεστε, κάθε αρχή και δύσκολη. Αν ξεχάσατε μόνο την αρχικοποίηση της μεταβλητής, το λάθος δεν είναι μεγάλο. Αρχικοποίηση χρειάζονται οι μεταβλητές που για την ενημέρωσή τους χρησιμοποιείται η προηγούμενη τιμή τους (όπως στο βήμα 3 για την μεταβλητή SUM). Μελετείστε πάλι το Παράδειγμα 1.4 και ξαναπροσπαθήστε.

1.3

Για να βρούμε τη συνάρτηση πολυπλοκότητας, αρκεί να μετρήσουμε τους αριθμούς των προσθέσεων και των καταχωρήσεων που πραγματοποιούνται σε κάθε βήμα του αλγορίθμου σαν συνάρτηση του αριθμού n των στοιχείων του πίνακα και να τους προσθέσουμε, ώστε να βρούμε τον συνολικό αριθμό προσθέσεων και καταχωρήσεων κατά την εκτέλεση του αλγορίθμου. Αυτή η διαδικασία παρουσιάζεται αμέσως παρακάτω.

Αριθμός προσθέσεων (βήμα 3):	n (αφού το βήμα 3 εκτελείται n φορές)
Αριθμός καταχωρήσεων (βήμα 1):	1 (αφού το βήμα 1 εκτελείται μια φορά)
Αριθμός καταχωρήσεων (βήμα 2):	n (αφού γίνονται n καταχωρήσεις στο Κ)
Αριθμός καταχωρήσεων (βήμα 3):	n (αφού το βήμα 3 εκτελείται n φορές)
Αριθμός καταχωρήσεων (βήμα 4):	1 (αφού το βήμα 4 εκτελείται μια φορά)
Σύνολο	$3n + 2$

Επομένως $f(n) = 3n + 2$.

Εδώ πρέπει να σημειώσουμε ότι στο βήμα 2 (διάταξη for) θεωρούμε ότι γίνονται μόνο καταχωρήσεις και όχι και προσθέσεις.

Αν απαντήσατε σωστά, έχετε κάνει ένα καλό ξεκίνημα. Αν όχι, ξαναπροσπαθήστε, μην απογοητεύεστε.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

1.4

α) Ο επικρατών όρος είναι ο $0.5 \cdot n^2$, διότι είναι $n^2 = n \cdot n > n \cdot \log n$ (αφού $n > \log n$). Επομένως είναι $f(n) = O(n^2)$.

Η άσκηση αυτή είναι σχετικά εύκολη, αλλά αν τα καταφέρατε μπράβο σας! Αν όχι, μελετείστε πάλι το Παράδειγμα 1.6α.

β) $$f(n) = \frac{3n}{2n} + \frac{5\log n}{2n} + \frac{1}{2n} = \frac{3}{2} \cdot 1 + \frac{5}{2} \cdot \frac{\log n}{n} + \frac{1}{2} \cdot \frac{1}{n}.$$

Επικρατών όρος είναι ο $\frac{3}{2} \cdot 1$, διότι $\frac{1}{n} < 1$ (αφού $n > 1$) και

$\frac{\log n}{n} < 1$ (αφού $n > \log n$).

Άρα, $f(n) = O(1)$.

Η άσκηση αυτή είναι δυσκολότερη από εκείνη στο (α). Αν τα καταφέρατε, σημαίνει ότι κατανοήσατε τον μηχανισμό εύρεσης της τάξης μεγέθους της συνάρτησης πολυπλοκότητας από την αναλυτική της έκφραση. Μπράβο σας! Αν όχι, δείτε πάλι το Παράδειγμα 1.6β. Ένα λάθος που πρέπει να αποφύγετε είναι το να αγνοήσετε τον σταθερό όρο « $\frac{3}{2} \cdot 1$ » και επομένως να καταλήξετε σε τάξη μεγέθους $O(\frac{\log n}{n})$.

Είναι ένα συνηθισμένο λάθος που θέλει προσοχή.

2.1

α) Ο πίνακας H είναι ένας τρισδιάστατος πίνακας, του οποίου κάθε στοιχείο παριστάνει μια επίδοση στο άθλημα και προσδιορίζεται από τρεις παραμέτρους, τον αθλητή που την έκανε, τον αγώνα στον οποίο έγινε και το έτος στο οποίο έγινε. Επομένως, πρέπει να ορίσουμε τρία σύνολα δεικτών, ένα για κάθε διάσταση, τα οποία είναι

$$I_1 = \{1, \ldots, 5\}, I_2 = \{1, 2, 3\}, I_3 = \{1988, \ldots, 1997\}.$$

Τα πλήθη στοιχείων των συνόλων αυτών είναι:

$N_1 = 5$, $N_2 = 3$ (με απλή παρατήρηση) και

Απαντήσεις ασκήσεων αυτοαξιολόγησης

$N_3 = 1997\text{-}1988 + 1 = 9 + 1 \Rightarrow N_3 = 10$ (με χρήση του τύπου 2.1).

Επομένως, ο πίνακας H είναι διαστάσεων $5 \times 3 \times 10$, δηλαδή έχει συνολικά 150 στοιχεία.

Αν τα καταφέρατε μέχρι εδώ, κάνατε ένα καλό ξεκίνημα. Αυτό είναι η βάση για τα επόμενα. Αν δεν τα καταφέρατε, ξαναπροσπαθήστε αφού ξαναδείτε τα Παραδείγματα 2.1 και 2.2. Μην απογοητεύεστε!

β) Επειδή ο πίνακας H είναι τρισδιάστατος, θα χρησιμοποιήσουμε τους τύπους (2.11) και (2.12) για να βρούμε τη ΣΑΠ. Οι τύποι αυτοί αναφέρονται γενικά σε m διαστάσεις, γι' αυτό θα βρούμε πρώτα τους αντίστοιχους τύπους για $m = 3$, που είναι

$$loc(A_{i_1,i_2,i_3}) = c_0 + c_1 i_1 + c_2 i_2 + c_3 i_3 \qquad (2.18)$$

όπου

$$c_3 = L,\ c_2 = N_3\, c_3,\ c_1 = N_2\, c$$

$$c_0 = b_A - c_1\ n_{l_1} - c_2\ n_{l_2} - c_3\ n_{l_3}$$

Οι τύποι αυτοί ισχύουν γενικά για κάθε τρισδιάστατο πίνακα.

Από την (2.19) με A ΚH και τα δεδομένα του πίνακα H υπολογίζουμε τις σταθερές της ΣΑΠ:

$$c_3 = 2,\ c_2 = 10 \times 2 = 20,\ c_1 = 3 \times 20 = 60$$

$$c_0 = 200 - 60 \times 1 - 20 \times 1 - 2 \times 1988 = -3856$$

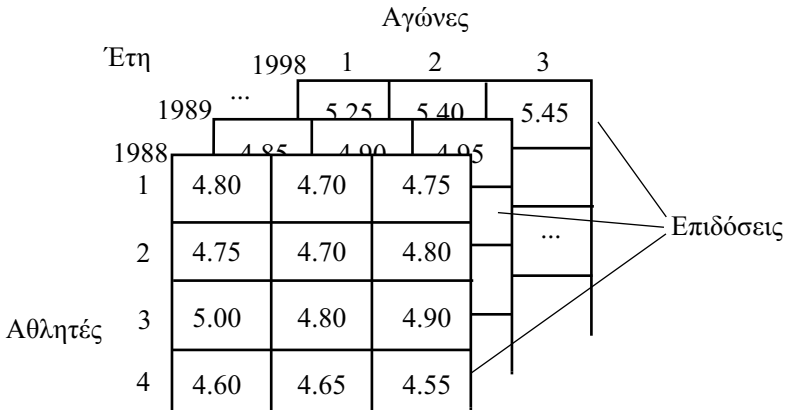
Επομένως, από την (2.18) η ΣΑΠ είναι

$$loc(H_{i_1,i_2,i_3}) = -3856 + 60\, i_1 + 20\, i_2 + 2\, i_3$$

Αυτό είναι το πιο δύσκολο βήμα της άσκησης. Η δυσκολία έγκειται, κυρίως, στην εξαγωγή του τύπου της συνάρτησης απεικόνισης ενός τρισδιάστατου πίνακα από αυτόν της συνάρτησης απεικόνισης ενός n-διάστατου πίνακα (τύπος 2.11). Αν το κάνατε σωστά, συγχαρητήρια! Αν όχι, μελετήστε πάλι την υποενότητα «2.1.5 Πολυδιάστατοι Πίνακες» και το Παράδειγμα 2.2 και προσπαθήστε ξανά, μην απογοητεύεστε.

γ) Εφαρμόζουμε την ΣΑΠ για $i_1 = 3$, $i_2 = 2$ και $i_3 = 1990$ και παίρνουμε

$$loc(H_{3,2,1990}) = -3856 + 60 \times 3 + 20 \times 2 + 2 \times 1990 = 344.$$



Δεδομένης της απάντησης στο (β), αυτό είναι το ευκολότερο βήμα της άσκησης. Εδώ δεν δικαιολογούνται λάθη, διότι πρόκειται περί απλής αντικατάστασης. Το μόνο λάθος, που ίσως δικαιολογείται, είναι η λανθασμένη αντιστοίχηση τιμών στους δείκτες, π.χ. 2, 3, 1990 αντί 3, 2, 1990. Αυτό το σημείο θέλει μια προσοχή.

Στο σχήμα, βλέπετε και μια γραφική απεικόνιση του τρισδιάστατου πίνακα (η οποία δεν ζητείται από την άσκηση). Οι τιμές είναι εικονικές και παριστάνουν επιδόσεις, π.χ. στο άλμα επί κοντώ. Παρατηρήστε ότι ένας τρισδιάστατος πίνακας θα μπορούσε να θεωρηθεί σαν μια σειρά δισδιάστατων πινάκων που θυμίζει καρτέλες ή σελίδες βιβλίου. Πράγματι, στην περίπτωση του τρισδιάστατου πίνακα, διακρίνουμε γραμμές, στήλες και *σελίδες* (*pages*).

2.2

α) Για ένα συμμετρικό πίνακα A ισχύει $A_{i,j} = A_{j,i}$ για κάθε i, j. Επομένως, όλα τα στοιχεία ενός συμμετρικού πίνακα, πλην αυτών της (κύριας) διαγωνίου, είναι διπλά. Η διαγώνιος παίζει το ρόλο του άξονα συμμετρίας. Εφόσον, λοιπόν, υπάρχουν διπλά στοιχεία θα μπορούσαν να καταχωρηθούν μια φορά. Αν υποθέσουμε ότι κρατάμε μόνο τα στοιχεία της διαγωνίου ($i = j$) και αυτά που είναι κάτω από τη διαγώνιο ($i \geq j$), τότε η κατάσταση που διαμορφώνεται είναι η ίδια με αυτή του κάτω τριγωνικού πίνακα. Άρα, το πλήθος των στοιχείων που θα αποθηκεύσουμε είναι (τύπος 2.13)

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Απαντήσεις ασκήσεων αυτοαξιολόγησης

$$\Sigma = \frac{n(n+1)}{2}.$$

Το κέρδος επομένως, είναι

$$\mathrm{K} = n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2} \qquad (2.20)$$

στοιχεία, τα οποία πολλαπλασιαζόμενα με το L μας δίνουν το κέρδος σε χώρο μνήμης.

Η απάντηση στο ερώτημα αυτό έγινε σχετικά εύκολη με την παραπάνω έξυπνη παρατήρηση της ομοιότητας του προβλήματος, με αυτό ενός κάτω τριγωνικού πίνακα. Αν κάνατε την ίδια παρατήρηση, συγχαρητήρια! Αν επαναλάβατε παρόμοια διαδικασία με αυτή του τριγωνικού πίνακα στην τελευταία υποενότητα και βρήκατε το ίδιο αποτέλεσμα, μπράβο σας! Αν δεν τα καταφέρατε, μελετήστε πιο προσεκτικά την περίπτωση του τριγωνικού πίνακα στο τέλος της υποενότητας «2.1.6 Ειδικοί Πίνακες». Η παραπάνω έξυπνη παρατήρηση κάνει εύκολη την απάντηση και στα δύο επόμενα ερωτήματα.

β) Αν υποθέσουμε ότι η αποθήκευση γίνεται κατά γραμμές, τότε έχουμε τα ίδια στοιχεία αποθηκευόμενα με τον ίδιο τρόπο, όπως και στην περίπτωση του κάτω τριγωνικού πίνακα. Επομένως, θα έχουμε την ίδια ΣΑΠ, αυτή που δίνεται από τους τύπους (2.14) και (2.15).

γ) Ο αντίστοιχος μονοδιάστατος πίνακας θα έχει το ίδιο πλήθος στοιχείων, όπως και στη περίπτωση του τριγωνικού πίνακα, που δίνεται από τον τύπο (2.16). Επίσης, ο τύπος αντιστοίχησης των στοιχείων του συμμετρικού και του ισοδύναμου μονοδιάστατου θα είναι ο (2.17).

δ) Το μόνο που αλλάζει στον συμμετρικό πίνακα σε σχέση με τον κάτω τριγωνικό είναι ότι τώρα στο στοιχείο με δείκτη k του DA δεν αντιστοιχεί μόνο το στοιχείο του A με δείκτες (i, j) αλλά και αυτό με δείκτες (j, i). Με βάση αυτή την παρατήρηση, οι ζητούμενοι αλγόριθμοι είναι οι παρακάτω:

```
ANAKTHSH (I, J, DA)
1   if I < J
2     then X ← 0
3     else K ← ((I * (I-1)/2) + J)
    endif
4   X ← DA[K]
5   print X
```

```
ENHMEROSH (I, J, X, DA)
1   if I ≥ J
2     then K ← ((I ∗ (I-1)/2) + J)
3     else K ← ((J ∗ (J-1)/2) + I)
    endif
4   DA[K] ← X
```

Το χαρακτηριστικό και στους δύο αλγορίθμους είναι ότι για τα συμμετρικά στοιχεία (*i, j*) και (*j, i*) καταλήγουμε στην ίδια τιμή για τον ισοδύναμο δείκτη *k*, λόγω εναλλαγής των I, J στα βήματα 2 και 3.

Αν τα καταφέρατε και δω, τότε πήγατε θαυμάσια στην άσκηση αυτή, που θα μπορούσαμε να την χαρακτηρίσουμε δύσκολη. Αν όχι, δεν υπάρχει λόγος ανησυχίας. Μελετήστε προσεκτικότερα το αντίστοιχο πρόβλημα του τριγωνικού πίνακα στην υποενότητα «2.1.6 Ειδικοί Πίνακες» και επανέλθετε.

2.3

Βοηθητικά Στοιχεία

Αν δεν είστε σίγουροι ότι η απάντησή σας είναι σωστή ή δεν μπορέσατε να καταλήξετε σε ολοκληρωμένη απάντηση, πριν προχωρήσετε στην μελέτη της απάντησης που δίνεται εδώ, μελετήστε τον παρακάτω Αλγόριθμο 2.1α, που αποτελεί μια παραλλαγή του Αλγορίθμου 2.1. Αφού τον μελετήσετε, προσπαθήστε με βάση αυτόν να σχεδιάσετε τον ζητούμενο αλγόριθμο. Ο αλγόριθμος αυτός έχει περισσότερα συγγενή στοιχεία με αυτόν της απάντησης.

Στον αλγόριθμο αυτό χρησιμοποιούμε επί πλέον τη μεταβλητή EURESH για καταχώρηση της επιτυχίας (TRUE) ή της αποτυχίας (FALSE) της αναζήτησης. Η λογική του αλγορίθμου είναι η ίδια με αυτή του αρχικού, απλώς χρησιμοποιείται αυτή η επί πλέον μεταβλητή που κάνει τον αλγόριθμο πιο κατανοητό.

