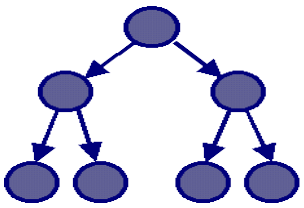
5. Δέντρα



5.1 Γραμμική Αναζήτηση και Δυαδική Αναζήτηση

Σε πολλές εφαρμογές, όταν πρόκειται να αναζητήσουμε ένα στοιχείο μέσα σε μια συλλογή στοιχείων δεδομένων, μπορούμε να οργανώσουμε αυτά τα δεδομένα σαν μια λίστα:

$$L_1, L_2, L_3, ..., L_n$$

Με αναζήτηση στη λίστα αυτή μπορούμε να δούμε αν κάποιο από τα στοιχεία L_i έχει μια συγκεκριμένη τιμή. Συνήθως, τα στοιχεία της λίστας είναι εγγραφές, οπότε η αναζήτηση γίνεται με βάση ένα πεδίο κλειδί αυτών των εγγραφών. Με άλλα λόγια, ψάχνουμε να βρούμε μια εγγραφή L_i της λίστας, που να περιέχει μια συγκεκριμένη τιμή στο πεδίο κλειδί. Για λόγους απλότητας, θα θεωρήσουμε ότι τα στοιχεία L_i είναι απλά στοιχεία και όχι εγγραφές.

Όταν εκτελούμε **γραμμική αναζήτηση** (**linear search**), ξεκινάμε με το πρώτο στοιχείο της λίστας και ανιχνεύουμε τη λίστα σειριακά μέχρι να βρούμε το στοιχείο που αναζητάμε ή μέχρι να φτάσουμε στο τέλος της λίστας. Αν υποθέσουμε ότι χρησιμοποιείται η υλοποίηση λίστας με σειριακή αποθήκευση σε πίνακα, τότε η διαδικασία γραμμικής αναζήτησης μιας λίστας μπορεί να είναι η ακόλουθη:

```
void LinearSearch (ListType L, int n, ListElementType Item, boolean *Found,
                   int *Location;
/*Δέχεται:    Μια λίστα n στοιχείων αποθηκευμένα στον πίνακα L και ένα στοιχείο Item.
Λειτουργία:  Εκτελεί γραμμική αναζήτηση των στοιχείων L[1], L[2], ..., L[n] για το
             στοιχείο Item.
Επιστρέφει:  Found=TRUE και στην Location την θέση του στοιχείου Item, αν η αναζήτηση
             είναι επιτυχής, διαφορετικά Found = FALSE */
{
   boolean Loc;
  *Found = FALSE;
  (*Location) = 1;
  while (!(*Found) && (*Location) <=n)
      if (Item == L[*Location])
        (*Found) = TRUE;
      else
        (*Location)++;
}
```

Στην παραπάνω διαδικασία υποθέτουμε ότι το αναγνωριστικό ListType έχει δηλωθεί ως τύπος πίνακα στοιχείων τύπου ListElementType και οι δείκτες του παίρνουν τιμές από 1 έως μια σταθερά ListLimit, δηλαδή: typedef ListElementType ListType[ListLimit].

Παρόμοια είναι μια διαδικασία για γραμμική αναζήτηση μιας συνδεδεμένης λίστας. Αντί για τη μεταβλητή Location μπορούμε να χρησιμοποιήσουμε έναν δείκτη LocPtr, ο οποίος αρχικά να δείχνει στον πρώτο κόμβο και να μετακινείται από τον έναν κόμβο στον άλλο ακολουθώντας τους δεσμούς, όπως φαίνεται παρακάτω:

```
void LinkedLinearSearch(ListPointer List, ListElementType Item, boolean
                        *Found, ListPointer *LocPtr)

/*Δέχεται:   Μια συνδεδεμένη λίστα, με τον δείκτη L να δείχνει στον πρώτο κόμβο, και
             ένα στοιχείο Item.
Λειτουργία: Εκτελεί γραμμική αναζήτηση σ' αυτήν την συνδεδεμένη λίστα για έναν κόμβο
             που να περιέχει το στοιχείο Item.
Επιστρέφει: Found=TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που περιέχει το
             στοιχείο Item, αν η αναζήτηση είναι επιτυχής, διαφορετικά Found=FALSE.*/
{
  *Found=FALSE;
  *LocPtr=List;
  while (!Found && (*LocPtr)!= NULL)
       if (Item==(*LocPtr)->Data)
         (*Found)=TRUE;
       else
         (*LocPtr)=(*LocPtr)->Next;
}
```

