12 Αφηρημένοι τύποι δεδομένων

Εισαγωγή στην Επιστήμη των Υπολογιστών © Εκδόσεις Κλειδάριθμος

12.1

Στόχοι

Μετά την ολοκλήρωση αυτού του κεφαλαίου, ο σπουδαστής θα είναι σε θέση:

- Να κατανοεί την έννοια του αφηρημένου τύπου δεδομένων.
- Να ορίζει τις στοίβες, τις βασικές λειτουργίες που εκτελούνται σε αυτές, τις εφαρμογές τους, και τον τρόπο υλοποίησής τους.
- Να ορίζει τις ουρές, τις βασικές λειτουργίες που εκτελούνται σε αυτές, τις εφαρμογές τους, και τον τρόπο υλοποίησής τους.
- Να ορίζει τις γενικές γραμμικές λίστες, τις βασικές λειτουργίες που εκτελούνται σε αυτές, τις εφαρμογές τους, και τον τρόπο υλοποίησής τους.
- Να ορίζει τα γενικά δέντρα και τις εφαρμογές τους.
- Να ορίζει τα δυαδικά δέντρα — ένας ειδικός τύπος δέντρων — και τις εφαρμογές τους.
- Να ορίζει τα δυαδικά δέντρα αναζήτησης (BST) και τις εφαρμογές τους.
- Να ορίζει τους γράφους και τις εφαρμογές τους.

12.2

12-1 ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Η επίλυση προβλημάτων με έναν υπολογιστή σημαίνει επεξεργασία δεδομένων. Για να επεξεργαστούμε δεδομένα, πρέπει να ορίσουμε τον τύπο τους και τη λειτουργία που θα εκτελεστεί σε αυτά. Ο ορισμός του τύπου των δεδομένων και της λειτουργίας που θα εφαρμοστεί σε αυτά είναι ένα βασικό στοιχείο της έννοιας του **αφηρημένου τύπου δεδομένων**, δηλαδή της απόκρυψης του τρόπου εκτέλεσης της λειτουργίας στα δεδομένα. Με άλλα λόγια, ο χρήστης ενός αφηρημένου τύπου δεδομένων πρέπει μόνο να γνωρίζει ποιες λειτουργίες είναι διαθέσιμες για τον συγκεκριμένο τύπο δεδομένων, χωρίς να χρειάζεται να γνωρίζει πώς εφαρμόζονται.

12.3

Απλοί αφηρημένοι τύποι δεδομένων

Μερικοί απλοί αφηρημένοι τύποι δεδομένων ορίζονται ήδη σε πολλές γλώσσες προγραμματισμού ως αναπόσπαστο μέρος τους. Για παράδειγμα, στη γλώσσα C ορίζεται ένα απλός αφηρημένος τύπος δεδομένων που ονομάζεται ακέραιος (integer). Πρόκειται για έναν ακέραιο τύπο δεδομένων με προκαθορισμένα διαστήματα τιμών. Στη C ορίζονται επίσης πολλές λειτουργίες που μπορούν να εφαρμοστούν σε αυτόν τον τύπο δεδομένων (οι πράξεις της πρόσθεσης, της αφαίρεσης, του πολλαπλασιασμού, της διαίρεσης, και τα λοιπά). Σε αυτή τη γλώσσα προγραμματισμού ορίζονται ρητά αυτές οι πράξεις σε ακεραίους, καθώς και το αποτέλεσμα που παράγουν. Ένας προγραμματιστής που γράφει ένα πρόγραμμα σε C για την πρόσθεση δύο ακεραίων πρέπει να γνωρίζει τον ακέραιο αφηρημένο τύπο δεδομένων και τις λειτουργίες που μπορούν να εφαρμοστούν σε αυτόν.

12.4

Σύνθετοι αφηρημένοι τύποι δεδομένων

Παρόλο που αρκετοί απλοί αφηρημένοι τύποι δεδομένων, όπως ακέραιοι, πραγματικοί αριθμοί, χαρακτήρες, δείκτες, και τα λοιπά, έχουν υλοποιηθεί και είναι διαθέσιμοι για χρήση στις περισσότερες γλώσσες, δεν συμβαίνει το ίδιο για πολλούς χρήσιμους σύνθετους αφηρημένους τύπους δεδομένων. Όπως θα δούμε σε αυτό το κεφάλαιο, χρειαζόμαστε αφηρημένους τύπους δεδομένων λίστας, στοίβας, ουράς, και ούτω καθεξής. Για να είναι αποδοτικοί αυτοί οι αφηρημένοι τύποι δεδομένων, πρέπει να δημιουργούνται και να αποθηκεύονται σε μια βιβλιοθήκη ώστε να μπορούν να χρησιμοποιούνται από τον υπολογιστή.

Η έννοια της αφαίρεσης είναι η εξής:
1. Είναι γνωστό το τι μπορεί να κάνει ένας τύπος δεδομένων
2. Το πώς το κάνει είναι κρυμμένο

12.5

Ορισμός

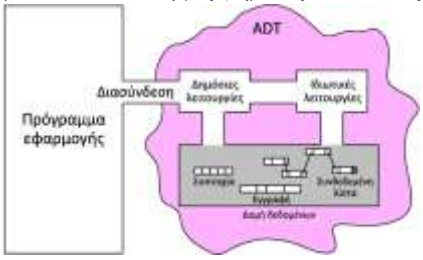
Ας προχωρήσουμε τώρα στον ορισμό των αφηρημένων τύπων δεδομένων. Ένας αφηρημένος τύπος δεδομένων είναι ένας τύπος δεδομένων ο οποίος συνοδεύεται από τις λειτουργίες που έχουν νόημα για τον συγκεκριμένο τύπο. Τα δεδομένα και οι λειτουργίες τους "ενθυλακώνονται" (encapsulate) και κρύβονται από τον χρήστη.

Αφηρημένος τύπος δεδομένων:
1. Ορισμός δεδομένων.
2. Ορισμός λειτουργιών.
3. Ενθυλάκωση δεδομένων και λειτουργιών.

12.6

Μοντέλο αφηρημένου τύπου δεδομένων

Το μοντέλο του αφηρημένου τύπου δεδομένων φαίνεται στην Εικόνα 12.1. Μέσα στην περιοχή του αφηρημένου τύπου δεδομένων υπάρχουν δύο διαφορετικά τμήματα του μοντέλου: η δομή δεδομένων και οι λειτουργίες (δημόσιες και ιδιωτικές).



Εικόνα 12.1 **Το μοντέλο ενός αφηρημένου τύπου δεδομένων**

12.7

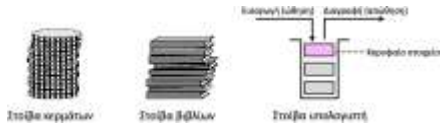
Υλοποίηση

Οι γλώσσες προγραμματισμού δεν παρέχουν πακέτα αφηρημένων τύπων δεδομένων. Για να δημιουργηθεί ένας αφηρημένος τύπος δεδομένων πρέπει πρώτα να υλοποιηθεί και να τοποθετηθεί σε μια βιβλιοθήκη. Ο βασικός σκοπός αυτού του κεφαλαίου είναι η παρουσίαση μερικών συνηθισμένων οριζόμενων από τον χρήστη αφηρημένων τύπων δεδομένων και των εφαρμογών τους. Παρέχουμε ωστόσο και μια σύντομη περιγραφή για την υλοποίηση κάθε αφηρημένου τύπου δεδομένων για όσους θέλουν να ασχοληθούν περισσότερο με το θέμα. Τον ψευδοκώδικα των αλγορίθμων για τις υλοποιήσεις τον αφήνουμε ως ενδιαφέρουσες ασκήσεις.