Απάντηση

Η βασική ιδέα στην οποία στηρίζεται ο ζητούμενος αλγόριθμος είναι ότι η αναζήτηση δεν πρέπει να σταματά όταν βρει το πρώτο ζητούμενο στοιχείο, αλλά σε κάθε περίπτωση, συνεχίζει μέχρι το τελευταίο

Απαντήσεις ασκήσεων αυτοαξιολόγησης

στοιχείο του πίνακα, για ενδεχόμενη ανεύρεση και άλλων ίδιων στοιχείων. Αν κάνατε αυτή τη σκέψη, τότε κάνατε μια καλή αρχή και συλλάβατε την ουσία της διαφοράς από το βασικό αλγόριθμο. Αν όχι, μην απογοητεύεστε, διότι αυτή είναι ουσιαστικά η πρώτη άσκηση τροποποίησης αλγορίθμου που αναλάβατε να φέρετε εις πέρας μόνοι σας και χρειάζεστε κάποια εξοικείωση. Ο τροποποιημένος αλγόριθμος παρουσιάζεται στο παρακάτω πλαίσιο ως Αλγόριθμος Α2.3 (GRAM-ANAZHTHSH-OLWN).

ΑΛΓΟΡΙΘΜΟΣ 2.1α: ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ (ΠΑΡΑΛΛΑΓΗ)

Είσοδος: Ένας πίνακας (Α), το μέγεθος του πίνακα (Ν) και η προς αναζήτηση τιμή (Χ).
Έξοδος: Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα σε περίπτωση επιτυχίας ή μήνυμα αποτυχίας. Εσωτερικά δεν συμβαίνει τίποτα.

```
GRAM-ANAZHTHSH(A, N, X)
1   I ← 1 , EURESH ← FALSE                {Αρχικοποίηση μεταβλητών}
2   while (not EURESH) and (I ≤ N)        {Έλεγχος τέλους επανάληψης}
3       if A[I]=X
4         then EURESH ← TRUE              {Καταγραφή επιτυχίας}
5         else I ← I + 1                  {Ενημέρωση δείκτη θέσης}
        endif
    endwhile
6   if EURESH=TRUE                        {Αναγνώριση αποτελέσματος}
7     then print I                        {Επιστροφή θέσης στοιχείου}
8     else print 'ΑΠΟΤΥΧΙΑ'               {Επιστροφή μηνύματος αποτυχίας}
    endif
```

Τώρα, η συνθήκη λήξης της αναζήτησης σχετίζεται με το αν έχουμε φτάσει στο τέλος του πίνακα (I ≤ N) (βήμα 2) και όχι και με την επιτυχία της αναζήτησης, όπως ο αρχικός. Επίσης, μια άλλη βασική σκέψη είναι ότι πρέπει να επιστρέφει στην έξοδο τη θέση κάθε στοιχείου που έχει την ζητούμενη τιμή μόλις το βρίσκουμε, αλλιώς πρέπει να βρούμε ένα τρόπο να αποθηκεύουμε τις θέσεις των στοιχείων αυτών και να τις επιστρέφουμε στο τέλος. Αυτό όμως είναι πολύ δύσκολο, διότι δεν είναι γνωστός ο αριθμός των τέτοιων στοιχείων εκ των προτέρων. Αν κάνατε και αυτή τη σκέψη, τότε πάτε πολύ καλά.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Αν όχι, μην απογοητεύεστε, είναι αρχή ακόμη.

Παρατηρήστε ότι τώρα ο δείκτης I αυξάνεται σε κάθε επανάληψη του σώματος της διάταξης while (βήματα 2-6), ανεξαρτήτως αν έχουμε επιτυχία ή όχι στο τρέχον βήμα αναζήτησης. Γι' αυτό το αντίστοιχο βήμα (βήμα 6), βρίσκεται εκτός της πρώτης διάταξης if (βήματα 3-5), σε αντίθεση με τον βασικό Αλγόριθμο 2.1. Αν και αυτό το κάνατε σωστά, τότε συγχαρητήρια! Αν όχι, πρέπει να είστε προσεκτικοί σε τέτοια λάθη. Μελετήστε προσεκτικά τη διαφορά αυτή στους δύο αλγορίθμους.

ΑΛΓΟΡΙΘΜΟΣ Α2.3: ΠΟΛΛΑΠΛΗ ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ

Είσοδος:Ένας πίνακας (Α), το μέγεθος του πίνακα (Ν) και η προς αναζήτηση τιμή (Χ).
Έξοδος: Εξωτερικά, επιστρέφει τις θέσεις του στοιχείου στον πίνακα, σε περίπτωση επιτυχίας, ή μήνυμα αποτυχίας. Εσωτερικά , δεν συμβαίνει τίποτα.

GRAM-ANAZHTHSH-OLWN(A, N, X)

```
1    I ← , EURESH ← FALSE          {Αρχικοποίηση μεταβλητών}
2    while I ≤ N                   {Έλεγχος τέλους επανάληψης}
3      if A[I]=X                   {Σύγκριση στοιχείου}
4        then EURESH ← TRUE        {Καταχώρηση επιτυχίας}
5             print I              {Επιστροφή θέσης στοιχείου}
       endif
6      I ← I + 1                   {Ενημέρωση δείκτη θέσης}
     endwhile
7    if EURESH=FALSE               {Αναγνώριση αποτελέσματος}
8      print 'ΑΠΟΤΥΧΙΑ'             {Επιστροφή μηνύματος αποτυχίας}
     endif
```

2.4

Ο ζητούμενος αλγόριθμος παρουσιάζεται στο επόμενο πλαίσιο και αποτελεί επέκταση του Αλγορίθμου 2.2. Η βασική διαδικασία είναι η ακόλουθη: Συγκρίνουμε τη δοθείσα τιμή με τις τιμές των δύο στοιχείων που οι δείκτες τους τριχοτομούν τον πίνακα. Αν κάποιο από αυτά είναι το ζητούμενο, σταματά η αναζήτηση (επιτυχία). Αν όχι, επιλέγουμε από τα τρία τμήματα αυτό στο οποίο πρέπει να βρίσκεται το ζητούμενο στοιχείο και συνεχίζουμε με την ίδια διαδικασία.

ΑΛΓΟΡΙΘΜΟΣ Α2.4: ΤΡΙΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

Είσοδος: Ένας πίνακας (Α), το μέγεθος του πίνακα (Ν) και η προς αναζήτηση τιμή (Χ).

Έξοδος: Εξωτερικά, επιστρέφει τη θέση του στοιχείου στον πίνακα, σε περίπτωση επιτυχίας, ή μήνυμα αποτυχίας. Εσωτερικά, δεν συμβαίνει τίποτα.

TRIAD-ANAZHTHSH (A, N, X)

```
1    NL ← 1, NU ← N                                  {Αρχικοποίηση μεταβλητών}
2    T1 ← [(2*NL+NU)/3], T2 ← [(NL+2*NU)/3]          {Δείκτες τριχοτόμησης}
3    while (A[T1] ≠ X) and (A[T2] ≠ X) and (NL ≤ NU) {Συνθήκη επανάληψης}
4       if A[T2] < X                                 {Σύγκριση στοιχείων}
5         then NL ← T2 + 1                           {Ενημέρωση κάτω ορίου}
6         else if A[T1] < X                          {Σύγκριση στοιχείων}
7                 then NL ← T1+1, NU ← T2-1          {Ενημέρωση ορίων}
                  else NU ← T1-1                     {Ενημέρωση πάνω ορίου}
               endif
       endif
9      T1 ← [(2*NL+NU)/3], T2 ← [(NL+2*NU)/3]       {Εύρεση νέων ορίων }
    endwhile
10   if A[T1] = X                                    {Αναγνώριση αποτελέσματος}
11     then print T1                                 {Επιστροφή θέσης στοιχείου}
12     else if A[T2] = X                             {Αναγνώριση αποτελέσματος}
13            then print T2                          {Επιστροφή θέσης στοιχείου}
14            else print 'ΑΠΟΤΥΧΙΑ'                  {Μήνυμα αποτυχίας}
           endif
    endif
```

Τα βασικά σημεία για τη σχεδίαση του αλγορίθμου είναι τα εξής: Πρώτον, η εύρεση των τύπων υπολογισμού των δύο δεικτών που τριχοτομούν τον πίνακα (ή το τρέχον τμήμα του πίνακα). Αυτό γίνεται ως εξής Αν NL και NU είναι ο μικρότερος και μεγαλύτερος δείκτης του πίνακα (τμήματος) αντίστοιχα, τότε ο μικρότερος από τους δύο δείκτες τριχοτόμησης θα είναι T1 = NL + (NU - NL)/3 = (2NL + NU)/3, ενώ ο μεγαλύτερος T2 = NU - (NU - NL)/3 = (NL + 2NU)/3 και τα τρία τμήματα είναι τα [NL, T1] (αριστερό), [T1, T2] (μεσαίο) και [T2, NU] (δεξιό). Αν καταλήξατε σ' αυτά τα αποτελέσματα, κάνατε ένα

καλό πρώτο βήμα. Αν όχι, ελέγξτε πάλι τους υπολογισμούς σας.

Δεύτερον, πρέπει να προσδιορίσουμε τη νέα συνθήκη της διάταξης επανάληψης while. Η επανάληψη τερματίζεται, α) όταν έχουμε εξαντλήσει τον πίνακα, δηλ. NL > NU (αποτυχία), β) όταν βρούμε το ζητούμενο στοιχείο, δηλαδή όταν A[T1] = X ή A[T2] = X. Επομένως, το σώμα της διάταξης while θα εκτελείται ενόσω A[T1] ≠ X και A[T2] ≠ X και NL ≤ NU. Αν μια από τις σχέσεις αυτές δεν αληθεύει, τότε σταματά η εκτέλεση. Αν προσδιορίσατε σωστά και τις συνθήκες, μπράβο σας! Αν όχι, μελετήστε με προσοχή το επόμενο σημείο, ακριβώς παρακάτω.

Τρίτον, πρέπει να προσδιορίσουμε τη λογική επιλογής του ενός από τα τρία τμήματα και τις μικροαλλαγές στα άκρα του. Η λογική αυτή έχει ως εξής:

α) Αν X > A[T2], τότε επιλέγουμε το δεξιό τμήμα, διότι το X δεν μπορεί να βρίσκεται στα προηγούμενα, αφού ο πίνακας είναι διατεταγμένος. Στην περίπτωση αυτή, το νέο υπό εξέταση τμήμα είναι το [T2 + 1, NU], διότι το A[T2] έχει ελεγχθεί ως προς την ισότητα, στη συνθήκη της διάταξης while.

β) Αν X > A[T1] και X < A[T2], τότε επιλέγουμε το μεσαίο τμήμα και, το νέο υπό εξέταση τμήμα είναι το [T1 + 1, T2 - 1] πάλι, διότι τα A[T1], A[T2] έχουν ελεγχθεί ως προς την ισότητα, στη συνθήκη της διάταξης επανάληψης.

γ) Στην απομένουσα περίπτωση, X < A[T1], επιλέγουμε το αριστερό τμήμα, και το νέο τμήμα είναι το [NL, T1 - 1].

Η λογική αυτή αντικατοπτρίζεται στον παραπάνω αλγόριθμο. Αν και σ' αυτό το σημείο είστε επιτυχείς, συγχαρητήρια! Αν όχι, μην απογοητεύεστε, αυτή είναι μια, σχετικά, δύσκολη άσκηση. Μελετήστε τη προσεκτικά και τρέξτε τον αλγόριθμο βήμα-βήμα για ένα συγκεκριμένο πίνακα, για να κατανοήσετε τη λειτουργία του.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

3.1

Η διαφορά του προβλήματος που θέτει η άσκηση αυτή με εκείνο που αντιμετωπίζει ο βασικός αλγόριθμος 3.5 είναι ότι εδώ, δε δίνεται η θέση εισαγωγής του στοιχείου, διότι δεν απαιτείται, και είναι ένα από τα επί μέρους προβλήματα που πρέπει να λύσει ο αλγόριθμος που θα

Απαντήσεις ασκήσεων αυτοαξιολόγησης

σχεδιάσουμε. Αυτή είναι μια βασική παρατήρηση, που αν την κάνατε, είστε στον σωστό δρόμο. Αν όχι, τότε σκεφθείτε ποια διαφορά θα υπήρχε μεταξύ των δύο προβλημάτων.

Η βασική σκέψη για τη σχεδίαση του αλγορίθμου είναι ότι πρέπει να αποτελείται από δύο τμήματα. Το πρώτο τμήμα θα προσδιορίζει τη θέση εισαγωγής με βάση τη διάταξη και το δεύτερο θα πραγματοποιεί την εισαγωγή με τον ίδιο τρόπο όπως και ο Αλγόριθμος 3.2. Αν σκεφθήκατε έτσι, τότε μπράβο σας! Έχετε κάνει ένα μεγάλο βήμα για την απάντηση. Αν όχι, τότε πιθανόν να καταλήξετε σε μια πιο πολύπλοκη ή και λανθασμένη απάντηση. Όμως, μην απογοητεύεστε, διότι μπορεί να υπάρχουν περισσότεροι από ένας αλγόριθμοι, αφ' ενός, και τα λάθη είναι αυτά που μας κάνουν να μαθαίνουμε, αφ' ετέρου.

Ο προσδιορισμός της θέσης εισαγωγής επιτυγχάνεται με διαδοχικές συγκρίσεις του X (της δοθείσας τιμής) με τις τιμές των στοιχείων της λίστας. Η θέση εισαγωγής είναι αυτή για την οποία, πρώτη φορά θα συμβεί, η τιμή του στοιχείου να είναι μεγαλύτερη από τη δοθείσα, ενώ προηγουμένως ήταν μικρότερη.

Ο αλγόριθμος παρουσιάζεται στο παρακάτω πλαίσιο ως Αλγόριθμος A3.1 (DIAT-SL-EISAGWGH). Το πρώτο τμήμα του αποτελείται από τα βήματα 1-3. Η συνθήκη της διάταξης while στο τμήμα αυτό, αφ' ενός μεν εξασφαλίζει ότι θα σταματήσει η επανάληψη, δηλαδή η αύξηση του μετρητή I (βήμα 3), όταν βρεθεί η θέση εισαγωγής (δηλαδή όταν $X < L[I]$), αφ' ετέρου δε ότι δεν θα υπερβεί το μέγεθος της λίστας (δηλαδή ότι δεν θα γίνει $I > T$). Αν το σχεδιάσατε παρόμοια, συγχαρητήρια! Αυτό ήταν ουσιαστικά το νέο στοιχείο του αλγορίθμου. Αν όχι, τότε ενδεχομένως να σχεδιάσατε ένα πιο πολύπλοκο αλγόριθμο, που όμως μπορεί να δουλεύει σωστά.

ΑΛΓΟΡΙΘΜΟΣ Α3.1: ΔΙΑΤΕΤΑΓΜΕΝΗ ΕΙΣΑΓΩΓΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), το μέγεθός του (N), το τρέχον μέγεθος της αντίστοιχης διατεταγμένης λίστας (T) και η προς εισαγωγή τιμή (X).