Για την διαδικασία αυτή υποθέτουμε ότι οι τύποι ListPointer, και ListElementType είναι ορισμένοι όπως στην ενότητα 4.5 για τις συνδεδεμένες λίστες με χρήση δεικτών.

Στην καλύτερη περίπτωση το στοιχείο που αναζητούμε είναι το πρώτο στοιχείο της λίστας, ενώ στην χειρότερη το στοιχείο δεν υπάρχει στη λίστα, οπότε πρέπει να εξετάσουμε όλους τους κόμβους. Ωστόσο, αν πρόκειται για διατεταγμένη λίστα, δηλαδή αν τα στοιχεία είναι οργανωμένα σε αύξουσα (ή φθίνουσα) διάταξη, τότε υπάρχει συνήθως η δυνατότητα να καθοριστεί αν το στοιχείο δεν είναι μέσα στη λίστα, χωρίς να χρειάζεται να εξεταστούν όλα τα στοιχεία της λίστας. Μόλις εντοπιστεί στοιχείο μεγαλύτερο από αυτό που αναζητούμε, η αναζήτηση σταματά. Έστω, για παράδειγμα, ότι αναζητούμε τον αριθμό 12 στην παρακάτω λίστα:

1, 3, 5, 7, 9, 11, 13, 15, 17, 19

Όταν φτάσουμε στον αριθμό 13, δεν υπάρχει λόγος να συνεχίσουμε την αναζήτηση, αφού ο 13 είναι μεγαλύτερος από τον 12 κι επομένως και οι αριθμοί που έπονται του 13 είναι επίσης μεγαλύτεροι του 12.

Ένας αλγόριθμος για την γραμμική αναζήτηση σε διατεταγμένη λίστα είναι ο ακόλουθος:

ΓΡΑΜΜΙΚΗ ΑΝΑΖΗΤΗΣΗ ΓΙΑ ΔΙΑΤΕΤΑΓΜΕΝΕΣ ΛΙΣΤΕΣ

/*Δέχεται: Μια διατεταγμένη λίστα *L1, L2, ..., Ln* με τα στοιχεία σε αύξουσα διάταξη και ένα στοιχείο *Item*.

Λειτουργία: Εκτελεί μια γραμμική αναζήτηση της διατεταγμένης λίστας για το στοιχείο *Item*.