12.8

12-2 ΣΤΟΙΒΕΣ

Η στοίβα είναι μια περιορισμένη γραμμική λίστα στην οποία όλες οι προσθήκες και οι διαγραφές γίνονται στο ένα άκρο, την κορυφή. Αν μια σειρά στοιχείων δεδομένων εισαχθεί σε μια στοίβα και μετά αφαιρεθεί από αυτή, η σειρά των στοιχείων θα αντιστραφεί. Λόγω αυτού του χαρακτηριστικού αντιστροφής οι στοίβες είναι γνωστές ως δομές δεδομένων LIFO (last in, first out, δηλαδή "αυτό που μπαίνει τελευταίο βγαίνει πρώτο").



Εικόνα 12.2 Τρεις αναπαραστάσεις στοίβας

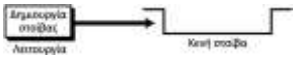
Λειτουργίες σε στοίβες

Υπάρχουν τέσσερις βασικές λειτουργίες, *δημιουργίας στοίβας* (stack), *ώθησης* (push), *απώθησης* (pop), και *ελέγχου για κενή στοίβα* (empty), τις οποίες ορίζουμε σε αυτό το κεφάλαιο.

Η λειτουργία *δημιουργίας στοίβας*

Η λειτουργία *δημιουργίας στοίβας* (stack) δημιουργεί μια κενή στοίβα. Η μορφή είναι η εξής.

stack (όνομαΣτοίβας)

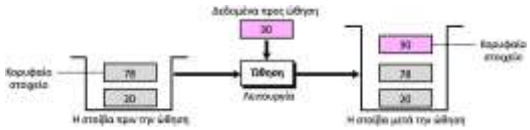


Εικόνα 12.3 Λειτουργία δημιουργίας στοίβας

Η λειτουργία *ώθησης*

Η λειτουργία *ώθησης* (push) εισαγάγει ένα στοιχείο στην κορυφή της στοίβας. Η μορφή είναι η εξής.

push (όνομαΣτοίβας, στοιχείοΔεδομένων)

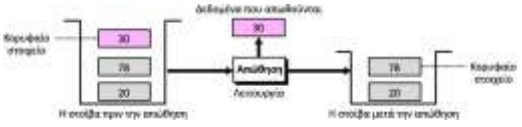


Εικόνα 12.4 Η λειτουργία ώθησης

Η λειτουργία *απώθησης*

Η λειτουργία *απώθησης* (pop) διαγράφει ένα στοιχείο από την κορυφή της στοίβας. Η μορφή είναι η εξής.

pop (όνομαΣτοίβας, στοιχείοΔεδομένων)



Εικόνα 12.5 Η λειτουργία απώθησης

Η λειτουργία *ελέγχου για κενή λίστα*

Η λειτουργία *ελέγχου για κενή στοίβα* ελέγχει την κατάσταση της στοίβας. Η μορφή είναι η εξής.

empty (όνομαΣτοίβας)

Η λειτουργία αυτή επιστρέφει την τιμή true αν η στοίβα είναι κενή ή την τιμή false αν η στοίβα δεν είναι κενή.

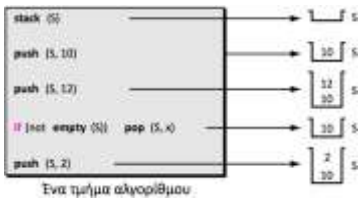
Αφηρημένοι τύποι δεδομένων στοίβας

Μια στοίβα ορίζεται ως αφηρημένος τύπος δεδομένων ως εξής:

Αφηρημένος τύπος δεδομένων στοίβας	
Ορισμός	Μια λίστα στοιχείων δεδομένων που μπορούν να προσπελαστούν μόνο από το ένα άκρο, το οποίο ονομάζεται *κορυφή* της στοίβας.
Λειτουργίες	**stack:** Δημιουργεί μια κενή στοίβα.
	push: Προσθέτει ένα στοιχείο στην κορυφή.
	pop: Διαγράφει το στοιχείο στην κορυφή.
	empty: Ελέγχει την κατάσταση της στοίβας.

Παράδειγμα 12.1

Στην Εικόνα 12.6 παρουσιάζεται ένα τμήμα του αλγορίθμου με το οποίο εφαρμόζονται οι λειτουργίες που ορίζονται παραπάνω σε μια στοίβα *S*.



Εικόνα 12.6 **Παράδειγμα 12.1**

Εφαρμογές στοίβας

Οι εφαρμογές στοίβας μπορούν να ταξινομηθούν σε τέσσερις μεγάλες κατηγορίες: ***αντιστροφή δεδομένων***, ***αντιστοίχιση δεδομένων***, ***αναβολή χρήσης δεδομένων***, και ***οπισθοδρόμηση***. Στις ενότητες που ακολουθούν περιγράφονται οι δύο πρώτες.

Αντιστροφή στοιχείων δεδομένων

Κατά την αντιστροφή δεδομένων ένα καθορισμένο σύνολο στοιχείων αναδιατάσσεται έτσι ώστε το πρώτο και το τελευταίο στοιχείο να αλλάξουν θέσεις μεταξύ τους, ενώ και όλα τα ενδιάμεσα στοιχεία μεταξύ του πρώτου και του τελευταίου να αντιμετατεθούν με αντίστοιχο τρόπο. Για παράδειγμα, η λίστα (2, 4, 7, 1, 6, 8) γίνεται (8, 6, 1, 7, 4, 2).

Παράδειγμα 12.2

Στο Κεφάλαιο 2 (Εικόνα 2.6 στη σελίδα 57) δείξαμε ένα απλό διάγραμμα UML για τη μετατροπή ενός ακεραίου από το δεκαδικό σε οποιοδήποτε άλλο σύστημα. Παρόλο που ο αλγόριθμος είναι πολύ απλός, αν τυπώσουμε τα ψηφία τού μετατρεπόμενου ακεραίου κατά τη δημιουργία τους, αυτά θα εμφανιστούν σε αντίστροφη σειρά. Η εντολή εκτύπωσης σε οποιαδήποτε γλώσσα προγραμματισμού τυπώνει χαρακτήρες από αριστερά προς τα δεξιά, όμως ο αλγόριθμος δημιουργεί τα ψηφία από τα δεξιά προς τα αριστερά. Για να λύσετε το πρόβλημα μπορείτε να χρησιμοποιήσετε το χαρακτηριστικό της αντιστροφής στοίβας (δομή LIFO).

Ο Αλγόριθμος 12.1 δείχνει τον ψευδοκώδικα για τη μετατροπή ενός δεκαδικού ακεραίου σε δυαδικό και την εκτύπωση του αποτελέσματος. Καταρχήν δημιουργούμε μια κενή στοίβα. Έπειτα χρησιμοποιούμε έναν βρόχο while για να δημιουργήσουμε τα bit, τα οποία όμως δεν τυπώνουμε αλλά ωθούμε (push) στη στοίβα. Όταν δημιουργηθούν όλα τα bit τερματίζουμε τον βρόχο. Τώρα χρησιμοποιούμε έναν άλλο βρόχο για να απωθήσουμε (pop) τα bit από τη στοίβα και να τα τυπώσουμε. Σημειώστε ότι τα bit τυπώνονται σε αντίστροφη σειρά από αυτήν με την οποία δημιουργήθηκαν.