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, γίνονται οι απαραίτητες μετακινήσεις στοιχείων και ενημερώνεται ο δείκτης τέλους της λίστας σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
DIAT-SL-EISAGWGH(L, N, T, X)
1   I ← 1                          {Αρχικοποίηση μετρητή
2   while (I ≤ T) and (L[I] < X)   {Έλεγχος τέλους επανάληψης}
3     I ← I+1                      {Ενημέρωση μετρητή}
    endwhile
4   K ← T                          {Αρχικοποίηση μετρητή}
5   while K ≥ I                    {Έλεγχος τέλους επανάληψης}
6     L[K+1] ← L[K]                {Μετακίνηση στοιχείου}
7     K ← K-1                      {Ενημέρωση μετρητή}
    endwhile
8   L[I] ← X                       {Καταχώρηση στοιχείου}
9   T ← T+1                        {Ενημέρωση του δείκτη τέλους}
```

Το δεύτερο τμήμα, βήματα 4-9, είναι αντιγραφή του Αλγορίθμου 3.2, πλην του ελέγχου υπερχείλισης (βήματα 1-2), που παραλείπεται για λόγους απλότητας. Δεν απαιτείται καμία αλλαγή. Δουλεύει σωστά και στις ακραίες περιπτώσεις, όταν δηλαδή το X είναι είτε μικρότερο είτε μεγαλύτερο από όλα τα στοιχεία της λίστας. Αν ο αλγόριθμος που σχεδιάσατε είναι διαφορετικός, τότε πιθανόν να είναι πιο πολύπλοκος, λόγω επί πλέον ελέγχων. Για να βεβαιωθείτε για την ορθότητα του αλγορίθμου σας, τρέξτε παράλληλα τους δύο αλγορίθμους σε συγκεκριμένα παραδείγματα, που να καλύπτουν όλες τις περιπτώσεις, για να δείτε αν συμπίπτουν τα αποτελέσματά τους.

3.2

Ο ζητούμενος αλγόριθμος είναι ένας συνδυασμός αναζήτησης και διαγραφής, όπως και ο Π3.3. Βέβαια, η αναζήτηση τώρα δεν αφορά την πρώτη εμφάνιση του ζητούμενου στοιχείου, αλλά όλες. Επομένως,

Απαντήσεις ασκήσεων αυτοαξιολόγησης

όσον αφορά την αναζήτηση, θα στηριχθούμε στον Αλγόριθμο Α2.3 και, όσον αφορά την διαγραφή, στον Αλγόριθμο 3.3. Αν κάνατε αυτές τις βασικές παρατηρήσεις, τότε μπράβο σας! Έχετε κάνει ένα μεγάλο βήμα προς τη σωστή σχεδίαση. Αν όχι, μην απογοητεύεστε. Πρέπει να συνηθίσετε να σκέφτεστε πρώτα σε πιο αφηρημένο επίπεδο και μετά να μπαίνετε στις λεπτομέρειες του αλγορίθμου.

ΑΛΓΟΡΙΘΜΟΣ Α3.2: ΠΟΛΛΑΠΛΗ ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΕΧΟΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ένας πίνακας (L), το τρέχον μέγεθος της αντίστοιχης συνεχόμενης λίστας (T) και η τιμή των προς διαγραφή κόμβων (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα αποτυχίας όταν δεν υπάρχουν στοιχεία, αλλιώς τίποτα. Εσωτερικά, γίνονται μετακινήσεις στοιχείων λόγω διαγραφών σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
SL-DIAGRAFH-OLWN(L, T, X)
1    I ← 1 , EURESH ← FALSE                {Αρχικοποίηση μεταβλητών}
2    while I ≤ T                           {Έλεγχος τέλους επανάληψης}
3        if L[I]=X                         {Έλεγχος ζητουμένου στοιχείου}
4          then EURESH ← TRUE              {Καταχώρηση επιτυχίας}
5            for K = I to T-1              {Καθορισμός ορίων επανάληψης}
6              L[K] ← L[K+1]               {Μετακίνηση μια θέση αριστερά}
             endfor
7            T ← T-1                       {Ενημέρωση δείκτη τέλους }
8            else I ← I + 1                {Αύξηση δείκτη θέσης κατά ένα}
         endif
     endwhile
9    if EURESH=FALSE                       {Αναγνώριση αποτελέσματος}
10     print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'           {Μήνυμα αποτυχίας}
     endif
```

Η βασική ιδέα στην οποία στηρίζεται ο ζητούμενος αλγόριθμος είναι ότι κατά την αναζήτηση, μόλις βρίσκουμε ένα στοιχείο με τιμή ίση με τη δοθείσα το διαγράφουμε πρώτα και, κατόπιν συνεχίζουμε την αναζήτηση του επόμενου στοιχείου κ.ο.κ. Αν σκεφθήκατε με παρόμοιο τρόπο, συγχαρητήρια! Αυτό αποτελεί το βασικό κορμό του αλγορίθμου. Αν όχι, τότε πιθανό να μπλεχτήκατε σε λεπτομέρειες που, ενδε-

Απαντήσεις ασκήσεων αυτοαξιολόγησης

χομένως, να κάνουν τον αλγόριθμό σας πιο πολύπλοκο και, ενδεχομένως, να μην δουλεύει σωστά σε όλα του τα τμήματα. Προσπαθήστε να μάθετε να σκέφτεστε πιο απλά.

Ο ζητούμενος αλγόριθμος παρουσιάζεται στο παραπάνω πλαίσιο, ως Αλγόριθμος Α3.3 (SL- DIAGRAFH-OLWN). Ουσιαστικά, είναι ο Αλγόριθμος Α2.3 (με L όπου Α), όπου όμως, κάθε φορά που βρίσκουμε ένα ζητούμενο στοιχείο (βήμα 3), ξεκινά μια διαδικασία διαγραφής του (βήματα 5-7), που είναι παρόμοια με αυτή του βασικού αλγορίθμου διαγραφής (Αλγόριθμος 3.3). Όπως και στον Αλγόριθμο Α2.3, χρησιμοποιούμε την τοπική μεταβλητή EURESH.

3.3

Η πρώτη σκέψη που μπορούμε να κάνουμε για τη σχεδίαση αυτού του αλγορίθμου είναι ότι πρέπει να βρούμε, κατ' αρχήν, τους δείκτες στον Κ-οστό και Μ-οστό κόμβο της λίστας, για να μπορούμε να τους προσπελάσουμε. Αν κάνατε αυτή τη σκέψη, τότε κάνατε μια πολύ καλή αρχή. Αν όχι, ξαναμελετήστε τους αλγορίθμους που αναφέρονται σε συνδεμένες λίστες για να διαπιστώσετε ότι η προσπέλαση ενός κόμβου απαιτεί την γνώση του δείκτη στον κόμβο.

Για τη συνέχεια, θεωρούμε ότι Κ < Μ. Για να βρούμε τους ζητούμενους δείκτες θα πρέπει, ξεκινώντας από την αρχή της λίστας, να περάσουμε από ένα-ένα κόμβο μετρώντας τη θέση του από την αρχή της λίστας και να κρατήσουμε τους δείκτες στον Κ-οστό και Μ-οστό κόμβο. Αυτή η διαδικασία δεν χρειάζεται να προχωρήσει πέραν του Μ-οστού κόμβου, διότι δεν έχει νόημα. Αν κάνατε παρόμοια σκέψη, μπράβο σας! Είναι το δεύτερο βήμα για τη σχεδίαση του αλγορίθμου. Αν όχι, ρίξτε μια ματιά στο Παράδειγμα 3.4. Μια τέτοια διαδικασία για την εύρεση του δείκτη στον Κ-οστό κόμβο υπάρχει στον Αλγόριθμο Π3.4.

Στηριζόμενοι στις παραπάνω σκέψεις, σχεδιάζουμε το πρώτο τμήμα του Αλγορίθμου Α3.3, που είναι ο ζητούμενος αλγόριθμος και παρουσιάζεται στο παρακάτω πλαίσιο. Το πρώτο αυτό τμήμα αποτελείται βασικά από μια διάταξη επανάληψης while (βήματα 2-6) που βασίζεται στην αντίστοιχη διάταξη του Αλγορίθμου Π3.4. Τώρα, βέβαια, τη θέση του δείκτη PK έχει πάρει ο PM, λείπει ο δείκτης PI, διότι δεν

Απαντήσεις ασκήσεων αυτοαξιολόγησης

ενδιαφερόμαστε για διαγραφή και έχει προστεθεί μια διάταξη if (βήματα 3-4) για την αποθήκευση του δείκτη στον Κ-οστό κόμβο που θα τον συναντήσουμε πριν τον Μ-οστό. Αν υλοποιήσατε τη σκέψη σας έτσι ή παρόμοια, συγχαρητήρια! Είναι μια πολύ καλή λύση. Αν όχι, δεν πειράζει, μπορεί να έχετε βρει μια πιο πολύπλοκη, αλλά σωστή λύση.

Το αποτέλεσμα λοιπόν του πρώτου τμήματος του αλγορίθμου είναι η εύρεση των δεικτών στους κόμβους με θέσεις Κ και Μ από την αρχή και η αποθήκευσή τους στις βοηθητικές μεταβλητές (δείκτη) PK και PM, αντίστοιχα. Το δεύτερο τμήμα του αλγορίθμου δεν απομένει παρά να πραγματοποιεί την εναλλαγή των στοιχείων των κόμβων αυτών. Αυτό δεν μπορεί να πραγματοποιηθεί σε ένα βήμα. Απαιτείται μια βοηθητική μεταβλητή (Χ) σαν ενδιάμεσος χώρος αποθήκευσης. Έτσι, καταχωρούμε κατ' αρχήν, την τιμή του στοιχείου (του κόμβου) Κ στην Χ (βήμα 8). Κατόπιν, καταχωρούμε την τιμή του στοιχείου Μ σαν τιμή του στοιχείου Κ (βήμα 9). Τέλος, καταχωρούμε την τιμή του Χ σαν τιμή του στοιχείου (του κόμβου) Μ (βήμα 10). Όλα αυτά είναι έτσι, αν το μήκος της λίστας είναι μεγαλύτερο του Μ. Αν δεν είναι, τότε θα γίνει PM = NIL και θα ενεργοποιηθεί το βήμα 11 για την επιστροφή μηνύματος λάθους. Αν το κάνατε έτσι, συγχαρητήρια! Αν όχι, συγκρίνετε τη δική σας λύση με αυτή του Αλγορίθμου Α3.3 και βρείτε τις διαφορές ή τα λάθη σας.

ΑΛΓΟΡΙΘΜΟΣ Α3.3: ΕΝΑΛΛΑΓΗ ΣΤΟΙΧΕΙΩΝ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ

Είσοδος: Ο δείκτης αρχής μιας απλά συνδεδεμένης λίστας (L) και δύο ακέραιοι διατεταγμένοι αριθμοί (K, M).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα λάθους σε περίπτωση ανυπαρξίας K-οστού ή M-οστού κόμβου, αλλιώς τίποτα. Εσωτερικά, γίνονται μεταβολές στη λίστα σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
DL-ENALLAGH(L, K, M)
1    PM ← L , I ← 1                          {Αρχικοποίηση}
2    while (PM ≠ NIL) and (I ≤ M)            {Έλεγχος τέλους}
3      if I = K                              {Ανίχνευση κόμβου K}
4        then PK ← PM                        {Ενημέρωση δείκτη K}
       endif
5      PM ← DEIKTHS(KOMBOS(PM))              {Ενημέρωση δείκτη M}
6      I ← I+1                               {Ενημέρωση μετρητή}
     endwhile
7    if (PM ≠ NIL)                           {Έλεγχος αποτελέσματος}
8      then X ← STOIXEIO(KOMBOS(PK))
9           STOIXEIO(KOMBOS(PK)) ← STOIXEIO(KOMBOS(PM))
10          STOIXEIO(KOMBOS(PM)) ← X
11     else print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'        {Μήνυμα λάθους}
     endif
```

3.4

Το πρόβλημα αυτό είναι μια παραλλαγή του προβλήματος που λύνει ο Αλγόριθμος 3.6. Τώρα, δε μας δίνεται ο δείκτης στον προηγούμενο κόμβο, αλλά η τιμή του προς διαγραφή στοιχείου και φυσικά, ο δείκτης αρχής της λίστας, αφού πρόκειται για συνδεδεμένη λίστα. Επομένως, μια βασική παρατήρηση είναι ότι πρέπει να βρούμε πρώτα το στοιχείο με τη δοθείσα τιμή και μετά να το διαγράψουμε. Αν κάνατε αυτή την παρατήρηση, τότε κάνατε ένα καλό πρώτο βήμα.

Η βασική ιδέα που πηγάζει από την παραπάνω παρατήρηση είναι ότι ο αλγόριθμος θα έχει δύο τμήματα. Στο πρώτο, αναζητούμε τον κόμβο του οποίου το στοιχείο του έχει τη δοθείσα τιμή και στο δεύτερο τον

Απαντήσεις ασκήσεων αυτοαξιολόγησης

διαγράφουμε. Είναι, δηλαδή, ο αλγόριθμος ένας συνδυασμός αναζήτησης και διαγραφής. Αν κάνατε και αυτή τη σκέψη, μπράβο σας! Αν όχι, προσπαθήστε να συνηθίσετε να σκέφτεστε ότι ένας νέος ζητούμενος αλγόριθμος πιθανόν να είναι συνδυασμός δύο άλλων βασικών αλγορίθμων. Μην απογοητεύεστε! Από τα λάθη μας μαθαίνουμε.

ΑΛΓΟΡΙΘΜΟΣ Α3.4: ΔΙΑΓΡΑΦΗ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΛΙΣΤΑ (ΠΑΡΑΛΛΑΓΗ)

Είσοδος: Ο δείκτης αρχής μιας απλά συνδεδεμένης λίστας (L) και η τιμή του προς διαγραφή στοιχείου (X).

Έξοδος: Εξωτερικά, επιστρέφει μήνυμα αποτυχίας σε περίπτωση που δεν υπάρχει στοιχείο, αλλιώς τίποτα. Εσωτερικά, γίνονται αλλαγές στη λίστα σε περίπτωση επιτυχίας, αλλιώς τίποτα.