Επιστρέφει: *Found*=TRUE και η *Location* είναι ίση με τη θέση του στοιχείου *Item*, αν η αναζήτηση είναι επιτυχής, διαφορετικά *Found*=FALSE.*/

```
1. Found ← FALSE          /*θέσε στη μεταβλητή Found που δηλώνει αν βρέθηκε ή όχι το στοιχείο Item στη διατεταγμένη λίστα την τιμή FALSE*/
   DoneSearching ← FALSE  /*θέσε στη μεταβλητή DoneSearching που δηλώνει αν τελείωσε η αναζήτηση του στοιχείου Item στη διατεταγμένη λίστα την τιμή FALSE*/
   Location ← 1           /*αρχικοποίηση της μεταβλητής Location ώστε να δείχνει στην 1η θέση της λίστας*/
2. Όσο DoneSearching = FALSE επανάλαβε
                          /*όσο δεν τελείωσε η αναζήτηση, δηλαδή όσο δεν βρέθηκε το στοιχείο Item και δεν φτάσαμε στο τέλος της λίστας*/
      Αν Item = L_Location τότε
                          /*αν το στοιχείο Item ισούται με το στοιχείο που βρίσκεται στη θέση Location της διατεταγμένης λίστας*/
         Found ← TRUE     /*το στοιχείο Item βρέθηκε στη θέση Location της διατεταγμένης λίστας*/
         DoneSearching ← TRUE
                          /*η αναζήτηση τελείωσε*/
      Αλλιώς_αν Item < L_Location ή Location = n τότε
                          /*αν το στοιχείο Item είναι μικρότερο από το στοιχείο που βρίσκεται στη θέση Location της διατεταγμένης λίστας ή το τρέχον στοιχείο (θέση Location) της διατεταγμένης λίστας είναι το τελευταίο (θέση n)*/
         DoneSearching ← TRUE
                          /*η αναζήτηση τελείωσε*/
      Αλλιώς
         Location ← Location + 1
                          /*αύξησε την τιμή της μεταβλητής Location κατά 1, έτσι ώστε αυτή να αναφέρεται στο επόμενο στοιχείο της διατεταγμένης λίστας*/
      Τέλος_αν
   Τέλος_επανάληψης
```

Αν το στοιχείο Item είναι κάποιο από τα L_1, L_2, ..., L_n, τότε κατά μέσο όρο θα γίνουν $n/2$ συγκρίσεις του Item με το L_i μέχρι να τερματιστεί η επανάληψη. Αν, όμως, το ζητούμενο στοιχείο είναι μικρότερο από όλα τα L_i, τότε η σύγκριση του Item με το $L_{Location}$ στο πρώτο πέρασμα από το βρόχο ενώ του παραπάνω αλγορίθμου θα θέσει την DoneSearching ίση με TRUE και ο βρόχος θα τερματιστεί. Όπως είναι φανερό, η γραμμική αναζήτηση για διατεταγμένες λίστες μπορεί να απαιτεί λιγότερες συγκρίσεις από ό,τι για μη διατεταγμένες λίστες. Στην χειρότερη περίπτωση, το Item είναι μεγαλύτερο από το L_n, πράγμα που σημαίνει ότι πρέπει να συγκρίνουμε το Item με όλα τα στοιχεία τις λίστας, όπως και στην μη διατεταγμένη λίστα.

Εκτός από γραμμική αναζήτηση, σε μια διατεταγμένη λίστα μπορούμε να εφαρμόσουμε και **δυαδική αναζήτηση** (**binary search**), όπως δείχνει και ο παρακάτω αλγόριθμος:

ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

/*Δέχεται: Μια διατεταγμένη λίστα *L1, L2, ..., Ln* με τα στοιχεία σε αύξουσα διάταξη και ένα στοιχείο *Item*.

Λειτουργία: Εκτελεί μια δυαδική αναζήτηση της διατεταγμένης λίστας για το στοιχείο *Item*.

Επιστρέφει: *Found*=TRUE και η *Location* είναι ίση με τη θέση του στοιχείου *Item*, αν η αναζήτηση είναι επιτυχής, διαφορετικά*Found*=FALSE.*/

. *Found* ← **FALSE**

/*Σε κάθε επανάληψη της δυαδικής αναζήτησης συγκρίνεται το στοιχείο *Item* με το μεσαίο στοιχείο μιας υπολίστας, η οποία στη συνέχεια χωρίζεται σε 2 άλλες υπολίστες. Η μία από τις υπολίστες αυτές απορρίπτεται και η αναζήτηση συνεχίζεται στην άλλη. Τα στοιχεία της αρχικής διατεταγμένης λίστας που ανήκουν στην υπό εξέταση υπολίστα καθορίζονται από τους δείκτες *First* και *Last* που δείχνουν τη θέση του 1ου και του τελευταίου στοιχείου της υπολίστας αντίστοιχα.*/