Παράδειγμα 12.2 (Συνέχεια)

Αλγόριθμος 12.1 Παράδειγμα 12.2

Αλγόριθμος: **DecimalToBinary** (αριθμός)

Σκοπός: Εκτύπωση του δυαδικού ισοδύναμου ενός δεδομένου ακεραίου (απόλυτη τιμή)

Προ-συνθήκη: Δίνεται ο ακέραιος που θα μετατραπεί (αριθμός)

Μετα-συνθήκη: Τυπώνεται ο δυαδικός ακέραιος

Επιστρέφεται: Τίποτα

```
{
        stack (S)
        while (number ≠ 0)
```


Παράδειγμα 12.2 (Συνέχεια)

Αλγόριθμος 12.1 *(συνέχεια)*

```
        {
                remainder ← number  mod 2
                push (S, remainder)
                number ← number / 2
        }
        while (not empty (S))



        {
                pop (S, x)
                print (x)
        }
        return
}
```


Αντιστοίχιση στοιχείων

Συχνά χρειάζεται να συνδυάζουμε (αντιστοιχίζουμε) μερικούς χαρακτήρες σε μια παράσταση. Για παράδειγμα, κατά τη δημιουργία μιας μαθηματικής παράστασης σε μια γλώσσα προγραμματισμού, πρέπει να χρησιμοποιούμε παρενθέσεις για να αλλάζουμε την προτεραιότητα των τελεστών. Οι επόμενες δύο παραστάσεις αποτιμώνται διαφορετικά επειδή η δεύτερη παράσταση περιέχει παρενθέσεις:

$3 \times 6 + 2 = 20$ $3 \times (6 + 2) = 24$

Όταν γράφουμε παραστάσεις με πολλές παρενθέσεις, είναι πολύ πιθανό να ξεχάσουμε κάποια παρένθεση. Ένα από τα καθήκοντα του μεταγλωττιστή είναι να πραγματοποιεί αυτόν τον έλεγχο για λογαριασμό μας. Έτσι, ο μεταγλωττιστής χρησιμοποιεί μια στοίβα για να διασφαλίζει ότι όλες οι παρενθέσεις ανοίγματος αντιστοιχίζονται με παρενθέσεις κλεισίματος.

Παράδειγμα 12.3

Ο Αλγόριθμος 12.2 δείχνει πώς μπορούμε να ελέγξουμε αν όλες οι παρενθέσεις ανοίγματος έχουν αντίστοιχες παρενθέσεις κλεισίματος.

Αλγόριθμος 12.2 Παράδειγμα 12.3

```
Αλγόριθμος: CheckingParentheses (παράσταση)
Σκοπός: Έλεγχος αντιστοίχισης παρενθέσεων σε μια παράσταση
Προ-συνθήκη: Δίνεται η παράσταση προς έλεγχο
Μετα-συνθήκη: Τυπώνονται μηνύματα σφαλμάτων αν δεν βρεθούν αντιστοιχίσεις παρενθέσεων
Επιστρέφεται: Τίποτα
{
    stack (S)
    while (όσο υπάρχουν χαρακτήρες στην παράσταση)
    {
        Char ← next character
        if (Char = '(')      push (S, Char)
        else
        {
            if (Char = ')')
```


Παράδειγμα 12.3 (Συνέχεια)

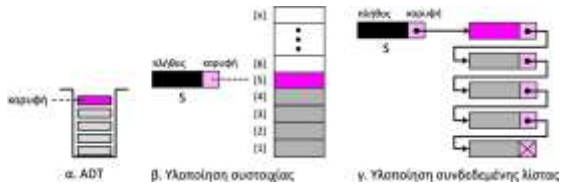
Αλγόριθμος 12.2 *(συνέχεια)*

```
            {
                if (empty (S))    print (μη αντιστοιχισμένη παρένθεση ανοίγματος)
                else              pop (S, x)
            }
        }
    }
    if (not empty (S))            print (παρένθεση κλεισίματος χωρίς αντιστοίχιση)
    return
}
```


Υλοποίηση στοίβας

Στο επίπεδο του αφηρημένου τύπου δεδομένων χρησιμοποιείται η στοίβα και οι τέσσερις λειτουργίες της, ενώ στο επίπεδο υλοποίησης πρέπει να επιλεγεί μια δομή δεδομένων και να υλοποιηθεί. Ο αφηρημένος τύπος δεδομένων στοίβας μπορεί να υλοποιηθεί είτε ως συστοιχία είτε ως συνδεδεμένη λίστα. Στην Εικόνα 12.7 παρουσιάζεται ένα παράδειγμα αφηρημένου τύπου δεδομένων στοίβας με πέντε στοιχεία. Στην εικόνα φαίνεται επίσης ο τρόπος υλοποίησης της στοίβας.

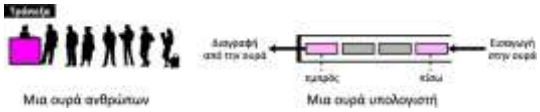
Στη δική μας υλοποίηση συστοιχίας έχουμε μια εγγραφή με δύο πεδία. Το πρώτο πεδίο μπορεί να χρησιμοποιηθεί για την αποθήκευση πληροφοριών που αφορούν τη στοίβα. Παρόμοια είναι και η υλοποίηση της συνδεδεμένης λίστας: υπάρχει ένας επιπλέον κόμβος που έχει το όνομα της στοίβας. Ο κόμβος αυτός διαθέτει δύο πεδία, έναν μετρητή και έναν δείκτη που δείχνει στο στοιχείο στην κορυφή.



Εικόνα 12.7 Υλοποίηση στοίβας

12-3 ΟΥΡΕΣ

Η **ουρά** (queue) είναι μια γραμμική λίστα στην οποία τα δεδομένα εισάγονται μόνο από το ***πίσω άκρο*** (το τέλος) και διαγράφονται μόνο από το ***εμπρός άκρο*** (την αρχή). Αυτοί οι περιορισμοί εξασφαλίζουν ότι η επεξεργασία των δεδομένων σε μια ουρά γίνεται με τη σειρά που αυτά λαμβάνονται. Με άλλα λόγια, η ουρά είναι μια δομή **FIFO** (first in, first out).



Εικόνα 12.8 Δύο αναπαραστάσεις ουρών

Λειτουργίες σε ουρές

Αν και μπορούν να οριστούν διάφορες λειτουργίες για μια ουρά, οι βασικές είναι οι εξής τέσσερις: η δημιουργία ουράς, η εισαγωγή σε ουρά, η εξαγωγή από ουρά, και ο έλεγχος για κενή ουρά.

Η λειτουργία *δημιουργίας ουράς*

Η λειτουργία δημιουργίας ουράς (queue) δημιουργεί μια κενή ουρά. Η μορφή είναι η εξής.

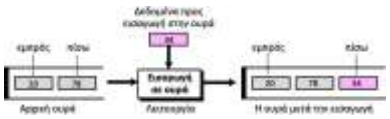
queue (όνομαΟυράς)

Εικόνα 12.9 Η λειτουργία δημιουργίας ουράς

Η λειτουργία *εισαγωγής σε ουρά*

Η λειτουργία εισαγωγής σε ουρά (enqueue) εισάγει ένα στοιχείο στο τέλος της ουράς. Η μορφή είναι η εξής.

enqueue (όνομαΟυράς, στοιχείοΔεδομένων)

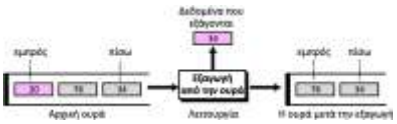


Εικόνα 12.10 Η λειτουργία εισαγωγής σε ουρά

Η λειτουργία *εξαγωγής από ουρά*

Η λειτουργία εξαγωγής από ουρά (dequeue) διαγράφει το στοιχείο στην αρχή της ουράς. Η μορφή είναι η εξής.

dequeue (όνομαΟυράς, στοιχείοΔεδομένων)



Εικόνα 12.11 Η λειτουργία εξαγωγής από ουρά

Η λειτουργία *ελέγχου για κενή ουρά*

Η λειτουργία ελέγχου για κενή ουρά ελέγχει την κατάσταση της ουράς. Η μορφή είναι η εξής.

empty (όνομαΟυράς)

Η λειτουργία αυτή επιστρέφει την τιμή true αν η ουρά είναι κενή ή την τιμή false αν η ουρά δεν είναι κενή.

