Δομές Δεδομένων

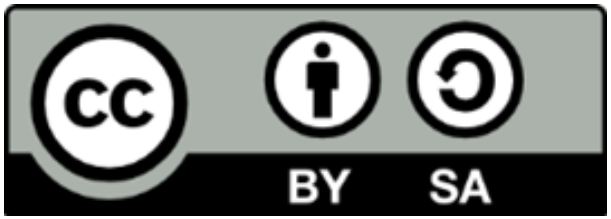
Ενότητα 8: Γραμμική Αναζήτηση και Δυαδική Αναζήτηση-Εισαγωγή στα Δέντρα και Δυαδικά Δέντρα-Δυαδικά Δέντρα Αναζήτησης & Υλοποίηση ΔΔΑ με δείκτες

Καθηγήτρια Μαρία Σατρατζέμη

Τμήμα Εφαρμοσμένης Πληροφορικής

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοποί ενότητας[1]

- Να γνωρίζουν τους αλγορίθμους που επιλύουν το πρόβλημα της αναζήτησης είτε με γραμμική αναζήτηση είτε με δυαδική αναζήτηση.
- Η αποθήκευση των δεδομένων γίνεται με πίνακα ή με συνδεδεμένη λίστα διατεταγμένη ή μη διατεταγμένη.
- Με βάση τη δυαδική αναζήτηση προτείνεται ως αποθηκευτική δομή το δυαδικό δένδρο αναζήτησης.

Σκοποί ενότητας[2]

- Να εισάγει τις Βασικές έννοιες για τα δένδρα, τα Δυαδικά δέντρα, τα Πλήρη δέντρα και τη Χρήση συνδεδεμένων δομών
- Να κατανοήσουν τη δδ Δυαδικό Δένδρο Αναζήτησης
- Τις πράξεις σε ΔΔΑ (εισαγωγή κόμβου, διαγραφή, αναζήτηση) και τις αντίστοιχες υλοποιήσεις του

Περιεχόμενα ενότητας[1]

- Εισαγωγή
- Γραμμική αναζήτηση &Διαδικασία γραμμικής αναζήτησης πίνακα
- Διαδικασία γραμμικής αναζήτησης συνδ. Λίστας
- Ανάλυση αλγορίθμου γραμμικής αναζήτησης
- Αλγόριθμος γραμ. αναζήτησης διατεταγμένης λίστας & Ανάλυση αλγορίθμου
- Αναδρομική δυαδική αναζήτηση

Περιεχόμενα ενότητας[2]

- Βασικές έννοιες
- Βασικές έννοιες: παράδειγμα
- Δυαδικά δέντρα
- Πλήρη δέντρα
- Χρήση συνδεδεμένων δομών

Περιεχόμενα ενότητας[3]

- Δυαδικά δέντρα αναζήτησης
- Δυαδικά δέντρα αναζήτησης: παράδειγμα
- Διαδικασία αναζήτησης σε ΔΔΑ
- Κατασκευή ΔΔΑ
- Διαδικασία εισαγωγής στοιχείου σε ΔΔΑ
- Εισαγωγή στοιχείων σε ΔΔΑ: παράδειγμα
- Διαγραφή κόμβου από ΔΔΑ
- Διαδικασία διαγραφής
- Διαδικασία αναζήτησης σε ΔΔΑ (2)

Γραμμική Αναζήτηση και Δυαδική Αναζήτηση

Εισαγωγή -1-

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

  L1, L2, L3, ..., Ln

- Με αναζήτηση στη λίστα αυτή μπορούμε να δούμε αν κάποιο από τα στοιχεία Li έχει μια συγκεκριμένη τιμή.

Εισαγωγή -2-

- 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 Li της λίστας, που να περιέχει μια συγκεκριμένη τιμή στο πεδίο κλειδί. Για λόγους απλότητας, θα θεωρήσουμε ότι τα στοιχεία Li 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 -1-

void LinearSearch(ListType List, int,
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 *ης μιας λίστας μπορεί να είναι η ακόλουθη.

Διαδικασία γραμμικής αναζήτησης πίνακα -2-

```
{
      *Found = FALSE;
       *Location =1;
     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 1 έως μια σταθερά ListLimit.

Διαδικασία γραμμικής αναζήτησης συνδ. Λίστας -1-

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.*/

Διαδικασία γραμμικής αναζήτησης συνδ. Λίστας -2-