First ← 1 /*Το στοιχείο που εξετάζεται πρώτο είναι το μεσαίο στοιχείο της διατεταγμένης λίστας, άρα ο δείκτης *First* παίρνει την αρχική τιμή 1 και ο *Last* την τιμή *n**/

Last ← n

2. **Όσο** *Found* = **FALSE και** *First* <= *Last* **επανάλαβε**

/*όσο δεν βρέθηκε το στοιχείο και εφόσον δεν έχουν ανιχνευθεί όλα τα στοιχεία, δηλαδή ο δείκτης *First* έχει τιμή μικρότερη ή ίση του *Last**/

α. *Location* ← (*First* + *Last*)/2 /*υπολογισμός της θέσης *Location* του μεσαίου στοιχείου της τρέχουσας υπολίστας*/

β. **Αν** *Item* < $L_{Location}$ **τότε** /*αν το στοιχείο *Item* είναι μικρότερο από το στοιχείο που βρίσκεται στη θέση *Location* της τρέχουσας υπολίστας*/

Last ← *Location* - 1 /*θέτουμε στον δείκτη *Last* την τιμή *Location* - 1, αφού η

	αναζήτηση θα συνεχιστεί στην υπολίστα που βρίσκεται πριν από το στοιχείο της τρέχουσας θέσης *Location* ,δηλαδή στα στοιχεία που βρίσκονται από τη θέση *First* έως τη θέση *Location* - 1 της διατεταγμένης λίστας*/
Αλλιώς_αν *Item* > $L_{Location}$ **τότε**	/*αν το στοιχείο *Item* είναι μεγαλύτερο από το στοιχείο που βρίσκεται στη θέση *Location* της τρέχουσας υπολίστας*/
First ← *Location* + 1	/*θέτουμε στον δείκτη *First* την τιμή *Location* + 1, αφού η αναζήτηση θα συνεχιστεί στην υπολίστα που βρίσκεται μετά από το στοιχείο της τρέχουσας θέσης *Location,* δηλαδή στα στοιχεία που βρίσκονται από τη θέση *Location* + 1 έως τη θέση *Last* της διατεταγμένης λίστας*/
Αλλιώς	
Found ← **TRUE**	/*το στοιχείο *Item* βρέθηκε στη θέση *Location* της διατεταγμένης λίστας*/
Τέλος_αν	
Τέλος_επανάληψης	

Το στοιχείο που εξετάζεται πρώτα είναι το μεσαίο στοιχείο της λίστας. Αν το μεσαίο στοιχείο δεν είναι ίσο με το ζητούμενο στοιχείο, τότε η αναζήτηση συνεχίζεται στο πρώτο μισό της λίστας (αν το στοιχείο Item είναι μικρότερο από το μεσαίο) ή στο τελευταίο μισό της λίστας (αν το στοιχείο Item είναι μεγαλύτερο από το μεσαίο στοιχείο). Κάθε φορά, δηλαδή, που περνάμε από το βρόχο, το μέγεθος της υπολίστας στην οποία συνεχίζουμε την αναζήτηση μειώνεται στο μισό. Αυτό σημαίνει ότι κάνουμε το πολύ $\log_2 n$ συγκρίσεις κι επομένως η δυαδική αναζήτηση πλεονεκτεί έναντι της γραμμικής (εμπειρικά έχει αποδειχθεί ότι για λίστες με περισσότερα από 20 στοιχεία, η δυαδική αναζήτηση έχει καλύτερη απόδοση από την γραμμική).