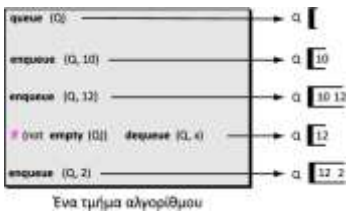
Αφηρημένοι τύποι δεδομένων ουράς

Μια ουρά ορίζεται ως αφηρημένος τύπος δεδομένων ως εξής:

Αφηρημένος τύπος δεδομένων ουράς	
Ορισμός	Μια λίστα στοιχείων δεδομένων στην οποία τα στοιχεία διαγράφονται από την αρχή της ουράς και εισάγονται από το τέλος της.
Λειτουργίες	**queue:** Δημιουργεί μια κενή ουρά. **enqueue:** Προσθέτει ένα στοιχείο στο τέλος. **dequeue:** Διαγράφει ένα στοιχείο από την αρχή. **empty:** Ελέγχει την κατάσταση της ουράς.

Παράδειγμα 12.4

Στην Εικόνα 12.12 παρουσιάζεται ένα τμήμα του αλγορίθμου με το οποίο εφαρμόζονται οι λειτουργίες που ορίζονται παραπάνω σε μια ουρά Q.



Ένα τμήμα αλγορίθμου

Εικόνα 12.12 **Παράδειγμα 12.4**

Εφαρμογές ουρών

Η ουρά αποτελεί μία από τις πιο συνηθισμένες δομές επεξεργασίας δεδομένων. Τη συναντάμε σχεδόν σε κάθε λειτουργικό σύστημα και δίκτυο αλλά και σε αμέτρητους άλλους τομείς. Για παράδειγμα, οι ουρές χρησιμοποιούνται σε online εμπορικές εφαρμογές όπως η επεξεργασία αιτήσεων πελατών, εργασιών, και παραγγελιών. Σε ένα υπολογιστικό σύστημα οι ουρές είναι απαραίτητες για την επεξεργασία εργασιών καθώς και για υπηρεσίες συστήματος όπως η παροχέτευση εκτυπώσεων (print spooling).

Παράδειγμα 12.5

Οι ουρές μπορούν να χρησιμοποιηθούν για την οργάνωση βάσεων δεδομένων σύμφωνα με ορισμένα χαρακτηριστικά των δεδομένων. Ας υποθέσουμε, για παράδειγμα, ότι έχουμε μια λίστα με ταξινομημένα δεδομένα που είναι αποθηκευμένα στον υπολογιστή και ανήκουν σε δύο κατηγορίες: μικρότερα από 1000 και μεγαλύτερα από 1000. Για να διαχωρίσουμε τις κατηγορίες διατηρώντας παράλληλα τη σειρά των δεδομένων στην κατηγορία όπου ανήκουν μπορούμε να χρησιμοποιήσουμε δύο ουρές. Ο ψευδοκώδικας για αυτήν τη λειτουργία παρουσιάζεται στον Αλγόριθμο 12.3.

Παράδειγμα 12.5 (Συνέχεια)

Αλγόριθμος 12.3 Παράδειγμα 12.5

```
Αλγόριθμος: Categorizer (list)
Σκοπός: Διαχωρισμός των δεδομένων σε δύο κατηγορίες και δημιουργία δύο ξεχωριστών λιστών.
Προ-συνθήκη: Δίνεται η αρχική λίστα
Μετα-συνθήκη: Τυπώνονται οι δύο λίστες
Επιστρέφεται: Τίποτα
{
    queue (Q1)
    queue (Q2)
    while (όσο υπάρχουν δεδομένα στη λίστα)
    {
        if (data < 1000)          enqueue (Q1, data)
        if (data ≥ 1000)          enqueue (Q2, data)
    }
```


Παράδειγμα 12.5 (Συνέχεια)

Αλγόριθμος 12.3 *(συνέχεια)*

```
    while (not empty(Q1))
    {
        dequeue (Q1, x)
        print (x)
    }
    while (not empty(Q2))
    {
        dequeue (Q2, x)
        print (x)
    }
    return
}
```

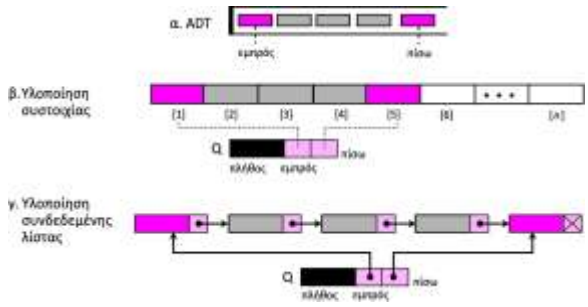

Παράδειγμα 12.6

Μια άλλη συνηθισμένη εφαρμογή των ουρών είναι για τη δημιουργία και τη ρύθμιση της ισορροπίας ανάμεσα σε μια οντότητα που παράγει γρήγορα δεδομένα και σε μια άλλη που τα αξιοποιεί. Για παράδειγμα, υποθέστε ότι σε έναν εκτυπωτή είναι συνδεδεμένη μια ΚΜΕ. Η ταχύτητα του εκτυπωτή δεν μπορεί να συγκριθεί με την ταχύτητα της ΚΜΕ. Αν η ΚΜΕ πρέπει να περιμένει τον εκτυπωτή για να τυπώνει τα δεδομένα που του στέλνει, τότε θα παραμένει αδρανής για μεγάλα χρονικά διαστήματα. Η λύση σε αυτό το πρόβλημα είναι η ουρά. Η ΚΜΕ δημιουργεί όσα τμήματα δεδομένων μπορεί να φιλοξενήσει η ουρά και τα στέλνει σε αυτήν. Έτσι η ΚΜΕ θα είναι διαθέσιμη για την εκτέλεση άλλων εργασιών. Τα τμήματα δεδομένων εξάγονται από την ουρά με αργούς ρυθμούς και τυπώνονται από τον εκτυπωτή. Η ουρά που χρησιμοποιείται για τον σκοπό αυτό συνήθως αναφέρεται ως ουρά παροχέτευσης (spool queue).

Υλοποίηση ουράς

Στο επίπεδο του αφηρημένου τύπου δεδομένων χρησιμοποιείται η ουρά και οι τέσσερις λειτουργίες της, ενώ στο επίπεδο υλοποίησης πρέπει να επιλεγεί μια δομή δεδομένων και να υλοποιηθεί. Ο αφηρημένος τύπος δεδομένων ουράς μπορεί να υλοποιηθεί είτε ως συστοιχία είτε ως συνδεδεμένη λίστα. Στην Εικόνα 12.13 της σελίδας 479 παρουσιάζεται ένα παράδειγμα αφηρημένου τύπου δεδομένων ουράς με πέντε στοιχεία. Στην εικόνα φαίνεται επίσης ο τρόπος υλοποίησης της ουράς. Στην υλοποίηση της συστοιχίας έχουμε μια εγγραφή με τρία πεδία. Το πρώτο πεδίο μπορεί να χρησιμοποιηθεί για την αποθήκευση πληροφοριών που αφορούν την ουρά.