```
{
   *Found = FALSE;
   *LocPtr = List;
   while (!(*Found) && (*LocPtr != NULL)
                if (Item == LocPtr->Data)
                        *Found = TRUE;
                else
                        *LocPtr = LocPtr->Next;
}
```

Ανάλυση αλγορίθμου γραμμικής αναζήτησης -1-

- **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.

Ανάλυση αλγορίθμου γραμμικής αναζήτησης -2-

- 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 -1-

/*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.*/

Αλγόριθμος γραμ. αναζήτησης διατεταγμένης λίστας -2-

1. Found ← FALSE

   DoneSearching ← FALSE

   Location ← 1

Αλγόριθμος γραμ. αναζήτησης διατεταγμένης λίστας -3-

2. Όσο DoneSearching == FALSE επανάλαβε

    Αν Item == LLocation τότε

        Found ← TRUE

        DoneSearching ← TRUE

    Αλλιώς_αν Item < LLocation
                ή Location == n τότε

        DoneSearching ← TRUE

    Αλλιώς Location ← Location + 1

    Τέλος_αν

Τέλος_επανάληψης

Ανάλυση αλγορίθμου -1-

- Αν το στοιχείο Item είναι κάποιο από τα L1, L2, ..., Ln, τότε κατά μέσο όρο θα γίνουν n/2 συγκρίσεις του Item με το Li μέχρι να τερματιστεί η επανάληψη.
- Αν, όμως, το ζητούμενο στοιχείο είναι μικρότερο από όλα τα Li, τότε η σύγκριση του Item με το LLocation στο πρώτο πέρασμα από το βρόχο θα θέσει την DoneSearching ίση με TRUE και ο βρόχος θα τερματιστεί.

Ανάλυση αλγορίθμου -2-

- 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 Ln, 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 -1-

/*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.*/

Αλγόριθμος δυαδικής αναζήτησης διατ. Λίστας -2-

1. Found ← FALSE

   DoneSearching ← FALSE

   Location ← 1

Αλγόριθμος δυαδικής αναζήτησης διατ. Λίστας -3-

1. Όσο Found == FALSE και First <= Last επανάλαβε

    α. Location ← (First + Last)/2

    β. Αν Item < LLocation τότε

        Last ← Location - 1

    Αλλιώς_αν Item > LLocation τότε

        First ← Location + 1

    Αλλιώς Found ← TRUE

    Τέλος_αν

Τέλος_επανάληψης

Ανάλυση αλγορίθμου δυαδικής αναζήτησης -1-

- 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).

Ανάλυση αλγορίθμου δυαδικής αναζήτησης -2-

- 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 log2n 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.

Αλγόριθμος αναδρομικής δυαδικής αναζήτησης-1-

/*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.

Αλγόριθμος αναδρομικής δυαδικής αναζήτησης-2-