```
DL-DIAGRAFH-S(L, X)
1    P ← L                                                    {Αρχικοποίηση}
2    while (P ≠ NIL) and (STOIXEIO(KOMBOS(P)) ≠ X)            {Έλεγχος}
3      PI ← P                                                 {Ενημέρωση}
4      P ← DEIKTHS(KOMBOS(P))                                 {Ενημέρωση}
     endwhile
5    if P = NIL                                               {Έλεγχος}
6      then print 'ΑΝΥΠΑΡΚΤΟ ΣΤΟΙΧΕΙΟ'                        {Αποτυχία}
7      else   DEIKTHS(KOMBOS(PI)) ← DEIKTHS(KOMBOS(P))       {Επιτυχία}
     endif
```

Ο αλγόριθμος παρουσιάζεται στο παραπάνω πλαίσιο ως Αλγόριθμος Α3.4 (DL-DIAGRAFH-S). Η τοπική μεταβλητή P αντιπροσωπεύει τον δείκτη στον τρέχοντα κόμβο, ενώ η PI στον προηγούμενο του τρέχοντα κόμβου. Ξεκινάμε από την αρχή της λίστας (βήμα 1: P ← L) και προχωρούμε από κόμβο σε κόμβο εξετάζοντας αν το στοιχείο του κόμβου είναι ίσο με X μέσω της διάταξης while (βήματα 2-4). Αν δεν είναι (δηλαδή αν STOIXEIO(KOMBOS(P)) ≠ X) και δεν έχουμε φτάσει στο τέλος της λίστας (δηλαδή αν PINIL), τότε καταχωρούμε την τρέχουσα τιμή της P στην PI (PI←P) και ενημερώνουμε την P με την τιμή του δείκτη στον επόμενο κόμβο (P←DEIKTHS(KOMBOS(P))). Αν μια από τις δύο σχέσεις της συνθήκης της διάταξης επανάληψης αληθεύει, τότε σταματά η επανάληψη και μεταβαίνουμε στη διάταξη if που ακολουθεί (βήματα 5-7). Η διάταξη while αντιπροσωπεύει το πρώτο τμήμα του αλγορίθμου, ενώ η διάταξη if το δεύτερο.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Η χρήση της μεταβλητής PI είναι απαραίτητη, διότι ο βασικός αλγόριθμος διαγραφής 3.6 απαιτεί τη γνώση του δείκτη στον προηγούμενο του προς διαγραφή κόμβου. Έτσι, το δεύτερο τμήμα του αλγορίθμου δεν είναι παρά ο Αλγόριθμος 3.6 με μικρές αλλαγές. Πρώτον, ο έλεγχος εγκυρότητας μετατρέπεται σε έλεγχο επιτυχίας ή αποτυχίας της αναζήτησης (P = NIL). Δεύτερον, το βήμα 3 του βασικού αλγορίθμου δεν χρειάζεται, διότι τώρα η τιμή του P είναι έτοιμη από την εκτέλεση του πρώτου τμήματος του αλγορίθμου.

4.1

Όπως και στην περίπτωση της συνεχόμενης στοίβας (Παράδειγμα 4.1), ο ζητούμενος αλγόριθμος έχει δύο τμήματα. Το πρώτο αφορά τη διαγραφή των κόμβων και στηρίζεται στην επαναληπτική εκτέλεση του κορμού (βήμα 4) του βασικού αλγορίθμου διαγραφής (Αλγόριθμος 4.4). Το δεύτερο τμήμα αφορά την έξοδο του κατάλληλου αποτελέσματος, ανάλογα με την περίπτωση (στοίβα με τουλάχιστον K στοιχεία ή με λιγότερα). Αν προσεγγίσατε έτσι τον ζητούμενο αλγόριθμο, ξεκινήσατε πολύ καλά. Αν όχι, ξαναδείτε το Παράδειγμα 4.1, που είναι παρόμοιο.

Ο ζητούμενος αλγόριθμος είναι ο παρακάτω Αλγόριθμος A4.1 (DS-DIAGRAFH-KP). Το πρώτο τμήμα συνιστά η διάταξη while (βήματα 2-5), που επαναλαμβάνει το κύριο βήμα του αλγορίθμου διαγραφής (βήμα 4). Επιπρόσθετα όμως, υπάρχει η ενημέρωση του μετρητή I (βήμα 5), που μετρά τον αριθμό των επαναλήψεων (και επομένως, των διαγραμμένων κόμβων), και η ενημέρωση του δείκτη PI, που αντιπροσωπεύει τον δείκτη στον προηγούμενο του τρέχοντος (προς διαγραφή) κόμβο. Ο PI είναι απαραίτητος για να είναι δυνατή η επιστροφή στην έξοδο του στοιχείου του τελευταίου διαγραφέντος κόμβου (βήμα 8). Αν το κάνατε παρόμοια, μπράβο σας! Αν δεν χρησιμοποιήσατε τον δείκτη PI, τότε πιθανότατα θα έχει πρόβλημα ο αλγόριθμός σας στην επιστροφή του κατάλληλου αποτελέσματος ή θα είναι πιο πολύπλοκος.

ΑΛΓΟΡΙΘΜΟΣ Α4.1: ΔΙΑΓΡΑΦΗ Κ ΣΤΟΙΧΕΙΩΝ ΣΕ ΣΥΝΔΕΔΕΜΕΝΗ ΣΤΟΙΒΑ

Είσοδος : Ο δείκτης κορυφής μιας μη κενής συνδεδεμένης στοίβας (S) και ένας ακέραιος αριθμός (K).

Έξοδος : Εξωτερικά, επιστρέφει την τιμή του στοιχείου του τελευταίου διαγραφέντος κόμβου και/ή μήνυμα άδεια στοίβας. Εσωτερικά, διαγράφονται οι K πρώτοι ή όλοι οι κόμβοι της λίστας.

SS-DIAGRAFH -KP (S,K)

```
1   I ← 1                                  {Αρχικοποίηση μετρητή}
2   while (S / NIL) and (I/K)              {Συνθήκη επανάληψης}
3      PI ← S                              {Ενημέρωση βοηθητικού δείκτη}
4      S ← DEIKTHS(KOMBOS(S))              {Ενημέρωση δείκτη κορυφής}
5      I ← I+1                             {Ενημέρωση μετρητή}
    endwhile
6   if I = K + 1                           {Έλενχος επιτυχίας}
7      then print STOIXEIO(KOMBOS(PI))     {Επιστροφή K -οστού στοιχείο}
8      else print STOIXEIO(KOMBOS(PI))     {Επιστροφή τελευταίου στοιχείο}
9           print 'ΑΔΕΙΑ ΣΤΟΙΒΑ'           {Μήνυμα άδειας στοίβας}
    endif
```

Η επανάληψη σταματά όταν είτε I > K είτε S = NIL (δηλαδή αδειάσει η στοίβα). Τότε, ξεκινά το δεύτερο τμήμα του αλγορίθμου, που αποτελείται από μια διάταξη if (βήματα 6-9). Κατ΄ αρχήν, εξετάζεται αν η περίπτωση ήταν η ομαλή (βήμα 6), δηλαδή, αν υπήρχαν K κόμβοι στη στοίβα. Αν ναι, τότε επιστρέφει το στοιχείο του K-οστού κόμβου (βήμα 7). Αλλιώς, δηλαδή στην περίπτωση που η στοίβα άδειασε πριν εξαχθούν K κόμβοι, επιστρέφει το στοιχείο του τελευταίου εξαχθέντος κόμβου (βήμα 8) και το μήνυμα άδειας στοίβας (βήμα 9). Αν και αυτό το τμήμα το σχεδιάσατε παρόμοια, συγχαρητήρια! Αν η διαφορά σας είναι κάποιος επιπλέον έλεγχος που ενδεχομένως δεν είναι απαραίτητος, όπως π.χ., ο έλεγχος για άδεια στοίβα μετά το βήμα 7, οπότε θα έχετε καταλήξει σε ένα σύνθετο if, δεν είναι τίποτα σπουδαίο, αφού ο αλγόριθμος είναι σωστός. Αν παραλείψατε όμως κάτι, τότε εξετάστε αν οι παραπάνω περιπτώσεις ικανοποιούνται από τον αλγόριθμό σας. Ξανακοιτάξτε το Παράδειγμα 4.1, που είναι ο αντίστοιχος αλγόριθμος για συνεχόμενη στοίβα.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

4.2

Χρησιμοποιούμε μια στοίβα που τα στοιχεία της είναι αριστερές και δεξιές παρενθέσεις όλων των τύπων. Η λύση (αλγόριθμος) για τον έλεγχο μιας αριθμητικής έκφρασης δίνεται στη συνέχεια σε φυσική γλώσσα.

Η αριθμητική έκφραση σαρώνεται από αριστερά προς τα δεξιά και κάθε αριστερή παρένθεση οποιουδήποτε τύπου που συναντάται, εισάγεται στη στοίβα. Οι άλλοι χαρακτήρες ή αριθμοί παραβλέπονται.

Όταν συναντήσουμε μια δεξιά παρένθεση, τότε:

1. Αν η στοίβα δεν είναι κενή, συγκρίνουμε τη δεξιά παρένθεση με την αριστερή που βρίσκεται στην κορυφή της στοίβας και

   1.1 αν είναι ίδιου τύπου, εξάγουμε την αριστερή παρένθεση από τη στοίβα και συνεχίζουμε τη σάρωση

   1.2 αν δεν είναι του ίδιου τύπου, η έκφραση έχει πρόβλημα (χρήση αριστερής και δεξιάς παρένθεσης διαφορετικού τύπου)

2. Αν η στοίβα είναι κενή, η έκφραση έχει πρόβλημα (περισσότερες δεξιές από αριστερές παρενθέσεις)

3. Όταν φθάσουμε στην τελευταία δεξιά παρένθεση της έκφρασης και ισχύει το 1.1, τότε

   3.1 αν η στοίβα καταλήξει άδεια, η έκφραση είναι σωστή

   3.2 αν δεν καταλήξει άδεια, τότε η έκφραση έχει πρόβλημα (περισσότερες αριστερές από δεξιές παρενθέσεις)

Αν δώσατε την ίδια, από πλευράς λογικής, λύση, συγχαρητήρια! Αν όχι, συγκρίνετε τις δύο λύσεις μέσω παραδειγμάτων που να καλύπτουν όλες τις δυνατές περιπτώσεις. Αν δυσκολευτήκατε, ξαναμελετήστε την εφαρμογή της στοίβας στην εκτίμηση αριθμητικών εκφράσεων (υποενότητα 4.4.2).

4.3

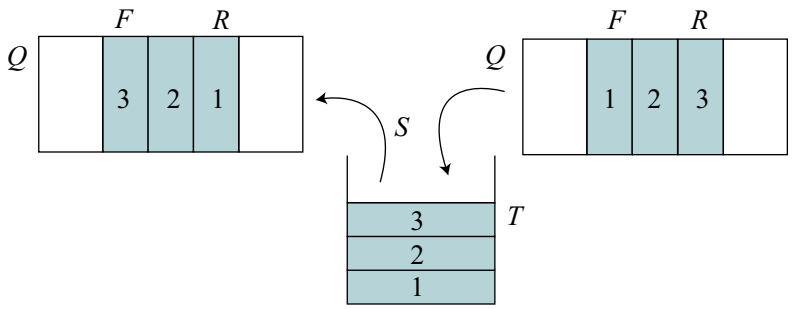
Υπόδειξη

Χρησιμοποιήστε μια κενή στοίβα σαν ενδιάμεσο. Αδειάστε δηλαδή την ουρά στη στοίβα και κατόπιν αδειάστε τη στοίβα πίσω στην ουρά.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Απάντηση

Η βασική ιδέα λοιπόν είναι να αδειάσουμε την ουρά σε μια άδεια στοίβα και στη συνέχεια τη στοίβα στην άδεια πλέον ουρά. Αυτό, εκ του φυσικού, αντιστρέφει τη σειρά των στοιχείων στην ουρά. Παρατηρήστε το παρακάτω σχήμα που απεικονίζει αυτή τη διαδικασία.



Επομένως, ο ζητούμενος αλγόριθμος αποτελείται από δύο τμήματα που το καθένα υλοποιεί ένα από τα δύο στάδια της παραπάνω διαδικασίας. Αν κάνατε αυτή τη σκέψη, χωρίς να δείτε την υπόδειξη, συγχαρητήρια! Αν σκεφτήκατε να χρησιμοποιήσετε ένα κενό πίνακα σαν ενδιάμεσο όπου θα αποθηκεύσετε τα στοιχεία της ουράς και μετά θα τα επανεισάγετε με αντίστροφη σειρά, και πάλι μπράβο σας, διότι ουσιαστικά η σκέψη σας αυτή απέχει ένα πολύ μικρό βήμα από την παραπάνω.

Ο ζητούμενος αλγόριθμος είναι ο Αλγόριθμος Α4.3 (SQANASTROFH) στο παρακάτω πλαίσιο. Χρησιμοποιούμε σαν βοηθητικά στοιχεία δύο τοπικές μεταβλητές-μετρητές (I και J) και μια στοίβα (S, T). Το πρώτο τμήμα αποτελείται από μια επαναληπτική διάταξη for (βήματα 2-4). Σε κάθε επανάληψη, αυξάνεται κατά ένα ο δείκτης κορυφής της στοίβας (βήμα 3) και καταχωρείται το τρέχον πρώτο στοιχείο της ουράς στη στοίβα, που προσδιορίζεται από την τρέχουσα τιμή του I, στην κενή θέση της στοίβας που δείχνει ο δείκτης κορυφής (βήμα 4). Στο τέλος των επαναλήψεων, όλα τα στοιχεία της ουράς έχουν μεταφερθεί στη στοίβα. Αν υλοποιήσατε αυτό το τμήμα με τον ίδιο τρόπο μπράβο σας! Αν χρησιμοποιήσατε διάταξη while, ο αλγόριθμός σας μπορεί να είναι σωστός, αρκεί να έχετε συμπεριλάβει στο σώμα του τα βήματα 3 και 4 και ένα επιπλέον βήμα για την ενημέρωση του I. Επίσης, θα πρέπει να αρχικοποιείτε το I στην τιμή F πριν τη διάταξη και η συνθήκη επανάληψης να είναι η $I \leq R$.

ΑΛΓΟΡΙΘΜΟΣ Α4.3: ΑΝΑΣΤΡΟΦΗ ΣΥΝΕΧΟΜΕΝΗΣ ΟΥΡΑΣ

Είσοδος: Ένας πίνακας (Q) και οι δείκτες αρχής (F) και τέλους (R) της αντίστοιχης συνεχόμενης ουράς.

Έξοδος: Εξωτερικά, δεν επιστρέφει τίποτα. Εσωτερικά, αναστρέφεται η σειρά των στοιχείων της ουράς.