Παρόμοια είναι και η υλοποίηση της συνδεδεμένης λίστας: υπάρχει ένας επιπλέον κόμβος που έχει το όνομα της ουράς. Ο κόμβος αυτός διαθέτει τρία πεδία: ένα πεδίο καταμέτρησης, έναν δείκτη που δείχνει στο στοιχείο στην αρχή της ουράς, και έναν δείκτη που δείχνει στο στοιχείο στο τέλος της ουράς.



Εικόνα 12.13 Υλοποίηση ουράς

12-4 ΓΕΝΙΚΕΣ ΓΡΑΜΜΙΚΕΣ ΛΙΣΤΕΣ

Οι στοίβες και οι ουρές που περιγράφηκαν στις δύο προηγούμενες ενότητες είναι **περιορισμένες γραμμικές λίστες**. Σε μια **γενική γραμμική λίστα** οι λειτουργίες, όπως η εισαγωγή και η διαγραφή, μπορούν να εφαρμόζονται σε οποιοδήποτε σημείο της λίστας - στην αρχή, στο τέλος, ή στο μέσο. Στην Εικόνα 12.14 παρουσιάζεται μια γενική γραμμική λίστα.

Εικόνα 12.14 Γενική γραμμική λίστα

Λειτουργίες σε γενικές γραμμικές λίστες

Αν και μπορούμε να ορίσουμε πολλές λειτουργίες σε μια γενική γραμμική λίστα, σε αυτό το κεφάλαιο θα περιγράψουμε μόνο τις έξι πιο συνηθισμένες: τη ***δημιουργία λίστας***, την ***εισαγωγή σε λίστα***, τη ***διαγραφή από λίστα***, την ***ανάκτηση από λίστα***, τη ***διάσχιση λίστας*** και τον ***έλεγχο για κενή λίστα***.

Η λειτουργία *δημιουργίας λίστας*

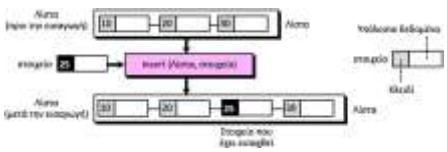
Η λειτουργία δημιουργίας λίστας (list) δημιουργεί μια κενή λίστα. Η μορφή είναι η εξής:

list (όνομαΛίστας)

Η λειτουργία *εισαγωγής σε λίστα*

Από τη στιγμή που θεωρούμε τα δεδομένα μιας γραμμικής λίστας ταξινομημένα, η εισαγωγή πρέπει να γίνεται με τέτοιον τρόπο ώστε να διατηρείται η διάταξη των στοιχείων. Για να καθορίζονται οι θέσεις όπου πρέπει να τοποθετούνται τα νέα στοιχεία, είναι απαραίτητη η εκτέλεση αναζήτησης. Ωστόσο, η αναζήτηση πραγματοποιείται στο επίπεδο υλοποίησης και όχι στο επίπεδο του αφηρημένου τύπου δεδομένων.

insert (όνομαΛίστας, στοιχείο)

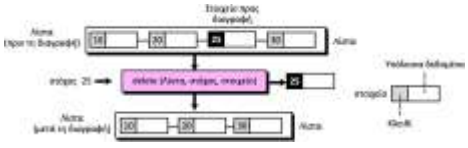


Εικόνα 12.15 **Η λειτουργία εισαγωγής σε λίστα**

Η λειτουργία *διαγραφής από λίστα*

Για τη διαγραφή δεδομένων από μια γενική λίστα (Εικόνα 12.16), αυτά πρέπει πρώτα να εντοπιστούν με αναζήτηση στη λίστα. Όταν βρεθεί η θέση των δεδομένων, τότε μπορεί να γίνει η διαγραφή τους. Η μορφή είναι η εξής:

delete (όνομαΛίστας, στόχος, στοιχείο)

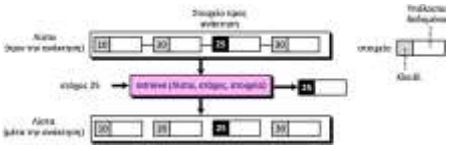


Εικόνα 12.16 **Η λειτουργία διαγραφής από λίστα**

Η λειτουργία *ανάκτησης*

Με τον όρο *ανάκτηση* εννοούμε την προσπέλαση ενός μόνο στοιχείου. Όπως και με την εισαγωγή και τη διαγραφή, θα πρέπει πρώτα να εκτελεστεί αναζήτηση στη λίστα και, σε περίπτωση που τα δεδομένα βρεθούν, τότε μπορούν να ανακτηθούν. Η μορφή της λειτουργίας ανάκτησης είναι η εξής:

retrieve (όνομαΛίστας, στόχος, στοιχείο)



Εικόνα 12.17 **Η λειτουργία ανάκτησης**

Η λειτουργία *διάσχισης*

Σε κάθε μία από τις προηγούμενες λειτουργίες η λίστα προσπελάζεται τυχαία για τον εντοπισμό ενός συγκεκριμένου στοιχείου. Η διάσχιση λίστας, από την άλλη, έχει να κάνει με ακολουθιακή προσπέλαση. Πρόκειται για μια λειτουργία κατά την οποία όλα τα στοιχεία της λίστας περνούν με τη σειρά από επεξεργασία, ένα προς ένα. Η μορφή είναι η εξής:

traverse (όνομαΛίστας, ενέργεια)

Η λειτουργία *ελέγχου για κενή λίστα*

Η λειτουργία ελέγχου για κενή λίστα ελέγχει την κατάσταση της λίστας. Η μορφή είναι η εξής:

empty (όνομαΛίστας)

Η λειτουργία αυτή επιστρέφει την τιμή true αν η λίστα είναι κενή ή την τιμή false αν η λίστα δεν είναι κενή.

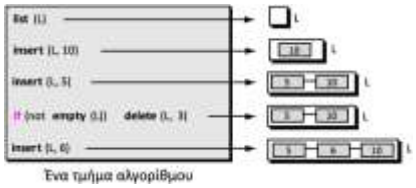
Αφηρημένος τύπος δεδομένων γενικής γραμμικής λίστας

Μια γενική γραμμική λίστα ορίζεται ως αφηρημένος τύπος δεδομένων ως εξής:

Αφηρημένος τύπος δεδομένων γενικής γραμμικής λίστας	
Ορισμός	Μια λίστα με ταξινομημένα στοιχεία δεδομένων, όλα του ίδιου τύπου
Λειτουργίες	**list:** Δημιουργεί μια κενή λίστα. **insert:** Προσθέτει ένα στοιχείο στη λίστα. **delete:** Διαγράφει ένα στοιχείο από τη λίστα. **retrieve:** Ανακτά ένα στοιχείο από τη λίστα. **traverse:** Διασχίζει τη λίστα ακολουθιακά. **empty:** Ελέγχει την κατάσταση της λίστας.

Παράδειγμα 12.7

Στην Εικόνα 12.18 παρουσιάζεται ένα τμήμα του αλγορίθμου με το οποίο εφαρμόζονται οι λειτουργίες που ορίζονται παραπάνω σε μια ουρά L. Σημειώστε ότι με την τρίτη και την πέμπτη λειτουργία τα νέα δεδομένα εισάγονται στη σωστή θέση επειδή η λειτουργία εισαγωγής καλεί τον αλγόριθμο αναζήτησης στο επίπεδο υλοποίησης ώστε να βρεθεί η θέση στην οποία πρέπει να εισαχθούν τα νέα δεδομένα. Η τέταρτη λειτουργία δεν διαγράφει το στοιχείο με την τιμή 3 επειδή δεν ανήκει στη λίστα.