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-3-

Αν First > Last τότε

    Found ← FALSE

Αλλιώς

    α. Location ← (First + Last)/2

    β. Αν Item < LLocation τότε

        Last ← Location - 1

Αλγόριθμος αναδρομικής δυαδικής αναζήτησης-4-

Εφάρμοσε τον αλγόριθμο της δυαδικής αναζήτησης

Αλλιώς_αν Item > LLocation τότε

First ← Location + 1

Εφάρμοσε τον αλγόριθμο της δυαδικής αναζήτησης

Αλλιώς

Found ← TRUE

Τέλος_αν

Τέλος_αν

Αποθήκευση διατεταγμένης λίστας σε συνδ. Δομή -1-

- 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.

Αποθήκευση διατεταγμένης λίστας σε συνδ. Δομή -2-

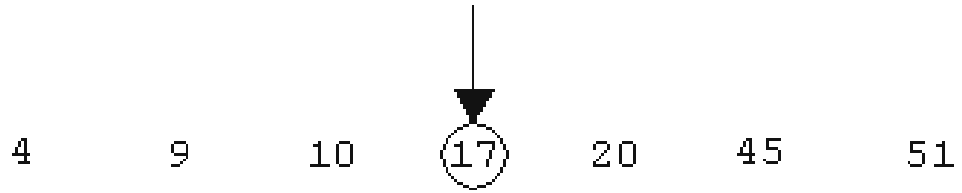
- Ωστόσο, υπάρχει τρόπος να αποθηκεύσουμε τα στοιχεία μιας διατεταγμένης λίστας σε μια συνδεδεμένη δομή και να την ανιχνεύσουμε με δυαδικό τρόπο.

Παράδειγμα διατεταγμένης λίστας σε συνδ. δομή -1-

- Παράδειγμα: θεωρούμε την παρακάτω λίστα ακεραίων:

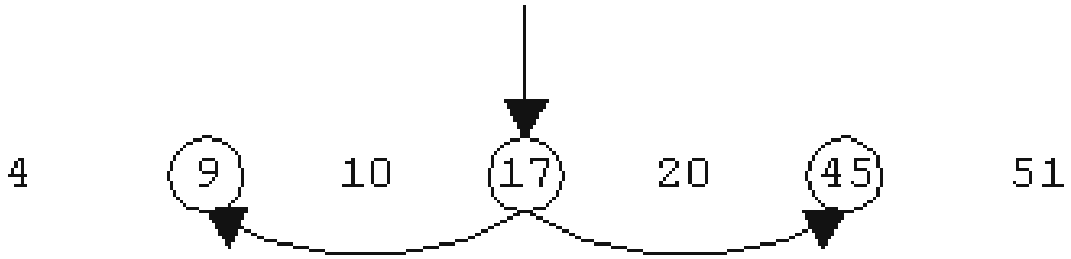
  4, 9, 10, 17, 20, 45, 51

- 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:



Παράδειγμα διατεταγμένης λίστας σε συνδ. δομή -2-

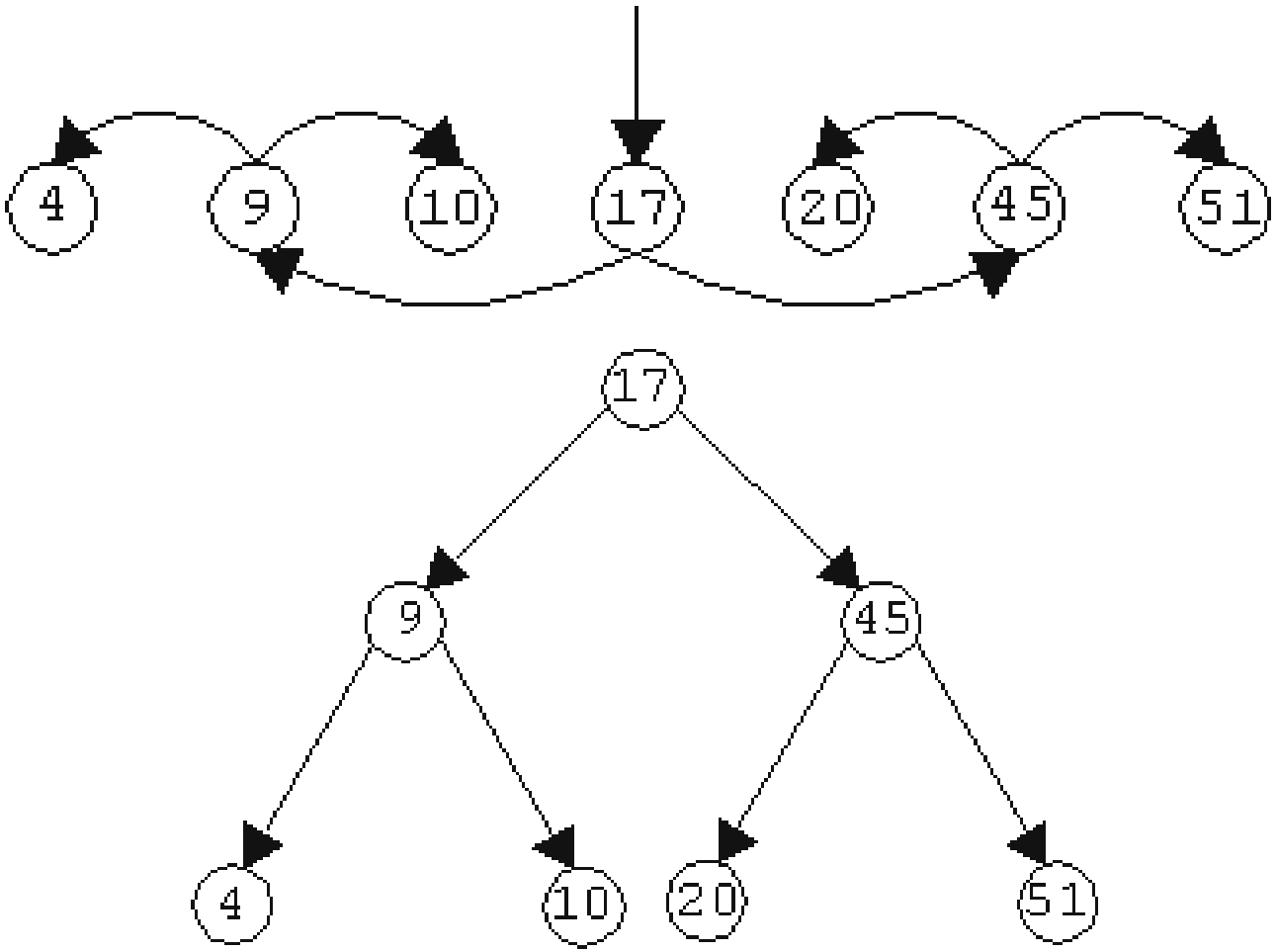
- Στο επόμενο βήμα, η αναζήτηση συνεχίζεται στο ένα από τα δύο μισά της λίστας, αφού εξετάζουμε το μεσαίο στοιχείο της μιας από τις δύο υπολίστες. Για να έχουμε πρόσβαση από τον αρχικά μεσαίο κόμβο σ' αυτά τα δύο στοιχεία, μπορούμε να διατηρούμε δύο δείκτες προς τους κόμβους αυτών των στοιχείων:



Αποθήκευση διατεταγμένης λίστας σε συνδ. Δομή -1-

- Κατά τον ίδιο τρόπο συνεχίζουμε την αναζήτηση στην επόμενη υπολίστα, πράγμα που σημαίνει ότι χρειαζόμαστε αντίστοιχους δείκτες.
- 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).

Αποθήκευση διατεταγμένης λίστας σε συνδ. Δομή -2-



Εισαγωγή στα Δέντρα και Δυαδικά Δέντρα

Βασικές έννοιες -1-

- Ένα **δέντρο** (**tree**) αποτελείται από:
- ένα πεπερασμένο σύνολο στοιχείων που ονομάζονται **κόμβοι** (**nodes**) ή **κορυφές** (**vertices**) και
- από ένα πεπερασμένο σύνολο κατευθυνόμενων τόξων (directed arcs), τα οποία συνδέουν ζεύγη κόμβων.

Βασικές έννοιες -2-

- Αν το δέντρο δεν είναι άδειο, τότε υπάρχει ένας κόμβος, ο οποίος δεν έχει εισερχόμενα τόξα. Αυτός ο κόμβος ονομάζεται **ρίζα** (**root**) του δέντρου και ξεκινώντας από αυτόν μπορούμε να καταλήξουμε σε οποιονδήποτε από τους υπόλοιπους κόμβους ακολουθώντας μια μοναδική διαδρομή από τόξα.