```
SQ-ANASTROFH (Q, F, R)
1    T ← 0                          {Αρχικοποίηση δείκτη κορυφή }
2    for I ← F to R                 {Όρια επανάληψης }
3      T ← T+1                      {Ενημέρωση δείκτη κορυφής}
4      S[T] ← Q[I]                  {Καταχώρηση στη στοίβα}
     endfor
5    TX ← T, FX ← F                 {Αρχικοποίηση βοηθητικών δεικτών}
6    for J ← 1 to TX                {Όρια επανάληψης }
7      Q[FX] ← S[T]                 {Εισαγωγή στην ουρά}
8      T ← T-1                      {Ενημέρωση δείκτη κορυφής}
9      FX ← FX+1                    {Ενημέρωση βοηθητικού δείκτη}
     endfor
```

Το δεύτερο τμήμα του αλγορίθμου είναι μια άλλη διάταξη for (βήματα 6-9) όπου γίνεται η αντίστροφη διαδικασία. Εδώ, χρησιμοποιούμε δύο βοηθητικές μεταβλητές που αρχικοποιούνται πριν τη διάταξη (βήμα 5). Η πρώτη (TX) χρησιμοποιείται στο όριο επανάληψης (βήμα 6) και η δεύτερη (FX) σαν προσωρινός δείκτης της τρέχουσας αρχής της ουράς. Σε κάθε επανάληψη, η τρέχουσα κορυφή της στοίβας καταχωρείται στο τέλος της ουράς (βήμα 7) και ενημερώνονται ο δείκτης κορυφής (βήμα 8) και ο προσωρινός δείκτης αρχής (βήμα 9). Αν και αυτό το σχεδιάσατε με τον ίδιο τρόπο, συγχαρητήρια! Αν χρησιμοποιήσατε διάταξη while, πρέπει να προσέξετε τα ίδια που ελέχθησαν για το πρώτο τμήμα. Αν δεν το κάνατε με τον ίδιο τρόπο, ελέγξτε αν χρησιμοποιήσατε βοηθητικές μεταβλητές ή όχι και γιατί.

5.1

Ξεκινάμε τη σχεδίαση του δέντρου από τη ρίζα του προς τα φύλλα με βάση την εξής διαδικασία:

Απαντήσεις ασκήσεων αυτοαξιολόγησης

(1) Βρίσκουμε τη ρίζα του δέντρου. Η ρίζα του δέντρου είναι το πρώτο στοιχείο στην ακολουθία των κόμβων που προκύπτει από την προδιατεταγμένη διαπέραση, διότι, ως γνωστόν, σ' αυτό τον τρόπο διαπέρασης επισκεπτόμαστε πρώτα τη ρίζα. Άρα η ρίζα του δέντρου είναι το 'Z'.

Αν ξεκινήσατε έτσι, τότε κάνατε μια καλή αρχή. Αν όχι, τότε ρίξτε μια ματιά στους τρόπους διαπέρασης ενός δέντρου.

(2) Βρίσκουμε το αριστερό παιδί της ρίζας Χρησιμοποιούμε καταρχήν την ακολουθία των κόμβων της ενδοδιατεταγμένης διαπέρασης για να βρούμε τους κόμβους του αριστερού υποδέντρου του 'Z'. Επειδή η ενδοδιατεταγμένη διαπέραση ακολουθεί τη σειρά επίσκεψη αριστερού δέντρου - επίσκεψη ρίζας - επίσκεψη δεξιού δέντρου, οι αριστερά του 'Z' ευρισκόμενοι κόμβοι στην ενδοδιατεταγμένη ακολουθία, δηλαδή οι 'Ε', 'Α', 'Γ' και 'Κ', αποτελούν το αριστερό υποδέντρο. Το αριστερό παιδί του 'Z', τώρα, είναι ο πρώτος κόμβος που συναντάμε στην προδιατεταγμένη ακολουθία των κόμβων του αριστερού δέντρου και είναι ο 'Α'. Επομένως, ο 'Α' είναι το αριστερό παιδί του 'Z'.

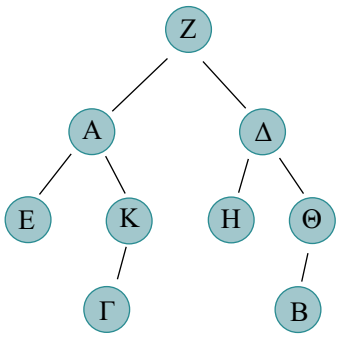
Αυτό είναι ένα πιο δύσκολο βήμα. Αν το κάνατε σωστά, συγχαρητήρια! Αν όχι, μην απογοητεύεστε. Χρειάζεται λίγη εξάσκηση παραπάνω για να συνηθίσετε το χειρισμό των δέντρων.

(3) Βρίσκουμε το δεξί παιδί της ρίζας. Με τρόπο αντίστοιχο με το (2) βρίσκουμε ότι το δεξί υποδέντρο του 'Z' αποτελείται από τους κόμβους 'Η', 'Δ', 'Β' και 'Θ' και ότι ο 'Δ' είναι η ρίζα του δεξιού υποδέντρου, δηλαδή το δεξί παιδί του 'Z', διότι αυτός είναι, από τους κόμβους του δεξιού δέντρου, που συναντάμε πρώτον στην προδιατεταγμένη ακολουθία.

Δεδομένου του προηγούμενου βήματος, αυτό είναι άμεσο επακόλουθο. Αν δεν το καταφέρατε, τότε ξαναμελετήστε προσεκτικά το προηγούμενο βήμα.

Θεωρώντας τους 'Α' και 'Δ' ως ρίζες, επαναλαμβάνουμε τα παραπάνω βήματα και βρίσκουμε και τα δικά τους παιδιά κ.ο.κ. Τελικά, το δέντρο που προκύπτει είναι το παρακάτω.

Αν καταλήξατε σ' αυτό το δέντρο, έστω και με άλλη σειρά σκέψεων, μπράβο σας! Αν όχι, μην απογοητεύεστε. Ξαναπροσπαθήστε.

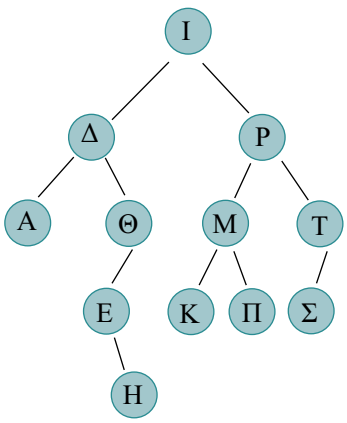


Απαντήσεις ασκήσεων αυτοαξιολόγησης

5.2

(α) Το δέντρο κατασκευάζεται σταδιακά, αν εφαρμόσετε τον αλγόριθμο εισαγωγής στοιχείου διαδοχικά για κάθε στοιχείο της ακολουθίας. Τα βήματα της όλης διαδικασίας έχουν ως εξής:

- Το πρώτο στοιχείο (‘Ι’), επειδή το δέντρο είναι στην αρχή κενό, τοποθετείται στη ρίζα του δέντρου.
- Το δεύτερο στοιχείο (‘Ρ’), επειδή είναι Ρ > Ι, τοποθετείται ως δεξιό παιδί του ‘Ι’.
- Το τρίτο στοιχείο (‘Δ’), επειδή είναι Δ < Ι, τοποθετείται ως αριστερό παιδί του ‘Ι’.



Αυτές οι τρεις πρώτες εφαρμογές του αλγορίθμου εισαγωγής είναι η βάση για τη συνέχεια. Αν τις καταφέρατε, κάνατε μια καλή αρχή. Αν όχι, μην απογοητεύεστε. Ξαναπροσπαθήστε.

- Στην εισαγωγή του τέταρτου στοιχείου (‘Θ’), επειδή είναι Θ < Ι,

Απαντήσεις ασκήσεων αυτοαξιολόγησης

προχωρούμε, καταρχήν, στο αριστερό υποδέντρο. Εκεί, επειδή είναι Θ > Δ, το 'Θ' τοποθετείται ως δεξιό παιδί του 'Δ'.

- Στο πέμπτο στοιχείο ('Τ'), επειδή είναι Τ > Ι, προχωρούμε στο δεξιό υποδέντρο. Εκεί, επειδή είναι Τ > Ρ, το 'Τ' τοποθετείται ως δεξιό παιδί του 'Ρ'.
- Στην εισαγωγή του επόμενου στοιχείου 'Ε', επειδή είναι Ε < Ι, προχωρούμε, καταρχήν, στο αριστερό υποδέντρο. Εκεί, επειδή είναι Ε > Δ, προχωρούμε στο δεξιό υποδέντρο του 'Δ'. Στη συνέχεια, επειδή είναι Ε < Θ, το 'Ε' τοποθετείται ως αριστερό παιδί του 'Θ'.
- Στο επόμενο στοιχείο 'Μ', επειδή είναι Μ > Ι, προχωρούμε, καταρχήν, στο δεξιό υποδέντρο. Εκεί, επειδή είναι Μ < Ρ, το 'Μ' τοποθετείται ως αριστερό παιδί του 'Ρ'.

Αν κάνατε σωστά και αυτά τα βήματα, πήγατε πολύ καλά. Αν δεν τα καταφέρατε κάπου, ξαναπροσπαθήστε. Η διαδικασία αυτή συνεχίζεται και για τα άλλα στοιχεία της ακολουθίας, οπότε προκύπτει το δυαδικό δέντρο αναζήτησης, που φαίνεται πιο πάνω. Αν το αποτέλεσμά σας είναι το ίδιο, μπράβο σας! Αν όχι, ξαναπροσπαθήστε. Δεν είναι δύσκολο, είναι θέμα εξοικείωσης.

(β) Ο προς διαγραφή κόμβος 'Δ' έχει δύο παιδιά (περίπτωση 3 της διαδικασίας διαγραφής). Επομένως, χρειάζεται να βρούμε τον κόμβο που είναι ο επόμενός του στην ακολουθία των κόμβων που αντιστοιχεί στην ενδοδιατεταγμένη διαπέραση του δέντρου. Η ακολουθία αυτή είναι η εξής: 'Α', 'Δ', 'Ε', 'Η', 'Θ', 'Ι', 'Κ', 'Μ', 'Π', 'Ρ', 'Σ', 'Τ'.

Εδώ χρειάζεται προσοχή, ώστε να γράψετε σωστά την ενδοδιατεταγμένη ακολουθία των κόμβων. Θυμηθείτε ότι αυτή καταλήγει στην αλφαβητική (αύξουσα) σειρά των στοιχείων. Μ' αυτό τον τρόπο μπορείτε να ελέγξετε αν κατασκευάσατε σωστά το δέντρο. Αν το καταφέρατε, μπράβο σας! Αν όχι, εξασκηθείτε στην αναγραφή των ακολουθιών διαπέρασης. Χρησιμοποιήστε και την προηγούμενη άσκηση αυτοαξιολόγησης για το σκοπό αυτό.

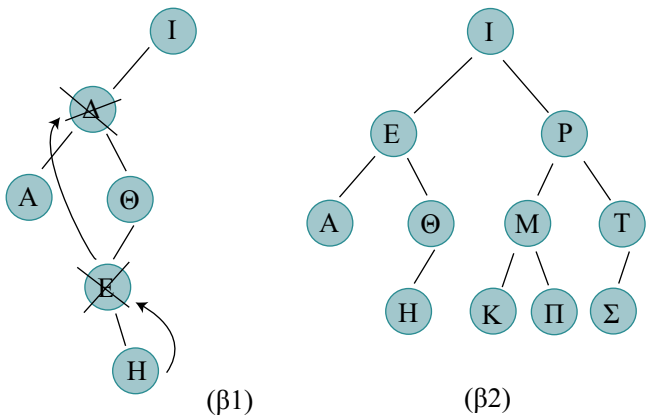
Ο επόμενος κόμβος του 'Δ' στην ακολουθία αυτή είναι ο 'Ε'. Στη συνέχεια γίνονται οι παρακάτω μεταβολές:

(α) Διαγράφεται ο 'Ε'. Επειδή έχει ένα παιδί ('Η'), διαγραφή του 'Ε' σημαίνει ότι το παιδί του καταλαμβάνει τη θέση του (περίπτωση 1 διαγραφής).

Απαντήσεις ασκήσεων αυτοαξιολόγησης

(β) Διαγράφεται ο 'Δ' και τη θέση του καταλαμβάνει ο 'Ε'.

Οι αλλαγές αυτές απεικονίζονται στο παρακάτω Σχήμα (β1), όπου χρησιμοποιείται μόνο το εμπλεκόμενο αριστερό υποδέντρο, και το τελικό δέντρο στο Σχήμα (β2).



(β1) (β2)

5.3

(α) Το δοθέν δέντρο είναι σωρός, διότι (α1) είναι δυαδικό δέντρο, δηλαδή κάθε κόμβος του έχει το πολύ δύο παιδιά, (α2) είναι πλήρες δέντρο, διότι όλα τα επίπεδα, πλην του τελευταίου, έχουν το μέγιστο αριθμό κόμβων και όλοι οι τερματικοί κόμβοι (τελευταίο επίπεδο) είναι τοποθετημένοι όσο το δυνατόν προς τα αριστερά, και (α3) η τιμή του στοιχείου κάθε κόμβου είναι μεγαλύτερη από τις τιμές των στοιχείων των παιδιών του.

Αυτό ήταν μια εύκολη απάντηση, που πηγάζει απευθείας από τον ορισμό του σωρού. Αν δεν την απαντήσατε, ξαναμελετήστε τον ορισμό του σωρού.

Η συνεχόμενη αναπαράσταση του σωρού αντικατοπτρίζεται στον παρακάτω πίνακα.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_9	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}
H	93	84	73	69	74	38	52	13	51	12	48	20			

Το μέγεθος του πίνακα είναι ίσο με το μέγιστο αριθμό κόμβων του πλήρους δέντρου, δηλαδή $2^{h+1} - 1 = 2^{3+1} - 1 = 2^4 - 1 = 16 - 1 = 15$, όπου $h = 3$ το ύψος του δέντρου. Το μέγεθος του πίνακα είναι μεγαλύτερο

Απαντήσεις ασκήσεων αυτοαξιολόγησης

από τον αριθμό κόμβων του δέντρου. Αν το υπολογίσατε σωστά, πήγατε καλά μέχρις εδώ. Αν όχι, ανατρέξτε στην Υποενότητα «5.2.1 Ορισμοί και Αναπαράσταση» για να θυμηθείτε τα περί συνεχόμενης αναπαράστασης.

Η τοποθέτηση των στοιχείων στον πίνακα γίνεται ως εξής: Πρώτο στοιχείο τοποθετείται αυτό της ρίζας (‘93’) και στη συνέχεια τα στοιχεία του αριστερού και δεξιού παιδιού της (‘84’, ‘73’). Κατόπιν τοποθετούνται με τη σειρά (αριστερό, δεξιό) τα στοιχεία των παιδιών του ‘84’ (‘69’, ‘74’). Μετά τα στοιχεία των παιδιών του ‘73’ (‘38’, ‘52’) κ.ο.κ. Αν τοποθετήσατε σωστά τα στοιχεία, μπράβο σας! Αν όχι, μην απογοητεύεστε. Δεν είναι δύσκολο, απλώς χρειάζεται εξοικείωση, ξαναπροσπαθήστε.

(β) Σύμφωνα με τη διαδικασία εισαγωγής ενός στοιχείου σε σωρό (Υποενότητα «5.4.2 Εισαγωγή»), απαιτούνται τα εξής βήματα:

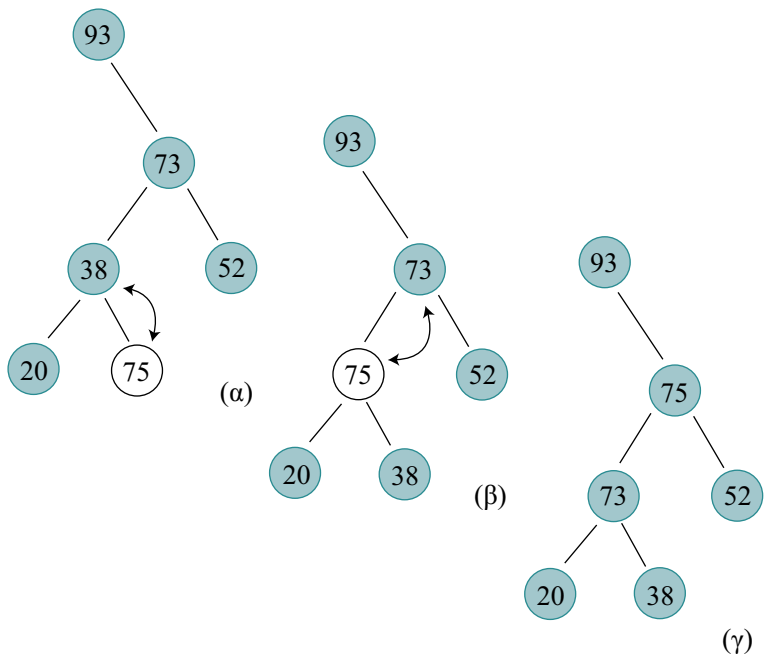
(β1) Τοποθετούμε το προς εισαγωγή στοιχείο (‘75’) στο τέλος του δέντρου (πίνακα), ώστε να εξακολουθούμε να έχουμε πλήρες δέντρο.

(β2) Μετακινούμε διαδοχικά τον κόμβο του στοιχείου προς τη ρίζα έως ότου δημιουργηθεί σωρός:

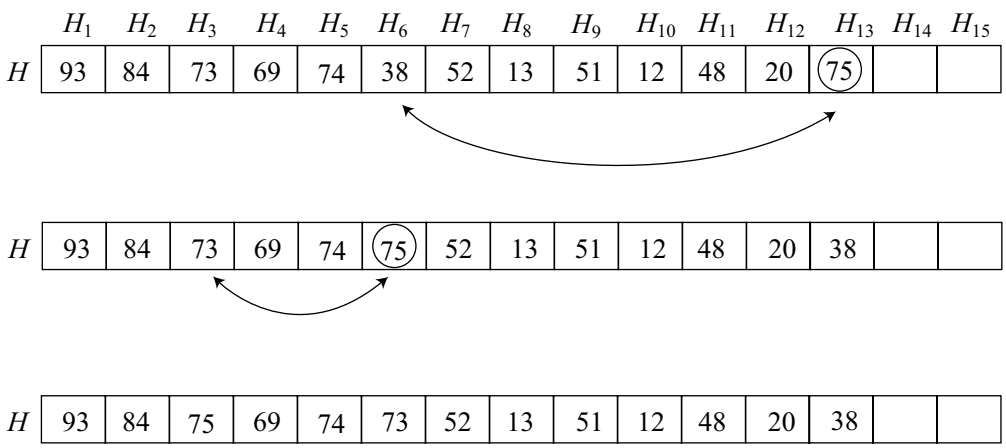
- Εναλλάσσουμε τον κόμβο του εισαχθέντος στοιχείου (‘75’) με το γονέα του (‘38’). Η θέση αυτή δε δημιουργεί σωρό, διότι η συνθήκη του σωρού, ενώ ικανοποιείται από τον κόμβο ‘75’ ($75 > 38$ και $75 > 20$), δεν ικανοποιείται από τον κόμβο ‘73’ ($73 < 75$).
- Εναλλάσσουμε τον κόμβο του εισαχθέντος στοιχείου (‘75’) με το νέο γονέα του (‘73’). Η θέση αυτή δημιουργεί σωρό, διότι η συνθήκη του σωρού ικανοποιείται για τον κόμβο ‘75’ ($75 > 73$ και $75 > 52$), αλλά και για όλους τους υπόλοιπους κόμβους.

Η διαδικασία αυτή είναι μια και μοναδική, δεν υπάρχουν εναλλακτικές. Αν την πραγματοποιήσατε σωστά, συγχαρητήρια! Αν όχι, ξαναμελετήστε το Παράδειγμα 5.2 και τα παραπάνω σχήματα (όπου χρησιμοποιείται μόνο το απαραίτητο τμήμα του δέντρου), τα οποία απεικονίζουν γραφικά την παραπάνω διαδικασία.

Απαντήσεις ασκήσεων αυτοαξιολόγησης



Επίσης, η ίδια διαδικασία παριστάνεται μέσω του πίνακα αναπαράστασης του δέντρου παρακάτω.



Αν κατασκευάσατε σωστά και τα σχήματα και τις αναπαραστάσεις του πίνακα, συγχαρητήρια! Αν όχι, μην τα παρατάτε. Επιστρέψτε στο Παράδειγμα 5.2 για βοήθεια.

6.1

Επειδή θα χρησιμοποιήσουμε μια στοίβα, θα πρέπει σ' αυτήν να απο-

Απαντήσεις ασκήσεων αυτοαξιολόγησης

θηκεύουμε κάθε φορά και τα δύο άκρα του ίδιου υποπίνακα που είναι προς διαχωρισμό, προφανώς σε διαδοχικές θέσεις.

Αν κάνατε αυτή τη σκέψη, τότε ουσιαστικά έχετε προσδιορίσει ποιοτικά τη λύση της άσκησης. Αν όχι, τότε προσοχή μην μπλέξετε σε περιπέτειες! Πάντως αυτή η σκέψη (λύση) είναι ουσιαστικά μονόδρομος. Υπάρχει, βέβαια, και μια άλλη λύση, την οποία όμως είναι δυσκολότερο να σκεφτείτε: Να χρησιμοποιηθεί μια στοίβα, τα στοιχεία της οποίας είναι εγγραφές αποτελούμενες από δυάδες αριθμών. Στη συνέχεια θα ακολουθήσουμε την πρώτη λύση.

ΑΛΓΟΡΙΘΜΟΣ Α6.1: ΓΡΗΓΟΡΗ ΔΙΑΤΑΞΗ (ΠΑΡΑΛΛΑΓΗ)

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A), ο δείκτης αρχής (NL) και ο δείκτης τέλους (NU) του υπό εξέταση (υπο)πίνακα.

Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται αναδιάταξη των στοιχείων του πίνακα.

DIAT-GRHGORH (A, NL, NU)