Ο παραπάνω αλγόριθμος δυαδικής αναζήτησης είναι επαναληπτικός. Μια άλλη εκδοχή για τη δυαδική αναζήτηση είναι να εφαρμόσουμε αναδρομικότητα, αφού κάθε φορά συγκρίνουμε το ζητούμενο στοιχείο με το μεσαίο στοιχείο της λίστας ή υπολίστας και, αν δεν είναι ίσα, συνεχίζουμε την αναζήτηση στο ένα από τα δύο μισά της λίστας ή υπολίστας κατά τον ίδιο ακριβώς τρόπο. Επομένως, ένας αλγόριθμος για αναδρομική δυαδική αναζήτηση φαίνεται παρακάτω:

ΑΝΑΔΡΟΜΙΚΗ ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

/*Δέχεται:	Μια διατεταγμένη λίστα με τα στοιχεία σε αύξουσα διάταξη, ένα δείκτη *Fisrt* στο πρώτο στοιχείο της λίστας, ένα δείκτη *Last* στο τελευταίο στοιχείο της λίστας και ένα στοιχείο *Item.*

Την πρώτη φορά που γίνεται κλήση του αλγορίθμου της δυαδικής αναζήτησης οι δείκτες *First* και *Last* έχουν τιμή 1 και *n* αντίστοιχα οπότε ανιχνεύεται η διατεταγμένη λίστα *L1, L2, ..., Ln*. Σε κάθε επόμενη κλήση ο αλγόριθμος ανιχνεύει μια υπολίστα της αρχικής, η οποία προσδιορίζεται από τους δείκτες *First* και *Last.*

Λειτουργία: Ψάχνει στην διατεταγμένη λίστα για το στοιχείο *Item* χρησιμοποιώντας αναδρομική δυαδική αναζήτηση.

Επιστρέφει: *Found*=TRUE και η *Location* είναι ίση με τη θέση του στοιχείου *Item*, αν η αναζήτηση είναι επιτυχής, διαφορετικά*Found*=FALSE.*/

Αν *First* > *Last* **τότε** /*αν έχουν ανιχνευθεί όλα τα στοιχεία, δηλαδή ο δείκτης *First* έχει τιμή μεγαλύτερη του *Last*/

Found ← **FALSE** /*το στοιχείο *Item* δεν βρέθηκε στη λίστα*/

Αλλιώς

α. *Location* ← (*First* + *Last*)/2

/*υπολογισμός της θέσης *Location* του μεσαίου στοιχείου της τρέχουσας υπολίστας*/

β. **Αν** *Item* < $L_{Location}$ **τότε**

/*αν το στοιχείο *Item* είναι μικρότερο από το στοιχείο που βρίσκεται στη θέση *Location* της τρέχουσας υπολίστας*/

Last ← *Location* - 1 /*θέτουμε στον δείκτη *Last* την τιμή *Location* - 1, αφού η αναζήτηση θα συνεχιστεί στην υπολίστα που βρίσκεται πριν από το στοιχείο της τρέχουσας θέσης *Location,* δηλαδή στα στοιχεία που βρίσκονται από τη θέση *First* έως τη θέση *Location* - 1 της διατεταγμένης λίστας*/

Εφάρμοσε τον αλγόριθμο της δυαδικής αναζήτησης

/*ο αλγόριθμος θα εφαρμοστεί στην υπολίστα που προσδιορίζεται από την τρέχουσα τιμή του δείκτη *First* και την τιμή του *Last* που ενημερώσαμε με την προηγούμενη εντολή*/

Αλλιώς_αν *Item* > $L_{Location}$ **τότε**

/*αν το στοιχείο *Item* είναι μεγαλύτερο από το στοιχείο που βρίσκεται στη θέση *Location* της τρέχουσας υπο λίστας*/

First ← *Location* + 1 /*θέτουμε στον δείκτη *First* την τιμή *Location* + 1, αφού η αναζήτηση θα συνεχιστεί στην υπολίστα που βρίσκεται μετά από το στοιχείο της τρέχουσας θέσης *Location,* δηλαδή στα στοιχεία που βρίσκονται από τη θέση *Location* + 1 έως τη θέση *Last* της διατεταγμένης λίστας*/