- Οι υπόλοιποι κόμβοι χωρίζονται σε n >= 0 ξένα μεταξύ τους υποσύνολα, T1, T2, ..., Tn, κάθε ένα από τα οποία είναι ένα δέντρο και ονομάζεται **υποδέντρο** (**subtree**) της ρίζας.

Βασικές έννοιες -3-

- Οι κόμβοι που δεν έχουν εξερχόμενα τόξα ονομάζονται **φύλλα** (**leaves**), ενώ οι κόμβοι που δεν είναι ούτε φύλλα ούτε ρίζα ονομάζονται **εσωτερικοί κόμβοι** (**internal nodes**).

- Επίσης, οι κόμβοι που είναι άμεσα προσπελάσιμοι από έναν κόμβο (δηλαδή με χρήση ενός μόνο κατευθυνόμενου τόξου) ονομάζονται **παιδιά** (**children**) του κόμβου αυτού και ο ίδιος ο κόμβος ονομάζεται **πατέρας** (**parent**) των παιδιών του.

Βασικές έννοιες -4-

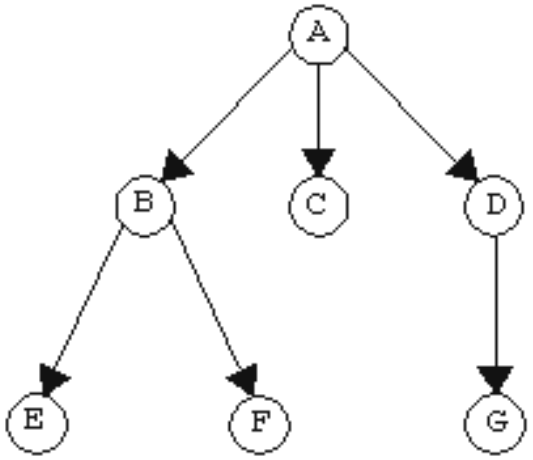
- Οι κόμβοι που είναι παιδιά του ίδιου κόμβου-πατέρα ονομάζονται **αδέρφια** (**siblings**).

- Τέλος, όσον αφορά στους κόμβους ενός δέντρου, ο αριθμός των υποδέντρων ενός κόμβου ονομάζεται **βαθμός του κόμβου** αυτού.

- Ο μέγιστος βαθμός των κόμβων του δέντρου καθορίζει και τον **βαθμό του δέντρου**.

Βασικές έννοιες: παράδειγμα

- Στο συγκεκριμένο δέντρο που έχει βαθμό 3 ή αλλιώς είναι ένα τριαδικό δέντρο:
- ο κόμβος A είναι η ρίζα του δέντρου,
- οι κόμβοι E, F και G είναι φύλλα και
- οι κόμβοι B, C, D είναι εσωτερικοί κόμβοι.

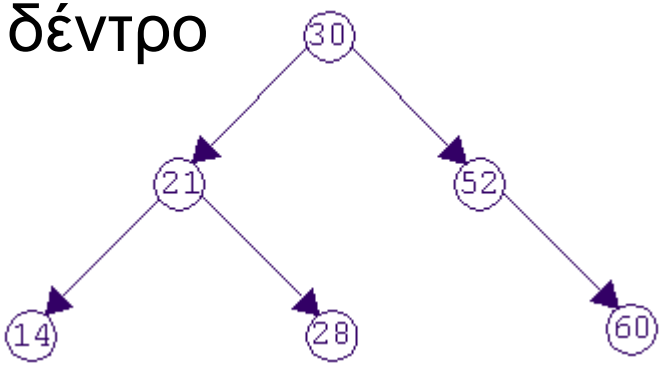


•Ο κόμβος A είναι πατέρας των B, C, D, ο B είναι πατέρας των E και F και ο D είναι πατέρας του κόμβου G.
•Επίσης, οι κόμβοι B, C, D είναι αδέρφια μεταξύ τους και παιδιά του A, οι E και F είναι αδέρφια μεταξύ τους και παιδιά του B, ενώ ο G είναι παιδί του D.

Δυαδικά Δέντρα Αναζήτησης & Υλοποίηση ΔΔΑ με δείκτες

Δυαδικά δέντρα αναζήτησης

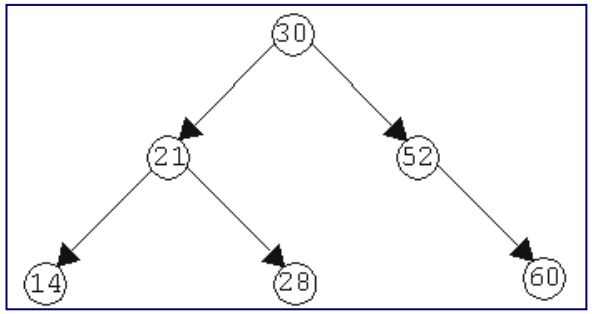
- Στο δυαδικό δέντρο

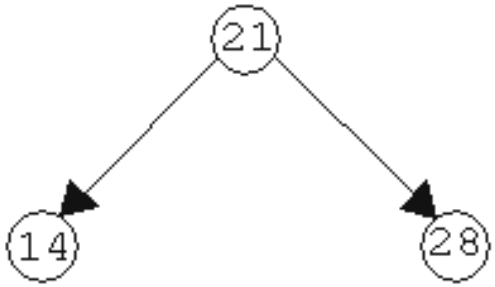


- η τιμή κάθε κόμβου είναι μεγαλύτερη από την τιμή του αριστερού παιδιού του (εφόσον υπάρχει) και μικρότερη από την τιμή του δεξιού παιδιού του (εφόσον υπάρχει).
- Ένα δέντρο με αυτήν την ιδιότητα **ονομάζεται δυαδικό δέντρο αναζήτησης**, **ΔΔΑ**, (**binary search tree, BST**), επειδή μπορούμε να αναζητήσουμε ένα στοιχείο σ' αυτό χρησιμοποιώντας έναν αλγόριθμο δυαδικής αναζήτησης.

Δυαδικά δέντρα αναζήτησης: παράδειγμα -1-

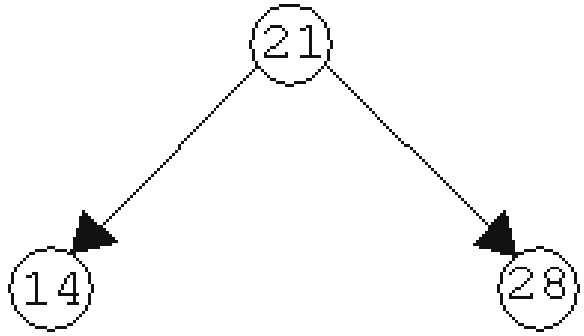
- Έστω, για παράδειγμα ότι αναζητούμε τον αριθμό 14.
- Ξεκινώντας από την ρίζα του δυαδικού δέντρου αναζήτησης, βλέπουμε ότι ο αριθμός 14 είναι μικρότερος του 30, επομένως συμπεραίνουμε ότι ο αριθμός που αναζητούμε βρίσκεται στα αριστερά της ρίζας ή αλλιώς στο αριστερό υποδέντρο (subtree) με ρίζα τον κόμβο 21:



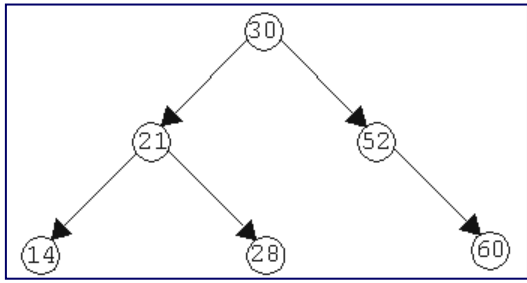


Δυαδικά δέντρα αναζήτησης: παράδειγμα -2-

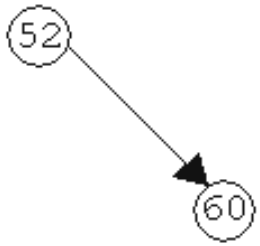
- Η αναζήτηση συνεχίζεται συγκρίνοντας το 14 με την τιμή 21, που βρίσκεται στη ρίζα του παραπάνω υποδέντρου.
- Επειδή το 14 είναι μικρότερο του 21, ο ζητούμενος αριθμός βρίσκεται στο αριστερό υποδέντρο με τιμή 14 στην ρίζα:
- Το παραπάνω υποδέντρο αποτελείται από έναν μόνο κόμβο με τιμή 14.
- Εξετάζοντας αυτήν την τιμή βλέπουμε ότι εντοπίσαμε τον κόμβο που αναζητούσαμε.



Δυαδικά δέντρα αναζήτησης: παράδειγμα -3-



- Έστω τώρα ότι αναζητούμε τον κόμβο με τιμή 43.
- Εξετάζοντας αυτήν την τιμή με την τιμή της ρίζας, οδηγούμαστε στο δεξί υποδέντρο με ρίζα τον κόμβο 52:
- Ο αριθμός 43 είναι μικρότερος από το 52, επομένως ο αντίστοιχος κόμβος πρέπει να βρίσκεται στο αριστερό υποδέντρο του παραπάνω υποδέντρου.
- Επειδή όμως δεν υπάρχει αριστερό υποδέντρο, συμπεραίνουμε ότι ο αριθμός 43 δεν υπάρχει στο δέντρο.



Αλγόριθμος αναζήτησης σε ΔΔΑ -1-

- Ο δείκτης LocPtr δείχνει αρχικά στην ρίζα του ΔΔΑ και αντικαθίσταται κάθε φορά από τον αριστερό ή δεξιό δεσμό του τρέχοντος κόμβου, ανάλογα με το αν το στοιχείο που αναζητούμε είναι μικρότερο ή μεγαλύτερο από την τιμή του κόμβου αυτού.
- Η διαδικασία συνεχίζεται μέχρι να βρεθεί το ζητούμενο στοιχείο ή μέχρι ο δείκτης LocPtr να πάρει τιμή NULL, υποδηλώνοντας ένα κενό υποδέντρο, οπότε το στοιχείο δεν βρίσκεται στο δέντρο.

Αλγόριθμος αναζήτησης σε ΔΔΑ -2-

- Θεωρούμε ότι το τμήμα δεδομένων (Data) κάθε κόμβου του ΔΔΑ είναι εγγραφή που περιέχει ένα πεδίο κλειδί.
- Η τιμή που αναζητούμε συγκρίνεται με τις τιμές των πεδίων κλειδιών των κόμβων και το αποτέλεσμα της σύγκρισης χρησιμοποιείται για να καθοριστεί αν η αναζήτηση θα συνεχιστεί με το αριστερό ή δεξί υποδέντρο ή αν θα τερματιστεί, επειδή βρέθηκε το στοιχείο. .

Συνάρτηση αναζήτησης σε ΔΔΑ -1-

void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found, BinTreePointer *LocPtr)

/*Δέχεται: Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.

Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.

Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.

Διαφορετικά η Found έχει τιμή FALSE.*/

Συνάρτηση αναζήτησης σε ΔΔΑ -2-

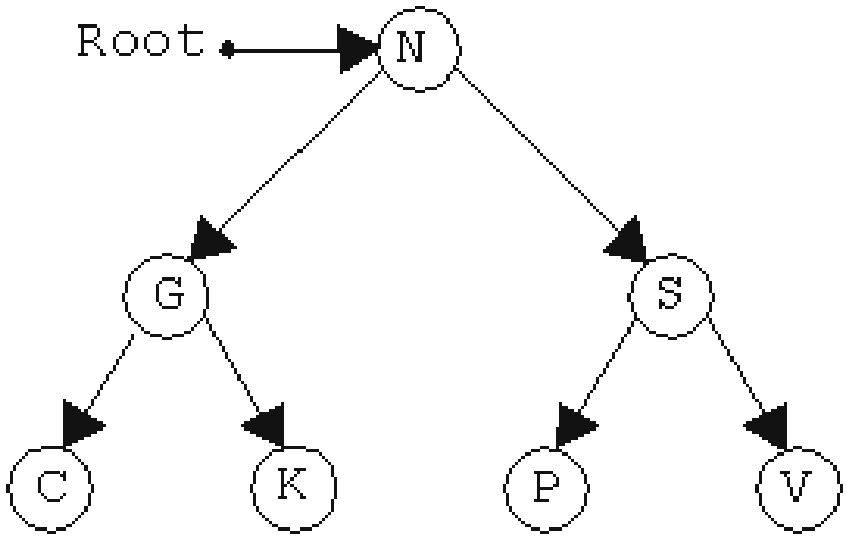
```
{
    *LocPtr = Root; *Found = FALSE;
    while (!(*Found) && (*LocPtr) != NULL)
            if (KeyValue < LocPtr->Data)
                    (*LocPtr) = LocPtr->LChild;
            else