```
1    T ← 0                                  {Θέση κενών στοιβών}
2    if NL < NU                             {Έλεγχος διαχωρίσιμου πίνακα}
3       then T ← 2                          {Αρχικοποίηση δείκτη στοίβας}
4            S[1] ← NL, S[2] ← NU           {Αρχικοποίηση ορίων πίνακα}
     endif
5    while T ≠ 0                            {Συνθήκη επανάληψης}
6       NL ← S[T], T ← T - 1                {Εξαγωγή κορυφής στοίβας}
7       NU ← S[T], T ← T - 1                {Εξαγωγή κορυφής στοίβας}
8       DIAT-DIAXORISMOS(A, NL, NU)         {Διαχωρισμός πίνακα}
9       if NL < P - 1                       {Έλεγχος διαχωρίσιμου πίνακα}
10         then T ← T + 1, S[T] ← NL        {Εισαγωγή στη στοίβα}
11              T ← T + 1, S[T] ← P - 1     {Εισαγωγή στη στοίβα}
        endif
12      if P + 1 < NU                       {Έλεγχος διαχωρίσιμου πίνακα}
13         then T ← T + 1, S[T] ← P + 1     {Εισαγωγή στη στοίβα}
14              T ← T + 1, S[T] ← NU        {Εισαγωγή στη στοίβα}
        endif
     endwhile
```

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Οι αλλαγές φαίνονται στον Αλγόριθμο Α6.1 στο παραπάνω πλαίσιο. Παρατηρήστε ότι τώρα, μετά από κάθε καταχώριση (εισαγωγή) στη στοίβα ή εξαγωγή (διαγραφή) από τη στοίβα των άκρων ενός υποπίνακα, έχουμε δύο ενημερώσεις του δείκτη κορυφής. Επίσης, η αρχική τιμή που παίρνει ο δείκτης κορυφής τώρα είναι το '2'. Αυτό είναι και το σημείο που πιθανόν να κάνατε κάποιο λάθος. Αν και αυτό το κάνατε σωστά, συγχαρητήρια! Αν όχι, θα το κατανοήσετε με την εφαρμογή του σε ένα παράδειγμα.

Αν εφαρμόσουμε τον αλγόριθμο στο Παράδειγμα 6.2β, η στοίβα θα έχει τα εξής περιεχόμενα κατά την εκτέλεση του αλγορίθμου: S[1, 8], S[ ], S[1, 3, 5, 8], S[1, 3], S[1, 3, 5, 6], S[1, 3], S[1, 3], S[ ], S[ ]. Παρατηρήστε την αντιστοιχία με την περίπτωση των δύο στοιβών.

6.2

Το κλειδί για τη λύση της άσκησης αυτής είναι να εντοπίσουμε σε ποιο σημείο του Αλγόριθμου 6.2 πρέπει να γίνει επέμβαση. Το ζητούμενο εδώ είναι το εξής: Όταν ο τρέχων προς διαχωρισμό υποπίνακας έχει μέγεθος μικρότερο από κάποιο κατώφλι (έστω Μ), τότε δε θα χρησιμοποιούμε τον (μερικό) αλγόριθμο διαχωρισμού, αλλά θα χρησιμοποιούμε τον αλγόριθμο διάταξης επιλογής, ως μερικό αλγόριθμο, για τη διάταξη του πίνακα. Το σημείο του αλγορίθμου γρήγορης διάταξης στο οποίο ορίζεται ο τρέχων προς διαχωρισμό υποπίνακας είναι το βήμα 6, δηλαδή όταν εξάγονται τα άκρα του από τις κορυφές των δύο στοιβών, αμέσως πριν κληθεί ο αλγόριθμος διαχωρισμού (βήμα 8). Σ' αυτό το σημείο πρέπει να γίνουν οι βασικές αλλαγές.

Αν σκεφθήκατε μ' αυτόν ή παραπλήσιο τρόπο και εντοπίσατε το σημείο επέμβασης, συγχαρητήρια! Αυτό είναι το πρώτο, αλλά πολύ σημαντικό βήμα για την απάντηση. Αν δεν το σκεφτήκατε έτσι, τότε ξαναμελετήστε προσεκτικά τον αλγόριθμο 6.2, για να το κατανοήσετε πλήρως.

Τώρα, η αλλαγή που πρέπει να γίνει είναι να εξετάζουμε κάθε φορά, μετά τα βήματα 6 και 7, αν το μέγεθος του υποπίνακα είναι μικρότερο από το κατώφλι Μ και, αν είναι, να καλούμε τον αλγόριθμο διάταξης επιλογής, αλλιώς τον αλγόριθμο διαχωρισμού, όπως και πριν.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

Αυτό ήταν εύκολο να το προσδιορίσουμε μετά από τις παραπάνω σκέψεις, δεδομένου ότι δεν υπάρχει και εναλλακτική λύση. Η υλοποίηση αυτής της σκέψης φαίνεται στο παρακάτω πλαίσιο, όπου παρουσιάζεται αναλυτικά το τμήμα του αλγορίθμου που αφορά τις απαιτούμενες αλλαγές και συνοπτικά τα υπόλοιπα, ως Αλγόριθμος Α6.2 (DIAT-GRHGORH-EPILOGHS). Η τροποποίηση αντικατοπτρίζεται στα βήματα 5-7 του αλγορίθμου αυτού. Τα υπόλοιπα παραμένουν όπως στον αλγόριθμο 6.2.

ΑΛΓΟΡΙΘΜΟΣ Α6.2: ΓΡΗΓΟΡΗ ΔΙΑΤΑΞΗ ΜΕ ΕΠΙΛΟΓΗ

```
DIAT-GRHGORH-EPILOGHS (A, NL, NU)
1    <βήματα 1-4 αλγορίθμου 6.2>
2    while T ≠ 0                          {Συνθήκη επανάληψης}
3      NL ← SL[T], NU ← SU[T]             {Εξαγωγή κορυφών στοιβών}
4      T ← T - 1                          {Ενημέρωση δείκτη κορυφής}
5      if NU - NL < M
6        then DIAT-EPILOGHS (A, NL, NU)   {Κλήση διάταξης επιλογής}
7        else DIAT-DIAXORISMOS(A, NL, NU) {Κλήση διαχωρισμού}
       endif
8      <βήματα 9-14 αλγορίθμου 6.2>
     endwhile
```

Παρατηρήστε ότι η κλήση του αλγορίθμου διάταξης επιλογής γίνεται τώρα με τρεις παραμέτρους εισόδου αντί για δύο, όπως ο Αλγόριθμος 6.1. Ένα λάθος που πιθανόν να έχετε κάνει είναι να καλέσατε τον Αλγόριθμο 6.1 με παραμέτρους A, NU-NL. Δεν είναι σωστό, διότι τώρα ο προς διάταξη (υπο)πίνακας δεν έχει δείκτες στοιχείων από το 1 ως το N, αλλά σε μια υποπεριοχή του [1, N], την [NL, NU], όπου NL > 1 και NU < N. Έτσι, ο Αλγόριθμος 6.1 χρειάζεται μια μικρή αλλαγή για να προσαρμοστεί σ' αυτή την απαίτηση. Αν το σκεφτήκατε και αυτό, πάλι συγχαρητήρια! Αν όχι, τότε πρέπει να είστε προσεκτικοί και να δοκιμάζετε πάντα τον αλγόριθμό σας με ένα μικρό παράδειγμα.

Ο νέος αλγόριθμος διάταξης επιλογής παρουσιάζεται στο παρακάτω πλαίσιο, ως Αλγόριθμος 6.1α

ΑΛΓΟΡΙΘΜΟΣ 6.1α: ΔΙΑΤΑΞΗ ΕΠΙΛΟΓΗΣ (ΠΑΡΑΛΛΑΓΗ)

```
DIAT-EPILOGHS(A, NL, NU)
1    for K ← NL to NU-1                {Έλεγχος τέλους επανάληψης}
2        MINSTOIXEIO (A, K, NU)        {Εύρεση μικρότερου στοιχείου}
3        A[K] ↔ A[X]                   {Εναλλαγή τιμών στοιχείων}
     endfor
```

6.3

Εν γένει, υπάρχουν δύο λύσεις για να διατάξουμε τα στοιχεία κατά φθίνουσα σειρά. Η μια λύση είναι να τροποποιήσουμε την πρώτη φάση της διάταξης σωρού, να κατασκευάσουμε δηλαδή ένα σωρό ελαχίστων αντί για μεγίστων (ανατρέξτε στην Υποενότητα «5.4.1 Ορισμός και Αναπαράσταση» για τη διάκριση αυτή), οπότε, αν αφήσουμε τη δεύτερη φάση όπως έχει, θα έχουμε διάταξη κατά φθίνουσα σειρά, αφού τώρα θα διαγράφονται τα μικρότερα στοιχεία κάθε φορά.

Η δεύτερη λύση είναι να μην τροποποιήσουμε το είδος του σωρού, αλλά τον τρόπο με τον οποίο τοποθετούμε τα στοιχεία που διαγράφονται (ρίζες) στον πίνακα. Να τα τοποθετούμε δηλαδή με τέτοιον τρόπο ώστε να διατάσσονται κατά φθίνουσα σειρά.

Η πρώτη λύση είναι ριζικότερη, αφού απαιτεί αλλαγή της δομής αποθήκευσης, αλλά αποδοτικότερη. Η δεύτερη είναι ηπιότερη, αλλά λιγότερο αποδοτική, όπως θα δούμε στη συνέχεια. Εδώ θα ακολουθήσουμε τη δεύτερη, μιας και αυτή υπονοεί η άσκηση και είναι και δυσκολότερη.

Το κρίσιμο σημείο στη δεύτερη λύση είναι πώς θα τοποθετούμε τα στοιχεία (ρίζες) που διαγράφονται από το σωρό στον πίνακα. Είναι προφανές ότι θα πρέπει να τα τοποθετούμε κατ' αντίστροφη σειρά από ό,τι στον υπάρχοντα αλγόριθμο. Θα πρέπει δηλαδή το κάθε επόμενο στοιχείο να τοποθετείται μετά το προηγούμενο και όχι πριν, όπως στον Αλγόριθμο 6.4. Δηλαδή η κατάσταση μοιάζει με την εισαγωγή στοιχείων σε μια ουρά, μόνο που τώρα πρέπει να μετακινούμε τα προηγούμενα στοιχεία μια θέση αριστερά. Δηλαδή στο Παράδειγμα 6.3, στο δεύτερο βήμα της 2ης φάσης, το '65' δε θα πρέπει να τοποθετη-

Απαντήσεις ασκήσεων αυτοαξιολόγησης

θεί στη θέση A_5, αλλά θα πρέπει να μετακινηθεί πρώτα το στοιχείο '75' από τη θέση A_6 μια θέση αριστερά, δηλαδή στην A_5, και κατόπιν το '65' να εισαχθεί στην κενή θέση A_6. Σημειώστε ότι το στοιχείο που διαγράφεται τοποθετείται πάντα στην τελευταία θέση του πίνακα.

Αν σκεφτήκατε παρόμοια, μπράβο σας! Ήταν μια καλή σκέψη. Αν δεν το σκεφτήκατε, ξαναμελετήστε το μηχανισμό διάταξης κατά αύξουσα σειρά του Αλγόριθμου 6.4.

Η υλοποίηση της σκέψης αυτής στον Αλγόριθμο 6.4 θέλει αρκετή προσοχή, όσον αφορά τις μεταβολές στις τιμές των δεικτών του πίνακα που σχετίζονται με τη μετακίνηση των στοιχείων αριστερά. Μια υλοποίηση παρουσιάζεται στον παρακάτω Αλγόριθμο A6.3 (DIATSOROU-D), που είναι ο τροποποιημένος Αλγόριθμος 6.4.

Αυτό που έγινε είναι ότι προστέθηκε ένα τμήμα αλγορίθμου μεταξύ των βημάτων 4 και 5 του αρχικού αλγορίθμου. Αυτό το τμήμα (βήματα 6-9) ουσιαστικά αποτελείται από μια διάταξη επανάληψης for, που εκτελεί τις απαραίτητες μετακινήσεις πριν από την τοποθέτηση του διαγραφέντος στοιχείου (βήμα 10). Επίσης, προστέθηκε ένα βήμα στην αρχή (βήμα 1). Τέλος, η συνθήκη επανάληψης στο βήμα 3 άλλαξε (βήμα 4, N > 0), διότι τώρα το τελευταίο στοιχείο του σωρού πρέπει να διαγραφεί, αφού δεν είναι στη σωστή θέση, όπως συμβαίνει στην περίπτωση της κατ' αύξουσα σειρά διάταξης.

ΑΛΓΟΡΙΘΜΟΣ Α6.3: ΔΙΑΤΑΞΗ ΣΩΡΟΥ (ΦΘΙΝΟΥΣΑ)

Είσοδος: Ένας μη κενός μονοδιάστατος πίνακας (A) και το πλήθος των στοιχείων του (N).
Έξοδος: Εξωτερικά δεν επιστρέφει τίποτα. Εσωτερικά γίνεται αναδιάταξη των στοιχείων του πίνακα κατά φθίνουσα σειρά.