Εφάρμοσε τον αλγόριθμο της δυαδικής αναζήτησης

```
                                /*ο αλγόριθμος θα εφαρμοστεί στην υπολίστα που προσδιορίζεται
                                από την τιμή του δείκτη First που ενημερώσαμε με την
                                προηγούμενη εντολή και την τρέχουσα τιμή του Last*/
    Αλλιώς
        Found ← TRUE            /*το στοιχείο Item βρέθηκε στη θέση Location της διατεταγμένης
                                λίστας, οπότε θα τελειώσει και η αναδρομική εκτέλεση του
                                αλγορίθμου της δυαδικής αναζήτησης*/
    Τέλος_αν
Τέλος_αν
```

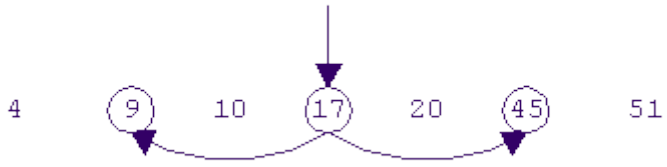
Παρόλο που η δυαδική αναζήτηση συνήθως είναι πιο αποτελεσματική από τη γραμμική, απαιτεί ωστόσο υλοποίηση της λίστας των στοιχείων με σειριακή αποθήκευση, προκειμένου να εξασφαλίζεται άμεση πρόσβαση στα στοιχεία της λίστας. Δεν είναι κατάλληλη για συνδεδεμένες λίστες, γιατί ο εντοπισμός του μεσαίου στοιχείου προϋποθέτει διάσχιση της υπολίστας των στοιχείων που προηγούνται.

Ωστόσο, υπάρχει τρόπος να αποθηκεύσουμε τα στοιχεία μιας διατεταγμένης λίστας σε μια συνδεδεμένη δομή και να την ανιχνεύσουμε με δυαδικό τρόπο. Σαν παράδειγμα, θεωρούμε την παρακάτω λίστα ακεραίων:

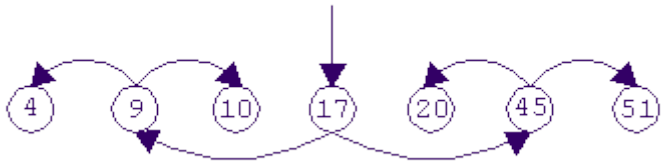
4, 9, 10, 17, 20, 45, 51

Αφού το πρώτο βήμα της δυαδικής αναζήτησης είναι να εξετάσουμε το μεσαίο στοιχείο της λίστας, μπορούμε να έχουμε άμεση πρόσβαση σ' αυτό το στοιχείο, διατηρώντας έναν δείκτη στον κόμβο που το περιέχει:

Στο επόμενο βήμα, η αναζήτηση συνεχίζεται στο ένα από τα δύο μισά της λίστας, αφού εξετάζουμε το μεσαίο στοιχείο της μιας από τις δύο υπολίστες. Για να έχουμε πρόσβαση από τον αρχικά μεσαίο κόμβο σ' αυτά τα δύο στοιχεία, μπορούμε να διατηρούμε δύο δείκτες προς τους κόμβους αυτών των στοιχείων:



Κατά τον ίδιο τρόπο συνεχίζουμε την αναζήτηση στην επόμενη υπολίστα, πράγμα που σημαίνει ότι χρειαζόμαστε αντίστοιχους δείκτες, όπως φαίνεται και στο παρακάτω σχήμα:



Η παραπάνω συνδεδεμένη λίστα μπορεί να σχεδιαστεί σε μορφή δέντρου, όπως δείχνει το ακόλουθο σχήμα. Ένα τέτοιο δέντρο ονομάζεται **δυαδικό δέντρο αναζήτησης** (**binary search tree**) και αποτελεί ειδικό είδος **δυαδικού δέντρου** (**binary tree**).