               if (KeyValue > LocPtr->Data)
                    (*LocPtr) = LocPtr->RChild
               else
                    (*Found) = TRUE
}
```

Κατασκευή ΔΔΑ -1-

- Ένα δυαδικό δέντρο αναζήτησης μπορεί να κατασκευαστεί με επαναλαμβανόμενες κλήσεις μιας διαδικασίας εισαγωγής στοιχείων σε ένα ΔΔΑ που αρχικά είναι κενό, δηλαδή Root = NULL.
- Για να καθορίσουμε τη θέση στην οποία θα εισαχθεί ένα στοιχείο χρησιμοποιούμε παρόμοια μέθοδο με αυτήν που χρησιμοποιήσαμε για την αναζήτηση ενός ΔΔΑ.
- Η μόνη αλλαγή που χρειάζεται να κάνουμε στην διαδικασία BSTSearch είναι να διατηρούμε έναν δείκτη προς τον πατέρα του τρέχοντος κόμβου που εξετάζεται, καθώς "κατεβαίνουμε" στο δέντρο αναζητώντας μια θέση για να εισάγουμε το στοιχείο.

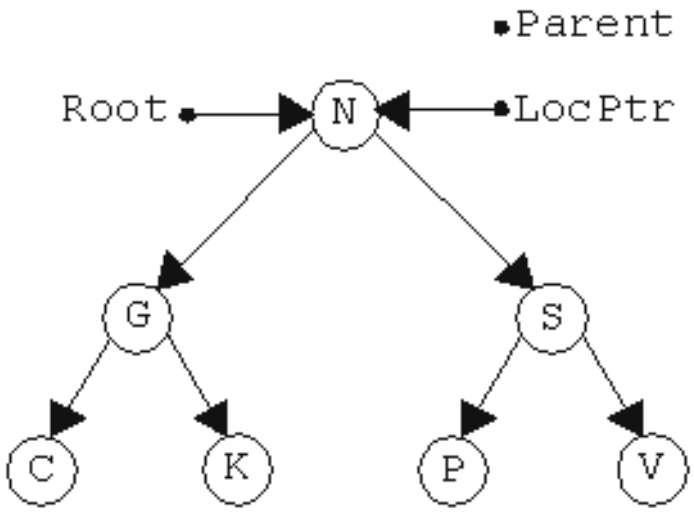
Κατασκευή ΔΔΑ -2-

- Έστω ότι έχει ήδη κατασκευαστεί το ΔΔΑ και επιθυμούμε να εισάγουμε το γράμμα Μ.
- Ξεκινάμε από την ρίζα και συγκρίνουμε το γράμμα Ν, που βρίσκεται στη ρίζα, με το Μ, που αναζητούμε.



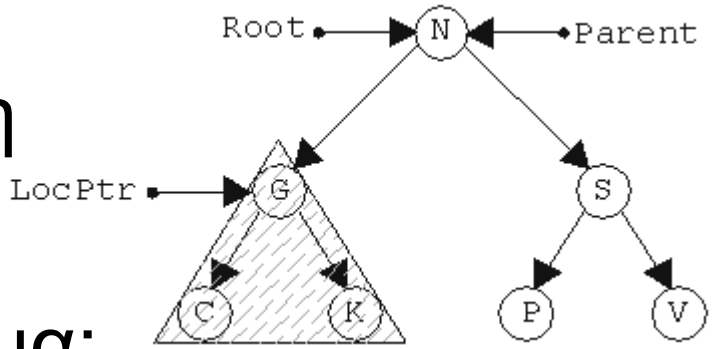
Κατασκευή ΔΔΑ -3-

- Οι δείκτες Root, Parent και LocPtr δείχνουν όπως φαίνεται στο ακόλουθο σχήμα:
- ο δείκτης Root θα δείχνει πάντα τη ρίζα του δέντρου,
- ο δείκτης LocPtr θα δείχνει τον τρέχοντα κόμβο και
- ο δείκτης Parent θα δείχνει τον πατέρα του τρέχοντος κόμβου.

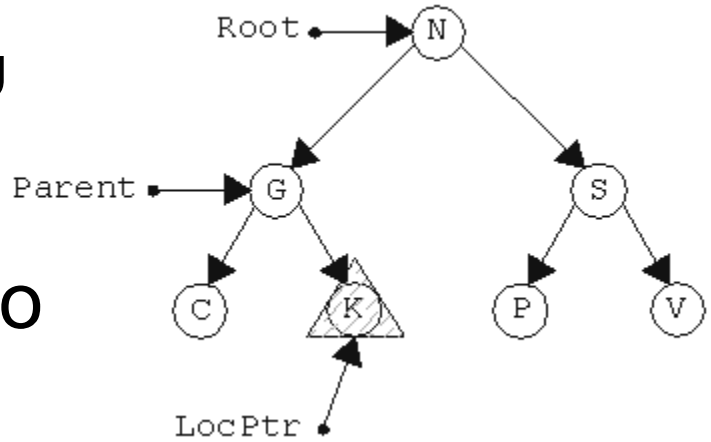


Κατασκευή ΔΔΑ -4-

- Επειδή ισχύει M<N, συνεχίζουμε την αναζήτηση στο αριστερό υποδέντρο, που έχει ρίζα το γράμμα G, όπως φαίνεται και στο σχήμα:

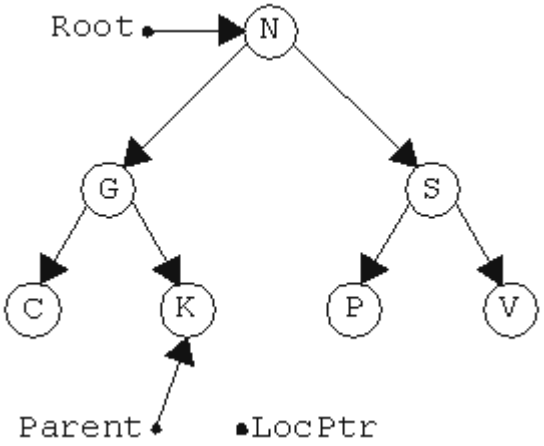


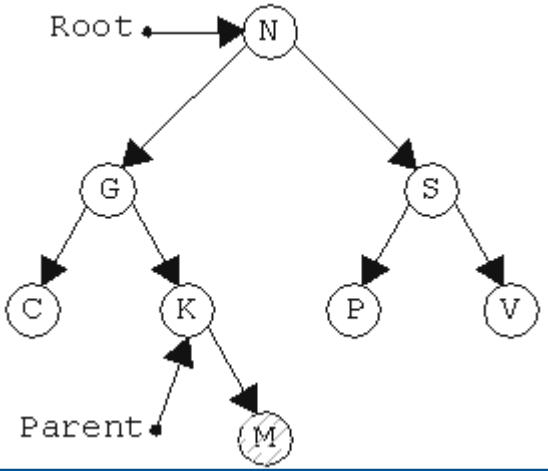
- Τώρα συγκρίνουμε το M με το περιεχόμενο της ρίζας του παραπάνω υποδέντρου και, επειδή M>G, κατεβαίνουμε στο δεξί υποδέντρο με ρίζα το γράμμα K:



Κατασκευή ΔΔΑ -5-

- Επειδή M>K, πρέπει να κατεβούμε στο δεξί υποδέντρο του παραπάνω υποδέντρου, που είναι κενό:
- Επειδή αυτό το δεξί υποδέντρο είναι κενό (LocPtr= NULL), συμπεραίνουμε ότι το γράμμα M δεν βρίσκεται μέσα στο ΔΔΑ και ότι θα πρέπει να εισαχθεί ως δεξί παιδί του κόμβου πατέρα του, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης Parent:





Διαδικασία εισαγωγής στοιχείου σε ΔΔΑ -1-

void BSTInsert(BinTreePointer *Root,
BinTreeElementType Item)

/*Δέχεται: Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.

Λειτουργία: Εισάγει το στοιχείο Item στο ΔΔΑ.

Επιστρέφει:Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του.*/

Διαδικασία εισαγωγής στοιχείου σε ΔΔΑ -2-