```
DIAT-SOROU-D(A, N)
1    L ← N
2    for I = 1 to N-1                          {Καθορισμός επαναλήψεων}
3        H-EISAGWGH(A, I, A[I + 1])            {Εισαγωγή στοιχείου στο σωρό}
     endfor
4    while N > 0                               {Συνθήκη επανάληψης}
5        H-DIAGRAFH(A, N)                      {Διαγραφή ρίζας σωρού}
6        M ← L - (N + 1)                       {Ενημέρωση μεταβλητής}
7        for J ← 1 to M                        {Καθορισμός επαναλήψεων}
8            K ← L - (M - J)                   {Ενημέρωση δείκτη}
9            A[K - 1] ← A[K]                   {Μετακίνηση στοιχείου}
         endfor
10       A[N + 1] ← X                          {Τοποθέτηση ρίζας στον πίνακα}
     endwhile
```

Στο βήμα 1 κρατάμε την αρχική τιμή του N, δηλαδή το μέγεθος του αρχικού πίνακα. Στο βήμα 6 καταχωρίζουμε στη βοηθητική μεταβλητή M τον αριθμό των ήδη διαγραφέντων στοιχείων από το σωρό κάθε φορά (θυμηθείτε ότι το N ελαττώνεται κατά ένα μέσα στον αλγόριθμο διαγραφής). Η επαναληπτική διάταξη for μετακινεί τα ήδη διαγραφέντα στοιχεία μια θέση αριστερά (βήμα 9). Η βοηθητική μεταβλητή K αντιπροσωπεύει το δείκτη τού προς μετακίνηση στοιχείου σε κάθε επανάληψη. Τέλος, μετά τις απαραίτητες μετακινήσεις, τοποθετείται το πιο πρόσφατα διαγραφέν στοιχείο στην τελευταία θέση του πίνακα, που έχει εν τω μεταξύ εκκενωθεί.

Αν υλοποιήσατε έτσι ή παρόμοια τον αλγόριθμο, συγχαρητήρια. Αυτό ήταν το πιο δύσκολο βήμα για τη λύση της άσκησης. Αν όχι, βάλτε κάτω χαρτί και μολύβι και εφαρμόστε τον αλγόριθμο στον πίνακα του Παραδείγματος 6.3 για να δείτε τη λειτουργία του και να τον κατανοήσετε. Τα βήματα της εφαρμογής αυτής για τη δεύτερη φάση του

Απαντήσεις ασκήσεων αυτοαξιολόγησης

αλγορίθμου απεικονίζονται στον πίνακα που ακολουθεί. Τα μικρά μη διακεκομμένα βέλη δείχνουν τις μετακινήσεις που γίνονται. Αυτές οι μετακινήσεις σε κάθε βήμα είναι που αυξάνουν την πολυπλοκότητα του αλγορίθμου και τον κάνουν λιγότερο αποδοτικό από τον αρχικό και από αυτόν της πρώτης λύσης.

2η ΦΑΣΗ						
N	A_1	A_2	A_3	A_4	A_5	A_6
6(5)	(75)	50	65	20	30	45
5(4)	(65)	50	45	20	30	**75**
4(3)	(50)	30	45	20	**75**	**65**
3(2)	(45)	30	20	**75**	**65**	**50**
2(1)	(30)	20	**75**	**65**	**50**	**45**
1(0)	(20)	**75**	**65**	**50**	**45**	**30**
	75	**65**	**50**	**45**	**30**	**20**

Απαντήσεις Δραστηριοτήτων

2.1

Ο πίνακας είναι μια ομογενής δομή, δηλαδή όλα τα στοιχεία του ανήκουν στον ίδιο τύπο δεδομένων, σε αντίθεση με την εγγραφή. Επομένως, σ' έναν πίνακα μπορούμε να έχουμε αποθηκευμένα μόνο ομοειδή στοιχεία, π.χ. ακεραίους αριθμούς, χαρακτήρες κλπ.

Ο πίνακας είναι μια δομή τυχαίας προσπέλασης, δηλαδή ο χρόνος προσπέλασης ενός στοιχείου του είναι ανεξάρτητος από τη θέση του στοιχείου στον πίνακα. Για να προσπελάσουμε ένα στοιχείο, δε χρειάζεται να επισκεφθούμε τα προηγούμενά του, όπως συμβαίνει στις δομές σειριακής προσπέλασης. Ο προσδιορισμός της θέσης ενός στοιχείου στη μνήμη γίνεται (εσωτερικά) με βάση τον δείκτη του στοιχείου, δηλαδή τη θέση του στοιχείου στον πίνακα, από τη συνάρτηση απεικόνισης του πίνακα (ΣΑΠ).

Ο πίνακας είναι μια στατική δομή, δηλαδή το μέγεθός του παραμένει αμετάβλητο κατά τη διάρκεια εκτέλεσης ενός προγράμματος, σε αντίθεση με άλλες δομές, τις δυναμικές δομές (όπως, π.χ. οι λίστες), που το μέγεθός τους μεταβάλλεται, δηλαδή προστίθενται νέες θέσεις/στοιχεία. Το μέγεθος ενός πίνακα καθορίζεται με τον ορισμό του.

Ο πίνακας είναι μια γραμμική δομή, δηλαδή οι σχέσεις μεταξύ των στοιχείων του είναι τέτοιες, ώστε απεικονιζόμενες τοποθετούν τα στοιχεία σε μια ευθεία γραμμή (Σχήμα 1.1α στην ενότητα 1.1.4). Το κάθε στοιχείο του πίνακα σχετίζεται με ένα άλλο στοιχείο του με τη σχέση 'γειτονικό', υπάρχει δηλαδή μια σχέση ένα-προς-ένα μεταξύ τους.

3.1

Τα εν λόγω κείμενα αναφέρονται στα πλεονεκτήματα και τα μειονεκτήματα των συνεχόμενων και συνδεδεμένων λιστών. Διακρίνουμε τέσσερις παραμέτρους ως προς τις οποίες γίνεται αυτή η σύγκριση: ορισμός δομής, αποδοτικότητα χώρου, αποδοτικότητα χρόνου και κριτήρια επιλογής.

Ορισμός δομής

- Το μέγιστο μέγεθος μιας συνεχόμενης λίστας πρέπει να καθοριστεί εκ των προτέρων, ενώ σε μια συνδεδεμένη λίστα δεν απαιτείται κάτι τέτοιο. Αυτό δεν είναι πάντα εύκολη υπόθεση.

Απαντήσεις Δραστηριοτήτων

- Το μέγεθος μιας συνεχόμενης λίστας δεν μπορεί να υπερβεί την προκαθορισμένη μέγιστη τιμή. Αυτό δημιουργεί προβλήματα στη λειτουργία του προγράμματος. Τέτοιο θέμα δεν τίθεται σε μια συνδεδεμένη λίστα.

Αποδοτικότητα χώρου

- Σε μια συνδεδεμένη λίστα απαιτείται επί πλέον χώρος μνήμης για την αποθήκευση των δεικτών κάθε κόμβου, ενώ αυτό δεν συμβαίνει σε μια συνεχόμενη λίστα.
- Σε μια συνεχόμενη λίστα, συνήθως δεσμεύεται επιπλέον χώρος στη μνήμη που δεν χρησιμοποιείται τελικά. Αυτό συμβαίνει, διότι η μέγιστη τιμή του μεγέθους της λίστας ορίζεται αρκετά μεγάλη, ώστε να καλύψει όλες τις δυνατές απαιτήσεις, με αποτέλεσμα ένα μεγάλο μέρος να μένει αχρησιμοποίητο. Αυτό δεν συμβαίνει σε μια συνδεμένη λίστα, διότι εκεί η μνήμη δεσμεύεται δυναμικά, όταν απαιτηθεί ένα καινούργιο στοιχείο (dynamic memory allocation).

Αποδοτικότητα χρόνου

- Σε μια συνεχόμενη λίστα η προσπέλαση ενός στοιχείου, δεδομένης της θέσης του, λόγω τυχαίας προσπέλασης, γίνεται γρηγορότερα απ' ότι σε μια απλά συνδεδεμένη λίστα, όπου πρέπει να προσπελάσουμε και όλα τα προηγούμενα στοιχεία. Στην περίπτωση της συνεχόμενης λίστας έχουμε πολυπλοκότητα $O(1)$, ενώ σ' αυτή της συνδεδεμένης $O(n)$, όπου n η θέση του στοιχείου, και στη μέση και στη χειρότερη περίπτωση. Βέβαια, σε μια συνδεδεμένη λίστα, αν μας δίνεται ο αντίστοιχος δείκτης, τότε και εκεί έχουμε πολυπλοκότητα $O(1)$.
- Αντίθετα, η εισαγωγή και η διαγραφή ενός στοιχείου γίνεται γρηγορότερα σε μια συνδεδεμένη λίστα απ' ό,τι σε μια συνεχόμενη, λόγω των απαιτούμενων μετακινήσεων των υπόλοιπων στοιχείων στην τελευταία. Έτσι, ενώ σε μια συνδεδεμένη λίστα, δεδομένου του αντίστοιχου δείκτη, έχουμε πολυπλοκότητα $O(1)$, σε μια συνεχόμενη λίστα, δεδομένης της θέσης, έχουμε πολυπλοκότητα $O(n)$ και στη μέση και στη χειρότερη περίπτωση. Αν, βέβαια, σε μια συνδεδεμένη λίστα δίνεται η θέση του στοιχείου, τότε η πολυπλοκότητα γίνεται και εκεί $O(n)$.

Κριτήρια επιλογής

- Μια συνδεδεμένη λίστα είναι καλύτερη επιλογή, όταν ο αριθμός των στοιχείων που θα δεχθεί έχει μεγάλη διακύμανση ή είναι άγνωστος, αλλιώς μια συνεχόμενη λίστα είναι καλύτερη.
- Μια συνδεδεμένη λίστα απαιτεί λιγότερο χώρο από μια συνεχόμενη όταν σχετικά λίγα στοιχεία υπάρχουν στη λίστα, ενώ το αντίθετο συμβαίνει, όταν ο πίνακας μιας συνεχόμενης λίστας είναι σχεδόν γεμάτος. Πιο συγκεκριμένα για $n > T \cdot S/(D + S)$, όπου n ο αριθμός των χρησιμοποιούμενων στοιχείων, T το μέγιστο μέγεθος της λίστας, S το μήκος του κάθε στοιχείου και D το μήκος κάθε δείκτη, τότε η χρήση συνεχόμενης λίστας είναι πιο συμφέρουσα από πλευράς χώρου.
- Μια συνδεδεμένη λίστα είναι καλύτερη επιλογή, όταν η εφαρμογή απαιτεί πολλές εισαγωγές και διαγραφές στοιχείων, λόγω καλύτερης αποδοτικότητας χρόνου.

Γλωσσάρι όρων

Αλγόριθμος (Algorithm): Μια σαφής περιγραφή του τρόπου λύσης ενός (γενικού) προβλήματος μέσω Η/Υ.

Αλγόριθμος διάταξης (Sorting algorithm): Ένας αλγόριθμος που τακτοποιεί (διατάσσει) τα στοιχεία ενός πίνακα κατά αύξουσα ή φθίνουσα σειρά.

Αναζήτηση (Search): Μια από τις πράξεις σε δομές δεδομένων που αφορά την εύρεση ενός δοθέντος στοιχείου σε μια δομή.

Αναπαράσταση δομής (Data structure representation): Ο τρόπος με τον οποίο αποθηκεύεται μια δομή δεδομένων στη μνήμη του Η/Υ.

Ανώτερη δομή (Higher-level structure): Σύνθετη δομή που χρησιμοποιεί θεμελιώδεις δομές για την υλοποίησή της.

Απλά συνδεμένη λίστα (Singly linked list): Μια συνδεμένη λίστα όπου κάθε κόμβος, εκτός της πληροφορίας (στοιχείο), περιέχει ένα δείκτη (σύνδεσμος) στον επόμενο κόμβο.

Αποδοτικότητα χρόνου (Time efficiency): Η απόδοση ενός αλγορίθμου όσον αφορά το χρόνο εκτέλεσής του.

Αποδοτικότητα χώρου (Space efficiency): Η απόδοση ενός αλγορίθμου όσον αφορά το μέγεθος της μνήμης που απαιτεί κατά την εκτέλεσή του.

Αφηρημένος τύπος δεδομένων (Abstract data type): Το μαθηματικό μοντέλο μιας δομής δεδομένων.

Βαροζυγισμένο δέντρο (Weight balanced tree): Δέντρο στο οποίο οι αριθμοί των κόμβων των υποδέντρων κάθε κόμβου δε διαφέρουν πολύ.

Γενική λίστα (General list): Λίστα χωρίς περιορισμούς στην εισαγωγή και διαγραφή (εξαγωγή) στοιχείων.

Γενικό δέντρο (General tree): Δέντρο χωρίς περιορισμούς στον αριθμό και την οργάνωση των κόμβων του.

Γλώσσα ψευδοκώδικα (Pseudocode): Μια αυστηρά καθορισμένη και ταυτόχρονα σχετικά απλή γλώσσα περιγραφής αλγόριθμων.

Γραμμική αναζήτηση (Linear search): Μέθοδος αναζήτησης στοι-

Γλωσσάρι όρων

χείου σε γραμμική δομή (συνήθως πίνακα), κατά την οποία συγκρίνουμε το δοθέν στοιχείο με κάθε στοιχείο της δομής, ξεκινώντας από την αρχή.

Γραμμική δομή (Linear structure): Δομή δεδομένων της οποίας τα στοιχεία (ή κόμβοι) μπορούν να παρασταθούν ως σημεία μιας ευθείας γραμμής, δηλαδή υπάρχει μια σχέση ένα προς ένα μεταξύ των στοιχείων της.

Γρήγορη διάταξη (Quicksort): Αλγόριθμος διάταξης που χρησιμοποιεί το διαδοχικό διαχωρισμό τής υπό διάταξη γραμμικής δομής (συνήθως πίνακας) σε δύο τμήματα, το ένα δεξιά και το άλλο αριστερά ενός στοιχείου της δομής, που λαμβάνεται ως βάση του διαχωρισμού και ονομάζεται άξονας.

Δείκτης (Index): Αριθμός ή σύμβολο που χρησιμοποιείται για τον προσδιορισμό των στοιχείων ενός πίνακα.

Δείκτης (Pointer): Τύπος δεδομένων με πεδίο τιμών το σύνολο των διευθύνσεων της κεντρικής μνήμης του Η/Υ.

Δέντρο-Σωρός (Heap): Βλέπε "Σωρός".

Διαγραφή (Deletion): Μια από τις πράξεις σε δομές δεδομένων που αφορά την αφαίρεση δοθέντος στοιχείου από μια δομή.

Διαπέραση (Traversal): Μια από τις πράξεις σε δομές δεδομένων που αφορά την επίσκεψη κάθε στοιχείου μιας δομής και εφαρμογή κάποιου είδους επεξεργασίας στο καθένα.

Διάταξη ελέγχου ροής (Flow control structure): Υπολογιστικό σχήμα που επιτρέπει τη διαφοροποίηση από την ακολουθιακή εκτέλεση ενός αλγορίθμου.

Διάταξη if (If structure): Διάταξη ελέγχου ροής που επιτρέπει την επιλογή εκτέλεσης μεταξύ δύο τμημάτων αλγορίθμου με βάση την αλήθεια κάποιας συνθήκης.

Διάταξη επιλογής (Selection sort): Αλγόριθμος διάταξης που πραγματοποιεί n-1 διαδοχικά περάσματα του υπό διάταξη πίνακα (μεγέθους n). Σε κάθε πέρασμα τοποθετείται ένα στοιχείο στη σωστή θέση.

Διάταξη for (For structure): Διάταξη ελέγχου ροής που επιτρέπει την

Γλωσσάρι όρων

επανάληψη εκτέλεσης ενός τμήματος αλγορίθμου ορισμένες φορές με βάση την αλήθεια κάποιας συνθήκης.

Διάταξη κατά γραμμές (Row-major order): Αναπαράσταση ενός δισδιάστατου πίνακα στη μνήμη του Η/Υ με βάση τις γραμμές του.

Διάταξη κατά στήλες (Column-major order): Αναπαράσταση ενός δισδιάστατου πίνακα στη μνήμη του Η/Υ με βάση τις στήλες του.

Διάταξη σωρού (Heapsort): Αλγόριθμος διάταξης που οργανώνει τα στοιχεία ενός πίνακα σε σωρό και τα διατάσσει πραγματοποιώντας διαδοχικές διαγραφές της ρίζας του τρέχοντος σωρού.

Διάταξη while (While structure): Διάταξη ελέγχου ροής που επιτρέπει την επανάληψη εκτέλεσης ενός τμήματος αλγορίθμου ορισμένες φορές με βάση κάποιο μετρητή.

