ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΔΕΝΤΡΑ
5.1 Γραμμική Αναζήτηση και Δυαδική Αναζήτηση

Εισαγωγή

- 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.

Γραμμική αναζήτηση

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:

Διαδικασία γραμμικής αναζήτησης πίνακα

```
void LinearSearch(ListType  List, 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 */
{
        *Found = FALSE;
        *Location =0;
        while (!(*Found) && (*Location) < n)
            if (Item == L[*Location])
                Found = TRUE;
            else
                (*Location)++;
}
```

Στη διαδικασία αυτή υποθέτουμε ότι το αναγνωριστικό **ListType** έχει δηλωθεί ως ένας **πίνακας** στοιχείων τύπου ListElementType και οι δείκτες του παίρνουν τιμές από 0 έως μια σταθερά ListLimit-1.

Διαδικασία γραμμικής αναζήτησης συνδ. λίστας

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 στη διαδικασία που παρουσιάζεται στη συνέχεια.

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.

Διαδικασία γραμμικής αναζήτησης συνδ. λίστας

```
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;
}
```

Ανάλυση αλγορίθμου γραμμικής αναζήτησης

Καλύτερη περίπτωση: το στοιχείο που αναζητούμε είναι το πρώτο στοιχείο της λίστας.

Χειρότερη περίπτωση: το στοιχείο δεν υπάρχει στη λίστα, οπότε πρέπει να εξετάσουμε όλους τους κόμβους.

Διατεταγμένη λίστα: 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.

Παράδειγμα: έστω 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.

Αλγόριθμος γραμ. αναζήτησης διατεταγμένης λίστας

/*Δέχεται: Μια διατεταγμένη λίστα L_1, L_2, ..., L_n 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**
   DoneSearching ← **FALSE**
   Location ← 0
2. **Όσο** *DoneSearching* == **FALSE επανάλαβε**
   Αν *Item* == $L_{Location}$ **τότε**
   Found ← **TRUE**
   DoneSearching ← **TRUE**
   Αλλιώς_αν *Item* < $L_{Location}$ **ή** *Location* == *n-1* **τότε**
   DoneSearching ← TRUE
   Αλλιώς
   Location ← *Location* + 1
   Τέλος_αν
   Τέλος_επανάληψης

Ανάλυση αλγορίθμου

- 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 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.

Αλγόριθμος δυαδικής αναζήτησης διατ. λίστας

/*Δέχεται: Μια διατεταγμένη λίστα $L_1, L_2, ..., L_n$ 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.*/

1. *Found* ← **FALSE**
   First ← 1
   Last ← n
2. **Όσο** *Found* == **FALSE και** *First* <= *Last* **επανάλαβε**
   α. *Location* ← (*First* + *Last*)/2
   β. **Αν** *Item* < $L_{Location}$ **τότε**
   Last ← *Location* - 1
   Αλλιώς_αν *Item* > $L_{Location}$ **τότε**
   First ← *Location* + 1
   Αλλιώς
   Found ← **TRUE**
   Τέλος_αν
   Τέλος_επανάληψης

Ανάλυση αλγορίθμου δυαδικής αναζήτησης

- 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).

Αναδρομική δυαδική αναζήτηση

Ο αλγόριθμος δυαδικής αναζήτησης που παρουσιάστηκε 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.

Στη συνέχεια παρουσιάζεται ένας αλγόριθμος για αναδρομική δυαδική αναζήτηση.

Αλγόριθμος αναδρομικής δυαδικής αναζήτησης

/*Δέχεται: Μια διατεταγμένη λίστα με τα στοιχεία σε αύξουσα διάταξη, ένα δείκτη *First* 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.*/

Αλγόριθμος αναδρομικής δυαδικής αναζήτησης

Αν *First* > *Last* **τότε**
  Found ← **FALSE**
Αλλιώς
  α. *Location* ← (*First* + *Last*)/2
  β. **Αν** *Item* < $L_{Location}$ **τότε**
    Last ← *Location* - 1
    Εφάρμοσε τον αλγόριθμο της δυαδικής αναζήτησης
  Αλλιώς_αν *Item* > $L_{Location}$ **τότε**
    First ← *Location* + 1
    Εφάρμοσε τον αλγόριθμο της δυαδικής αναζήτησης
  Αλλιώς
    Found ← **TRUE**
  Τέλος_αν
Τέλος_αν

Αποθήκευση διατεταγμένης λίστας σε συνδ. δομή

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**.

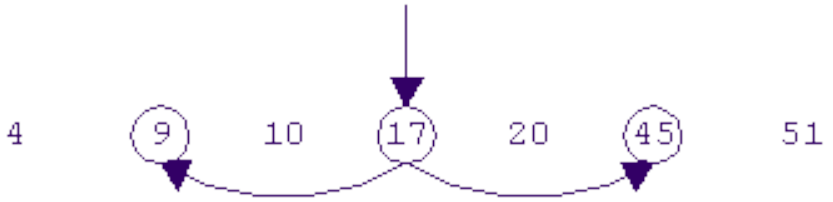
Παράδειγμα διατεταγμένης λίστας σε συνδ. δομή

Παράδειγμα: θεωρούμε την παρακάτω λίστα ακεραίων:

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

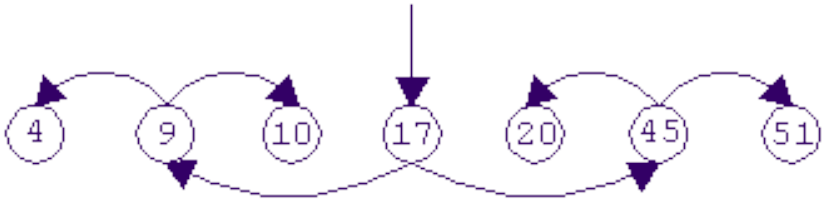
- Αφού το πρώτο βήμα της δυαδικής αναζήτησης είναι να εξετάσουμε το μεσαίο στοιχείο της λίστας, μπορούμε να έχουμε άμεση πρόσβαση σ' αυτό το στοιχείο, διατηρώντας έναν δείκτη στον κόμβο που το περιέχει:

- 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:



Αποθήκευση διατεταγμένης λίστας σε συνδ. δομή

- 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**).