```
{
  BinTreePointer LocPtr      /*δείκτης αναζήτησης
       Parent;               δείκτης προς το πατέρα του
                             τρέχοντος κόμβου*/
  boolean Found;             /* δείχνει αν το στοιχείο
                             Item Υπάρχει ήδη στο ΔΔΑ

  LocPtr = *Root; Parent = NULL;
  Found = FALSE;
```

Διαδικασία εισαγωγής στοιχείου σε ΔΔΑ -3-

```
while (!Found) && (LocPtr!= NULL)
    {
        Parent = LocPtr;
        if (Item < LocPtr->Data)
            LocPtr = LocPtr->LChild;
        else
            if (Item > LocPtr->Data)
                LocPtr = LocPtr->RChild;
            else
                Found = TRUE;
    }
```

Διαδικασία εισαγωγής στοιχείου σε ΔΔΑ -4-

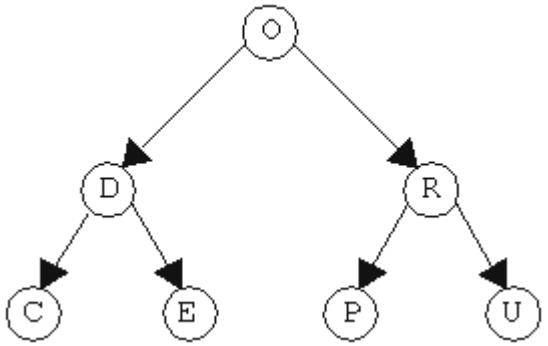
```
if (Found)
        printf('Το στοιχείο υπάρχει ήδη στο
              δέντρο\n');
    else
    {
         LocPtr = (BinTreePointer)malloc(sizeof
              (struct BinTreeNode));
       LocPtr ->Data = Item;
       LocPtr ->LChild = NULL;
       LocPtr ->RChild = NULL;
```

Διαδικασία εισαγωγής στοιχείου σε ΔΔΑ -5-