Διπλή ουρά (Double-ended queue ή Deque): Ουρά στην οποία επιτρέπεται εισαγωγή και διαγραφή στοιχείων και από τα δύο άκρα της.

Δομή δεδομένων (Data structure): Μια συλλογή πεπερασμένου πλήθους δεδομένων με καθορισμένη οργάνωση και τρόπους διαχείρισης (ή επεξεργασίας) τους.

Δομή σειριακής προσπέλασης (Sequential access structure): Δομή δεδομένων στην οποία, για να προσπελάσουμε ένα στοιχείο, πρέπει να περάσουμε από όλα τα προηγούμενα. Επομένως, ο χρόνος προσπέλασης ενός στοιχείου σε μια τέτοια δομή εξαρτάται από τη θέση του.

Δομή τυχαίας προσπέλασης (Random access structure): Δομή δεδομένων στην οποία ο χρόνος προσπέλασης ενός στοιχείου είναι ανεξάρτητος από τη θέση του.

Δυαδική αναζήτηση (Binary search): Μέθοδος αναζήτησης στοιχείου σε διατεταγμένο πίνακα, που στηρίζεται στη σύγκριση του δοθέντος στοιχείου με το "μεσαίο" στοιχείο του πίνακα και επιλογή τού ενός εκ των δύο τμημάτων του πίνακα (του αριστερά και δεξιά του μεσαίου στοιχείου) για τη συνέχεια.

Δυαδικό δέντρο αναζήτησης (Binary search tree): Δυαδικό δέντρο με διατεταγμένους κόμβους έτσι ώστε κάθε κόμβος να έχει μεγα-

Γλωσσάρι όρων

λύτερη τιμή από τους κόμβους του αριστερού του υποδέντρου και μικρότερη από αυτούς του δεξιού του υποδέντρου.

Δυναμική δομή (Dynamic structure): Δομή δεδομένων της οποίας το μέγεθος (δηλαδή ο αριθμός των στοιχείων) είναι δυνατόν να μεταβάλλεται κατά την εκτέλεση ενός προγράμματος.

Δισδιάστατος πίνακας (Two-dimensional array): Πίνακας του οποίου κάθε στοιχείο προσδιορίζεται από δύο δείκτες και ο οποίος παριστάνεται γραφικά ως αποτελούμενος από γραμμές και στήλες.

Εγγραφή (Record): Δομή δεδομένων με στοιχεία συνήθως όχι του ίδιου τύπου, που είναι ιεραρχικά δομημένα.

Εισαγωγή (Insertion): Μια από τις πράξεις σε δομές δεδομένων που αφορά την πρόσθεση δοθέντος στοιχείου σε συγκεκριμένη θέση σε μια δομή δεδομένων.

Εναλλαγή (Interchange): Βασική υπολογιστική πράξη που αφορά την αμοιβαία αλλαγή τιμών μεταξύ δύο μεταβλητών.

Ενδοδιατεταγμένη διαπέραση (Inorder traversal): Διαπέραση των κόμβων ενός δυαδικού δέντρου με βάση τη σειρά: αριστερό υποδέντρο – ρίζα – δεξιό υποδέντρο.

Εξωτερική διάταξη (External sorting): Η πράξη της διάταξης στοιχείων αποθηκευμένων (σε αρχείο) στη δευτερεύουσα μνήμη του Η/Υ.

Εσωτερική διάταξη (Internal sorting): Η πράξη της διάταξης στοιχείων αποθηκευμένων (σε πίνακα) στην κύρια μνήμη του Η/Υ.

Ετερογενής δομή (Heterogeneous structure): Δομή δεδομένων της οποίας τα στοιχεία μπορεί να μην είναι του ίδιου τύπου.

Θεμελιώδης δομή (Fundamental structure): Απλή δομή δεδομένων που συνήθως βρίσκεται ενσωματωμένη στις γλώσσες προγραμματισμού.

Ισοζυγισμένο δέντρο (Balanced tree): Δέντρο στο οποίο οι τερματικοί κόμβοι δεν έχουν μεγάλες διαφορές βάθους.

Καταχώριση (Assignment): Βασική υπολογιστική πράξη που αφορά την αποθήκευση τιμής σε μεταβλητή.

Γλωσσάρι όρων

Κάτω τριγωνικός πίνακας (Lower triangular array): Τετραγωνικός πίνακας του οποίου τα στοιχεία που βρίσκονται πάνω από τη διαγώνιο είναι μηδενικά.

Κλειδί (Key): Το πεδίο μιας εγγραφής που προσδιορίζει με μοναδικό τρόπο κάθε στιγμιότυπο της εγγραφής.

Κόμβος (Node): Δομικό στοιχείο των ανώτερων δομών όπου αποθηκεύεται η πληροφορία.

Κυκλική ουρά (Circular queue): Ουρά της οποίας τα δύο άκρα είναι ενωμένα.

Λογική FIFO (First-In First-Out Logic): Λογική κατά την οποία σε μια ουρά η οντότητα που εισέρχεται πρώτη εξυπηρετείται και πρώτη.

Λογική HPIFO (Highest-Priority-In First-Out Logic): Λογική κατά την οποία σε μια ουρά εξυπηρετείται (εξέρχεται) πρώτη η οντότητα (στοιχείο) που έχει τη μεγαλύτερη προτεραιότητα.

Λογική LIFO (Last-In First-Out Logic): Λογική κατά την οποία σε μια ουρά η οντότητα (στοιχείο) που εισέρχεται τελευταία εξυπηρετείται (εξέρχεται) πρώτη.

Μεταβλητή (Variable): Υπολογιστική οντότητα που αντιπροσωπεύει ένα σύνολο δεδομένων που σχετίζονται με μια πραγματική οντότητα, συγκεκριμένη ή αφηρημένη.

Μεταβλητή δείκτη (Pointer variable): Μεταβλητή με πεδίο τιμών τύπου δείκτη, που παίρνει δηλαδή ως τιμές διευθύνσεις της μνήμης του Η/Υ. Ονομάζεται και απλώς "δείκτης".

Μεταβλητή με δείκτη (Indexed variable): Μεταβλητή που δείχνει τη θέση του στοιχείου που αντιπροσωπεύει σ' έναν πίνακα.

Μεταδιατεταγμένη διαπέραση (Postorder traversal): Διαπέραση των κόμβων ενός δυαδικού δέντρου με βάση τη σειρά: αριστερό υποδέντρο – δεξί υποδέντρο – ρίζα.

Μη γραμμική δομή (Non-linear structure): Δομή δεδομένων που δεν είναι γραμμική. Τα στοιχεία της συνδέονται με σχέσεις τύπου ένα προς πολλά ή πολλά προς πολλά.

Μονοδιάστατος πίνακας (One-dimensional array): Πίνακας του

Γλωσσάρι όρων

οποίου κάθε στοιχείο προσδιορίζεται από ένα δείκτη και παριστάνεται γραφικά ως αποτελούμενος από μια γραμμή.

Ομογενής δομή (Homogeneous structure): Δομή δεδομένων της οποίας τα στοιχεία είναι του ίδιου τύπου.

Ουρά (Queue): Ειδική λίστα στην οποία επιτρέπονται εισαγωγές στοιχείων μόνο στο ένα άκρο της και διαγραφές στοιχείων μόνο από το άλλο άκρο της. Ακολουθεί τη λογική FIFO.

Ουρά προτεραιότητας (Priority queue): Ουρά που στην εξαγωγή ενός στοιχείου δεν παίζει ρόλο η σειρά εισαγωγής του, αλλά ο βαθμός προτεραιότητάς του. Ακολουθεί τη λογική HPIFO.

Πάνω τριγωνικός πίνακας (Upper triangular array): Τετραγωνικός πίνακας του οποίου τα στοιχεία που βρίσκονται κάτω από τη διαγώνιο είναι μηδενικά.

Πεδίο (Field): Η περιγραφή ενός στοιχείου μιας εγγραφής.

Πίνακας εγγραφών (Array of records): Πίνακας του οποίου τα στοιχεία είναι εγγραφές.

Πλήρες δέντρο (Complete tree): Δυαδικό δέντρο που σε όλα τα επίπεδά του, πλην ίσως του τελευταίου, έχει το μέγιστο δυνατό αριθμό κόμβων και όλοι οι τερματικοί του κόμβοι βρίσκονται όσο το δυνατόν πιο αριστερά.

Πολυδιάστατος πίνακας (Multi-dimensional array): Πίνακας του οποίου τα στοιχεία προσδιορίζονται από n δείκτες ($n > 1$).

Πολυπλοκότητα (Complexity): Η εκτίμηση της αποδοτικότητας ενός αλγορίθμου.

Πολωνικός συμβολισμός (Polish notation): Τρόπος αναγραφής αριθμητικών εκφράσεων κατά τον οποίο ο αριθμητικός τελεστής (σύμβολο αριθμητικής πράξης) βρίσκεται πριν από τους τελεστέους (αριθμούς, μεταβλητές).

Πράξεις (Operations): Τρόποι διαχείρισης ή επεξεργασίας των τύπων και των δομών δεδομένων.

Προδιατεταγμένη διαπέραση (Preorder traversal): Διαπέραση των κόμβων ενός δυαδικού δέντρου με βάση τη σειρά ρίζα – αριστερό υποδέντρο – δεξιό υποδέντρο.

Γλωσσάρι όρων

Ρίζα (Root): Ο κόμβος ενός δέντρου από τον οποίο ξεκινούν ακμές, αλλά δεν υπάρχουν ακμές που να καταλήγουν σ' αυτόν.

Στατική δομή (Static structure): Δομή δεδομένων της οποίας το οργανωτικό σχήμα και το μέγεθός της (δηλαδή ο αριθμός των στοιχείων) παραμένουν αμετάβλητα κατά την εκτέλεση ενός προγράμματος.

Στοίβα (Stack): Ειδική λίστα στην οποία επιτρέπονται εισαγωγές και διαγραφές (εξαγωγές) στοιχείων μόνο στο ένα άκρο της. Ακολουθεί τη λογική LIFO.

Σύγκριση (Comparison): Βασική υπολογιστική πράξη που αφορά τη σύγκριση δύο υπολογιστικών οντοτήτων.

Συμμετρικός πίνακας (Symmetric array): Τετραγωνικός πίνακας του οποίου τα συμμετρικά ως προς τη διαγώνιο στοιχεία είναι ίσα.

Συνάρτηση απεικόνισης πίνακα (Array mapping function): Συνάρτηση με την οποία εκφράζεται η σχέση της διεύθυνσης της θέσης μνήμης ενός στοιχείου πίνακα και του δείκτη (ή των δεικτών) που προσδιορίζουν το στοιχείο.

Συνάρτηση πολυπλοκότητας (Complexity function): Συνάρτηση με την οποία εκφράζεται η πολυπλοκότητα ενός αλγορίθμου.

Συνδεμένη αναπαράσταση (Linked representation): Τρόπος αναπαράστασης μιας δομής κατά την οποία τα στοιχεία της αποθηκεύονται σε μη διαδοχικές θέσεις μνήμης και η μετάβαση από το ένα στο άλλο γίνεται μέσω συνδέσμων (δεικτών).

Σύνδεσμος (Link): Υπολογιστική οντότητα που "συνδέει" δύο στοιχεία στη συνεχόμενη αναπαράσταση ώστε να είναι δυνατή η μετάβαση από το ένα στο άλλο. Υλοποιείται ως μεταβλητή δείκτη.

Συνεχόμενη αναπαράσταση (Contiguous representation): Τρόπος αναπαράστασης μιας δομής κατά την οποία τα στοιχεία της αποθηκεύονται σε διαδοχικές θέσεις μνήμης και η μετάβαση από το ένα στο άλλο γίνεται με τη μετάβαση στην επόμενη θέση μνήμης. Πρότυπο της αναπαράστασης αυτής είναι ο πίνακας.

Σύνολο δεικτών (Index set): Σύνολο που περιέχει τους δείκτες των στοιχείων ενός πίνακα.

Γλωσσάρι όρων

Τετραγωνικός πίνακας (Rectangular array): Δισδιάστατος πίνακας του οποίου οι δύο διαστάσεις του έχουν το ίδιο μέγεθος.

Τριδιαγώνιος πίνακας (Tridiagonal array): Τετραγωνικός πίνακας του οποίου όλα τα στοιχεία είναι μηδενικά, πλην αυτών της (κύριας) διαγωνίου, της υπερδιαγωνίου (δηλαδή αυτής που βρίσκεται από πάνω της) και της υποδιαγωνίου (δηλαδή αυτής που βρίσκεται από κάτω της).

Τύπος δεδομένων (Data type): Τρόπος παράστασης και χρήσης μιας ομάδας δεδομένων.

Υποδέντρο (Subtree): Δέντρο που έχει ως ρίζα κάποιον κόμβο του δέντρου στο οποίο ανήκει.

Υψοζυγισμένο δέντρο (Height balanced tree): Δέντρο στο οποίο τα ύψη των υποδέντρων κάθε κόμβου δε διαφέρουν πολύ.

Βιβλιογραφία

ΕΛΛΗΝΟΓΛΩΣΣΗ

Κοίλιας Χ., *Δομές Δεδομένων και Οργανώσεις Αρχείων*, Αθήνα, εκδόσεις Νέων Τεχνολογιών, 1993.

Λουκάκης Μ., *Δομές Δεδομένων και Αλγόριθμοι*, τόμος Α΄, Θεσσαλονίκη, εκδόσεις Ιθάκη, 1987.

Μανωλόπουλος Ι., *Δομές Δεδομένων*, τόμ. Α΄, Θεσσαλονίκη, ART of TEXT, 2η έκδοση, 1992.

Τσακαλίδης Α. Κ., *Δομές Δεδομένων*, Πανεπιστημιακές Παραδόσεις, Πανεπιστήμιο Πατρών, 1994.

Wirth N., *Αλγόριθμοι & Δομές Δεδομένων*, Αθήνα, εκδόσεις Κλειδάριθμος, 1990.

ΞΕΝΟΓΛΩΣΣΗ

Adamson I. A., *Data Structures and Algorithms: A First Course*, Springer-Verlag, London, 1996.

Cormen T. H., Leiserson C. E., Rivest R. L., *Introduction to Algorithms*, McGraw-Hill, New York, NY, 1996.

Horowitz E., Sahni S., *Fundamentals of Data Structures in Pascal*, 2nd Edition, Computer Science Press, Rockville, ML, 1987.

Knuth D. E., *The Art of Computer Programming*, *Vol. 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1973.

Knuth D. E., *The Art of Computer Programming*, *Vol.3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

Kruse R. L., *Data Structures and Program Design*, 2nd Edition, Prentice Hall, Englewood Cliffs, NJ, 1987.

Lipschutz S., *Schaum's Outline of Theory and Problems of Data Structures*, Schaum's Outline Series in Computer Science, McGraw-Hill, New York, NY, 1986.

Main M., **Savitch W.**, *Data Structures and Other Objects*, Turbo Pascal Edition, Benjamin/Cummings, Redwood City, CA, 1995.

Mehlhorn K., *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, Berlin, Germany, 1984.

Shaffer C. A., *A Practical Introduction to Data Structures and Algorithm Analysis*, Prentice Hall, Upper Saddle River, NJ, 1997.

Smith H. F., *Data Structures, Form and Function*, Harcourt Brace Jovanovich, Orlando, FL, 1987.

Standish T. A., *Data Structures, Algorithm and Software Principles*, Addison-Wesley, Reading, MA, 1994.

Stubbs D. F., Webre N. W., *Data Structures with Abstract Data Types and Pascal*, 2nd Edition, Brooks/Cole, Pacific Grove, CA, 1989.

Tenenbaum A. M. and Augenstein M. J., *Data Structures Using Pascal*, 2nd Edition, Prentice Hall, Englewood Cliffs, NJ, 1986.

Weiss M. A., *Data Structures and Algorithm Analysis*, 2nd Edition, Benjamin/Cummings, Redwood City, CA, 1995.