Ένα τμήμα αλγορίθμου

Εικόνα 12.18 Παράδειγμα 12.7

Εφαρμογές γενικών γραμμικών λιστών

Οι γενικές γραμμικές λίστες χρησιμοποιούνται σε περιπτώσεις όπου τα στοιχεία προσπελάζονται ακολουθιακά ή σε τυχαία σειρά. Για παράδειγμα, μια γραμμική λίστα μπορεί να χρησιμοποιηθεί σε ένα πανεπιστήμιο για την αποθήκευση των πληροφοριών των σπουδαστών που εγγράφονται σε κάθε εξάμηνο.

Παράδειγμα 12.8

Ας υποθέσουμε ότι σε ένα πανεπιστήμιο χρησιμοποιείται μια γενική γραμμική λίστα που περιέχει πληροφορίες σχετικά με τους σπουδαστές και ότι κάθε στοιχείο δεδομένων είναι μια εγγραφή με τρία πεδία: ID (αναγνωριστικό), Name (όνομα), και Grade (βαθμός). Ο Αλγόριθμος 12.4 βοηθά τους καθηγητές να αλλάζουν τον βαθμό των σπουδαστών. Η λειτουργία διαγραφής αφαιρεί ένα στοιχείο από τη λίστα, το οποίο όμως παρέχεται στο πρόγραμμα ώστε να μπορεί να αλλάζει τον βαθμό. Η λειτουργία εισαγωγής εισάγει πάλι το τροποποιημένο στοιχείο στη λίστα. Το στοιχείο περιέχει ολόκληρη την εγγραφή του σπουδαστή, και ο στόχος είναι το αναγνωριστικό (ID) που χρησιμοποιείται για την αναζήτηση στη λίστα.

Παράδειγμα 12.8 (Συνέχεια)

Αλγόριθμος 12.4 Παράδειγμα 12.8

```
Αλγόριθμος: ChangeGrade (StudentList, target, grade)
Σκοπός: Αλλαγή του βαθμού ενός σπουδαστή
Προ-συνθήκη: Δίνεται η λίστα των μαθητών και οι βαθμοί τους
Μετα-συνθήκη: Τίποτα
Επιστρέφεται: Τίποτα
{
    delete (StudentList, target, element)
    (element.data).Grade ← grade
    insert (StudentList, element)
    return
}
```


Παράδειγμα 12.9

Συνεχίζοντας με το Παράδειγμα 12.8, υποθέστε ότι ο καθηγητής θέλει να τυπώσει τις εγγραφές όλων των σπουδαστών στο τέλος του εξαμήνου. Αυτό μπορεί να γίνει με τον Αλγόριθμο 12.5. Υποθέτουμε ότι υπάρχει ένας αλγόριθμος με όνομα Print που τυπώνει τα περιεχόμενα των εγγραφών. Για κάθε κόμβο, η λειτουργία διάσχισης της λίστας καλεί τον αλγόριθμο Print ο οποίος μεταβιβάζει σε αυτήν τα δεδομένα που θα τυπωθούν.

Παράδειγμα 12.9 (Συνέχεια)

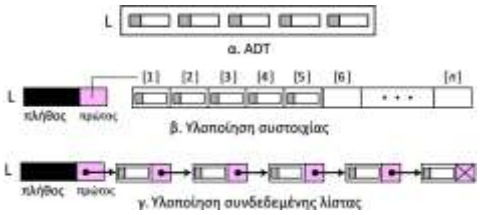
Αλγόριθμος 12.5 Παράδειγμα 12.9

```
Αλγόριθμος: PrintRecord (StudentList)
Σκοπός: Εκτύπωση των εγγραφών όλων των σπουδαστών στη λίστα StudentList
Προ-συνθήκη: Δίνεται η λίστα των σπουδαστών
Μετα-συνθήκη: Τίποτα
Επιστρέφεται: Τίποτα
{
    traverse (StudentList, Print)
    return
}
```

Υλοποίηση γενικής γραμμικής λίστας

Στο επίπεδο του αφηρημένου τύπου δεδομένων χρησιμοποιείται η λίστα και οι έξι λειτουργίες της, όμως στο επίπεδο υλοποίησης πρέπει να επιλεγεί μια δομή δεδομένων και να υλοποιηθεί. Ο αφηρημένος τύπος δεδομένων γενικής λίστας μπορεί να υλοποιηθεί είτε ως συστοιχία είτε ως συνδεδεμένη λίστα. Στην Εικόνα 12.19 παρουσιάζεται ένα παράδειγμα αφηρημένου τύπου δεδομένων λίστας με πέντε στοιχεία. Στην εικόνα φαίνεται επίσης ο τρόπος υλοποίησης της λίστας.

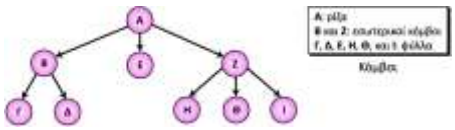
Παρόμοια είναι και η υλοποίηση της συνδεδεμένης λίστας: υπάρχει ένας επιπλέον κόμβος που έχει το όνομα της λίστας. Ο κόμβος αυτός διαθέτει δύο πεδία, έναν μετρητή και έναν δείκτη που δείχνει στο πρώτο στοιχείο.



Εικόνα 12.19 Υλοποίηση γενικής γραμμικής λίστας

12-5 ΔΕΝΤΡΑ

Ένα **δέντρο** (tree) αποτελείται από ένα πεπερασμένο σύνολο στοιχείων, τους **κόμβους** (nodes, ή **κορυφές**, vertices), και από ένα πεπερασμένο σύνολο προσανατολισμένων **γραμμών**, τα **τόξα** (arcs), τα οποία συνδέουν ζεύγη κόμβων.



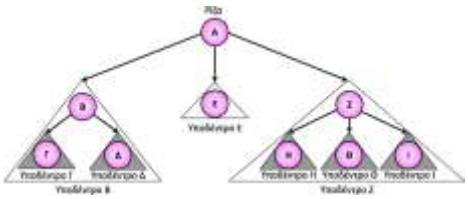
Εικόνα 12.20 Αναπαράσταση δέντρου

Οι κορυφές (vertices) ενός δέντρου χωρίζονται σε τρεις κατηγορίες: τη *ρίζα*, τα *φύλλα*, και τους *εσωτερικούς κόμβους*. Στον Πίνακα 12.1 φαίνεται το πλήθος των εξερχόμενων και εισερχόμενων τόξων που επιτρέπεται για κάθε τύπο κόμβου.

Πίνακας 12.1 Πλήθος εισερχόμενων και εξερχόμενων τόξων

Τύπος κόμβου	*Εισερχόμενα τόξα*	*Εξερχόμενα τόξα*
ρίζα	0	0 ή περισσότερα
φύλλο	1	0
εσωτερικός	1	1 ή περισσότερα

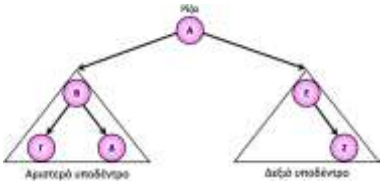
Κάθε κόμβος σε ένα δέντρο μπορεί να έχει ένα **υποδέντρο** (subtree). Το υποδέντρο κάθε κόμβου περιλαμβάνει έναν από τους θυγατρικούς κόμβους και όλους τους απογόνους του. Στην Εικόνα 12.21 παρουσιάζονται όλα τα υποδέντρα για το δέντρο της Εικόνας 12.20.



Εικόνα 12.21 Υποδέντρα

12-6 ΔΥΑΔΙΚΑ ΔΕΝΤΡΑ

Ένα δυαδικό δέντρο είναι ένα δέντρο στο οποίο κανένας κόμβος δεν έχει περισσότερα από δύο υποδέντρα. Με άλλα λόγια, ένας κόμβος μπορεί να έχει μηδέν, ένα, ή δύο υποδέντρα.



Εικόνα 12.22 Ένα δυαδικό δέντρο

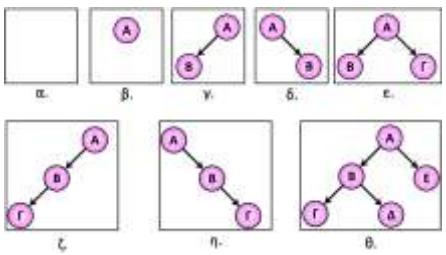
Αναδρομικός ορισμός των δυαδικών δέντρων

Στο Κεφάλαιο 8 παρουσιάσαμε τον αναδρομικό ορισμό ενός αλγορίθμου. Με αναδρομικό τρόπο μπορούμε να ορίσουμε και μια δομή ή έναν αφηρημένο τύπο δεδομένων. Στη συνέχεια δίνεται ο αναδρομικός ορισμός ενός δυαδικού δέντρου. Σημειώστε ότι, με βάση τον ορισμό αυτό, ρίζα μπορεί να έχει και το ίδιο το δυαδικό δέντρο αλλά και κάθε υποδέντρο.

Δυαδικό δέντρο

Ορισμός Ένα δυαδικό δέγτρο είναι είτε κενό είτε αποτελείται από έναν κόμβο, τη ρίζα, με δύο υποδέντρα, το κάθε ένα από τα οποία είναι επίσης ένα δυαδικό δέντρο.

Στην Εικόνα 12.23 φαίνονται οκτώ δέντρα, το πρώτο από τα οποία είναι ένα κενό (null) δυαδικό δέντρο.



Εικόνα 12.23 Παραδείγματα δυαδικών δέντρων

Λειτουργίες σε δυαδικά δέντρα

Οι έξι πιο συνηθισμένες λειτουργίες δυαδικών δέντρων είναι η ***δημιουργία δέντρου*** (δημιουργεί ένα κενό δέντρο), η ***εισαγωγή σε δέντρο***, η ***διαγραφή από δέντρο***, η ***ανάκτηση από δέντρο***, ο ***έλεγχος για κενό δέντρο***, και η ***διάσχιση δέντρου***. Οι πρώτες πέντε είναι αρκετά περίπλοκες και εκτός της εμβέλειας αυτού του βιβλίου. Σε αυτή την ενότητα θα ασχοληθούμε μόνο με τη διάσχιση δυαδικού δέντρου.

12.61

Διασχίσεις δυαδικών δέντρων

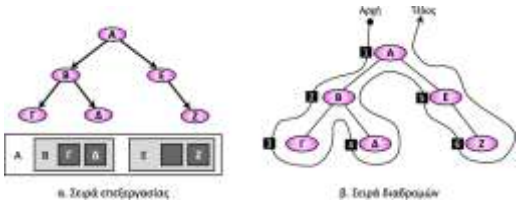
Μια διάσχιση δυαδικού δέντρου προϋποθέτει ότι η επεξεργασία κάθε κόμβου του δέντρου γίνεται μία και μοναδική φορά με μια προκαθορισμένη σειρά. Οι δύο γενικές προσεγγίσεις για την ακολουθία της διάσχισης είναι με **προτεραιότητα βάθους** και με **προτεραιότητα εύρους**.

Εικόνα 12.24 Διάσχιση προτεραιότητας βάθους ενός δυαδικού δέντρου

12.62

Παράδειγμα 12.10

Στην Εικόνα 12.25 μπορείτε να δείτε πώς φτάνουμε σε κάθε κόμβο ενός δέντρου με προδιατεταγμένη διάσχιση. Στην εικόνα φαίνεται επίσης η σειρά διαδρομών. Στην προδιατεταγμένη διάσχιση φτάνουμε σε έναν κόμβο αφού πρώτα περάσουμε από την αριστερή πλευρά του. Οι κόμβοι προσπελάζονται με την εξής σειρά: Α, Β, Γ, Δ, Ε, Ζ.

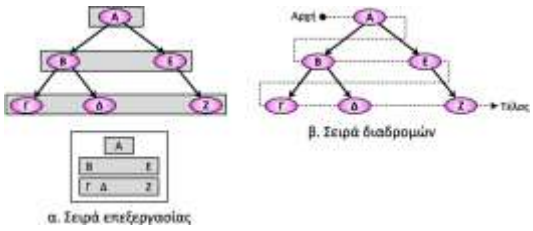


Εικόνα 12.25 Παράδειγμα 12.10

12.63

Παράδειγμα 12.11

Στην Εικόνα 12.26 μπορείτε να δείτε πώς φτάνουμε σε κάθε κόμβο ενός δέντρου με διάσχιση προτεραιότητας εύρους. Στην εικόνα φαίνεται επίσης η σειρά διαδρομών. Η σειρά της διάσχισης είναι Α, Β, Ε, Γ, Δ, Ζ.



Εικόνα 12.26 Παράδειγμα 12.11

12.64

Εφαρμογές δυαδικού δέντρου

Τα δυαδικά δέντρα έχουν πολλές εφαρμογές στην επιστήμη των υπολογιστών. Σε αυτή την ενότητα θα περιγράψουμε μόνο δύο από αυτές: την κωδικοποίηση Huffman και τα δέντρα παράστασης.

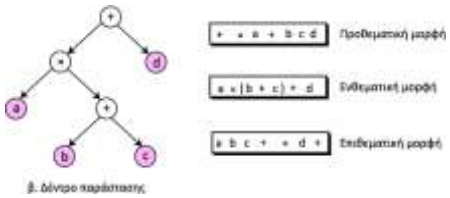
Κωδικοποίηση Huffman

Η κωδικοποίηση Huffman είναι μια τεχνική συμπίεσης που χρησιμοποιεί δυαδικά δέντρα για να δημιουργεί δυαδικό κώδικα μεταβλητού μήκους από μια συμβολοσειρά. Η κωδικοποίηση Huffman περιγράφεται με περισσότερες λεπτομέρειες στο Κεφάλαιο 15.

Δέντρα παράστασης

Μια αριθμητική παράσταση μπορεί να αναπαρασταθεί σε τρεις διαφορετικές μορφές: την **ενθεματική** (infix), την **επιθεματική** (postfix), και την **προθεματική** (prefix). Στην ενθεματική μορφή, ο τελεστής τοποθετείται μεταξύ των δύο τελεστέων. Στην επιθεματική μορφή, ο τελεστής τοποθετείται μετά από τους δύο τελεστέους του, και στην προθεματική μορφή τοποθετείται πριν από τους δύο τελεστέους του. Οι μορφές αυτές παρουσιάζονται παρακάτω για την πράξη της πρόσθεσης δύο τελεστέων A και B.

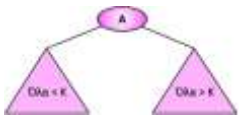
Προθεματική: + A B **Ενθεματική:** A + B **Επιθεματική:** A B +



Εικόνα 12.27 Δέντρο παράστασης

12-7 ΔΥΑΔΙΚΑ ΔΕΝΤΡΑ ΑΝΑΖΗΤΗΣΗΣ

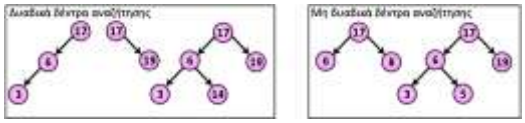
Ένα δυαδικό δέντρο αναζήτησης (binary search tree, BST) είναι ένα δυαδικό δέντρο με μια επιπλέον ιδιότητα: η τιμή του κλειδιού κάθε κόμβου είναι μεγαλύτερη από τις τιμές των κλειδιών όλων των κόμβων σε κάθε δεξιό υποδέντρο και μικρότερη από τις τιμές όλων των κόμβων σε κάθε δεξιό υποδέντρο. Η έννοια αυτή παρουσιάζεται στην Εικόνα 12.28.



Εικόνα 12.28 Δυαδικό δέντρο αναζήτησης (BST)

Παράδειγμα 12.12

Στην Εικόνα 12.29 παρουσιάζονται μερικά δυαδικά δέντρα που είναι δέντρα αναζήτησης και ορισμένα που δεν είναι. Σημειώστε ότι ένα δέντρο είναι δυαδικό δέντρο αναζήτησης όταν ολόκληρο το δέντρο αλλά και όλα τα υποδέντρα του είναι δυαδικά δέντρα αναζήτησης.

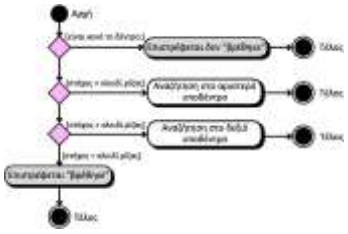


Εικόνα 12.29 **Παράδειγμα 12.12**

Μια πολύ ενδιαφέρουσα ιδιότητα των δυαδικών δέντρων αναζήτησης είναι ότι, αν εφαρμόσουμε την ενδοδιατεταγμένη διάσχιση σε ένα δυαδικό δέντρο, τα στοιχεία που προσπελάζονται ταξινομούνται σε αύξουσα σειρά. Για παράδειγμα, όταν εφαρμοστεί η ενδοδιατεταγμένη διάσχιση στα τρία δυαδικά δέντρα αναζήτησης της Εικόνας 12.29, θα πάρουμε τις λίστες (3, 6, 17), (17, 19), και (3, 6, 14, 17, 19), αντίστοιχα.

Με την ενδοδιατεταγμένη διάσχιση ενός δυαδικού δέντρου αναζήτησης δημιουργείται μια λίστα ταξινομημένη σε αύξουσα σειρά.

Ένα άλλο ενδιαφέρον χαρακτηριστικό των δυαδικών δέντρων αναζήτησης είναι ότι μπορούμε να χρησιμοποιούμε μια έκδοση της δυαδικής αναζήτησης που περιγράφηκε στο Κεφάλαιο 8. Στην Εικόνα 12.30 παρουσιάζεται το διάγραμμα UML για μια αναζήτηση σε ένα δυαδικό δέντρο αναζήτησης.



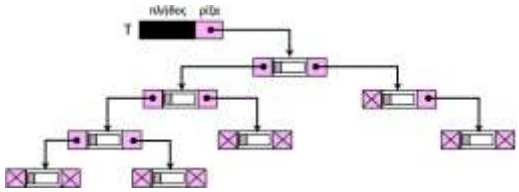
Εικόνα 12.30 **Ενδοδιατεταγμένη διάσχιση ενός δυαδικού δέντρου αναζήτησης**

Αφηρημένοι τύποι δεδομένων δυαδικού δέντρου αναζήτησης

Ο αφηρημένος τύπος δεδομένων για ένα δυαδικό δέντρο αναζήτησης είναι παρόμοιος με αυτόν που ορίσαμε για μια γενική γραμμική λίστα με την ίδια λειτουργία. Μάλιστα, πιο συχνά συναντάμε λίστες δυαδικών δέντρων αναζήτησης από ό,τι γενικές γραμμικές λίστες. Αυτό συμβαίνει επειδή η αναζήτηση ενός δυαδικού δέντρου αναζήτησης είναι πιο αποδοτική από την αναζήτηση μιας γραμμικής λίστας: στις γενικές γραμμικές λίστες χρησιμοποιείται ακολουθιακή αναζήτηση, ενώ στα δυαδικά δέντρα αναζήτησης χρησιμοποιείται μια εκδοχή της δυαδικής αναζήτησης.

Υλοποίηση δυαδικών δέντρων αναζήτησης

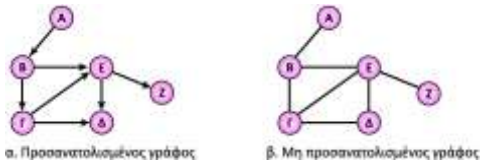
Τα δυαδικά δέντρα αναζήτησης μπορούν να υλοποιούνται με τη χρήση συστοιχιών ή συνδεδεμένων λιστών. Ωστόσο, πιο συνηθισμένες είναι οι δομές συνδεδεμένων λιστών, οι οποίες είναι και πιο αποδοτικές. Στην υλοποίηση χρησιμοποιούνται κόμβοι με δύο δείκτες, έναν αριστερό και έναν δεξιό.



Εικόνα 12.31 Υλοποίηση δυαδικού δέντρου αναζήτησης

12-8 ΓΡΑΦΟΙ

Ο γράφος (graph) είναι ένας αφηρημένος τύπος δεδομένων που αποτελείται από ένα σύνολο κόμβων, οι οποίοι ονομάζονται **κορυφές**, και ένα σύνολο γραμμών, οι οποίες ονομάζονται **ακμές** ή **τόξα** και συνδέουν τις κορυφές. Ενώ ένα δέντρο ορίζει μια ιεραρχική δομή στην οποία ένας κόμβος μπορεί να έχει μόνο έναν γονικό, κάθε κόμβος σε έναν γράφο μπορεί να έχει έναν ή περισσότερους γονικούς. Οι γράφοι μπορεί να είναι είτε **προσανατολισμένοι** είτε μη προσανατολισμένοι. Σε έναν **προσανατολισμένο γράφο** (directed graph, ή digraph) κάθε ακμή, η οποία συνδέει δύο κορυφές, έχει κατεύθυνση από τη μία κορυφή στην άλλη. Σε έναν μη προσανατολισμένο γράφο (undirected graph), δεν υπάρχει καμία κατεύθυνση. Στην Εικόνα 12.32 παρουσιάζεται ένα παράδειγμα προσανατολισμένου γράφου (α) και ένα παράδειγμα μη προσανατολισμένου γράφου (β).



Εικόνα 12.32 Γράφοι

Παράδειγμα 12.13

Σε έναν υπολογιστή μπορεί να χρησιμοποιηθεί ένας μη προσανατολισμένος γράφος για την αναπαράσταση ενός χάρτη πόλεων και των δρόμων που τις συνδέουν. Οι κορυφές είναι οι πόλεις και οι μη προσανατολισμένες ακμές είναι οι δρόμοι που τις συνδέουν. Για να δείξουμε τις αποστάσεις μεταξύ των πόλεων μπορούμε να χρησιμοποιήσουμε σταθμισμένους γράφους (weighted graphs), στους οποίους κάθε ακμή έχει ένα βάρος (weight) που αντιπροσωπεύει την απόσταση μεταξύ δύο πόλεων οι οποίες συνδέονται με αυτή την ακμή.

Παράδειγμα 12.14

Μια άλλη εφαρμογή των γράφων είναι στα δίκτυα υπολογιστών (Κεφάλαιο 6). Οι κορυφές μπορεί να αντιπροσωπεύουν τους κόμβους ή τους διανομείς, και οι ακμές να αντιπροσωπεύουν τα δρομολόγια. Κάθε ακμή μπορεί να έχει ένα βάρος το οποίο ορίζει το κόστος διαδρομής από έναν διανομέα στον γειτονικό του. Ένας δρομολογητής μπορεί να χρησιμοποιεί αλγορίθμους γράφου για να εντοπίζει τη συντομότερη διαδρομή ανάμεσα σε αυτόν και τον τελικό προορισμό ενός πακέτου.