```
if (Parent == NULL)    /*κενό δέντρο*/
        (*Root) = LocPtr;
   else
          if (Item < Parent ->Data)
              Parent ->LChild = LocPtr;
          else
               Parent -> RChild = LocPtr;
   }
   }
```

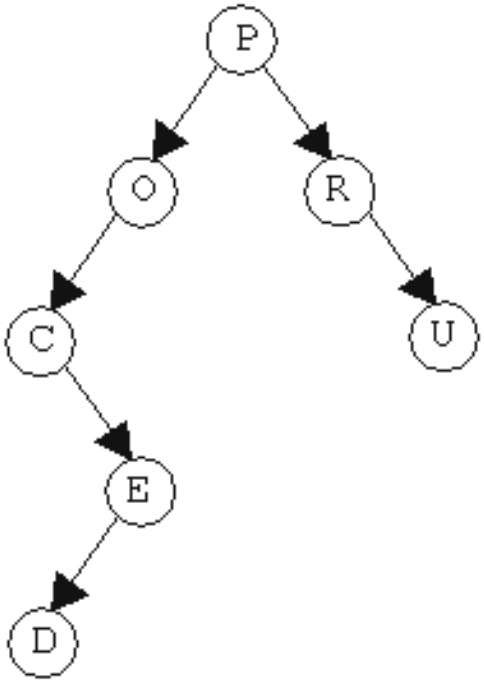
Εισαγωγή στοιχείων σε ΔΔΑ: παράδειγμα -1-

- Η σειρά με την οποία εισάγονται τα στοιχεία σε ένα ΔΔΑ καθορίζει και τη δομή του δέντρου.
- Αν για παράδειγμα θέλουμε να εισαγάγουμε τα γράμματα O, R, D, E, P, U, C, E, R με αυτήν την σειρά σε ένα ΔΔΑ, θα προκύψει το παρακάτω ισοζυγισμένο δέντρο (balanced tree):



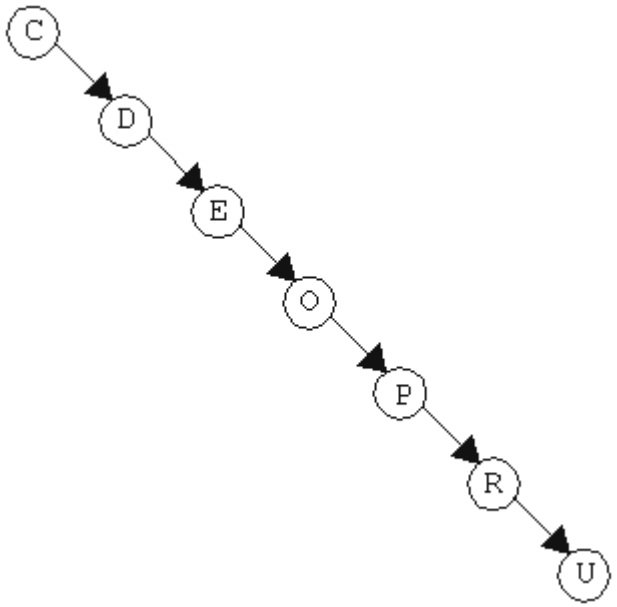
Εισαγωγή στοιχείων σε ΔΔΑ: παράδειγμα -2-

- ενώ, αν η εισαγωγή γίνει με τη σειρά P, R, O, C, E, D, U, R, E, τότε προκύπτει το παρακάτω μη ισοζυγισμένο δέντρο



Εισαγωγή στοιχείων σε ΔΔΑ: παράδειγμα -3-

- Τέλος, αν τα εισάγουμε με αλφαβητική σειρά, δηλαδή C, D, E, E, O, P, R, R, U, τότε το δέντρο εκφυλίζεται σε συνδεδεμένη λίστα:



Διαγραφή κόμβου από ΔΔΑ -1-

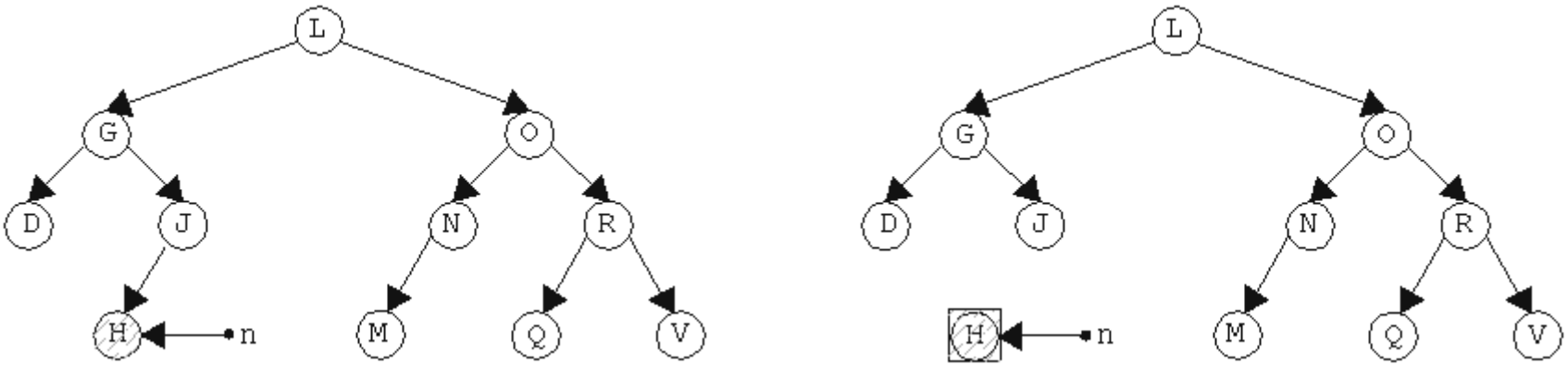
- Για την διαγραφή ενός κόμβου n από ένα ΔΔΑ υπάρχουν τρεις περιπτώσεις:
- Ο n είναι φύλλο
- Ο n έχει ένα παιδί
- Ο n έχει δύο παιδιά

Διαγραφή κόμβου από ΔΔΑ -2-

- Όταν ο κόμβος που θέλουμε να διαγράψουμε είναι φύλλο, τότε απλά θέτουμε τον αριστερό ή δεξιό δείκτη του πατέρα του n, ανάλογα με το αν ο n είναι το αριστερό ή το δεξί παιδί του πατέρα του, ίσο με NULL.

Διαγραφή κόμβου από ΔΔΑ -3-

- Αν, για παράδειγμα, θέλουμε να διαγράψουμε τον κόμβο Η από το παρακάτω δέντρο.
- τότε θέτουμε απλά τον αριστερό δείκτη του κόμβου J ίσο με NULL και πετάμε τον κόμβο Η, όπως φαίνεται στο σχήμα που ακολουθεί:

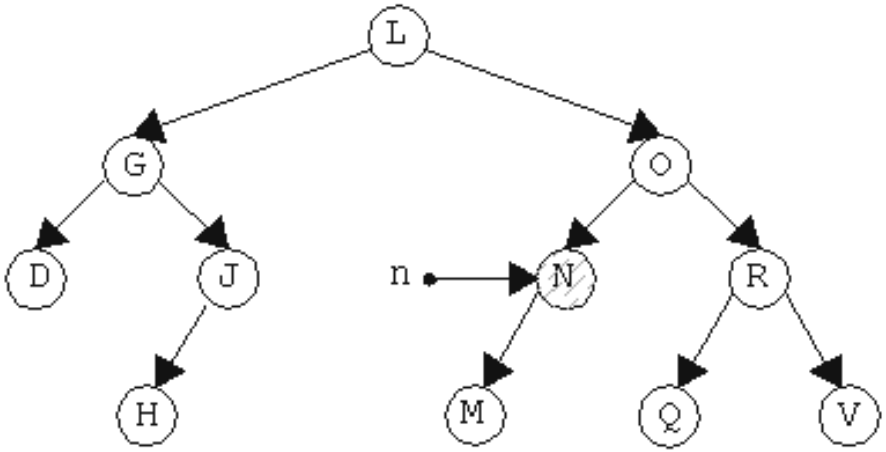


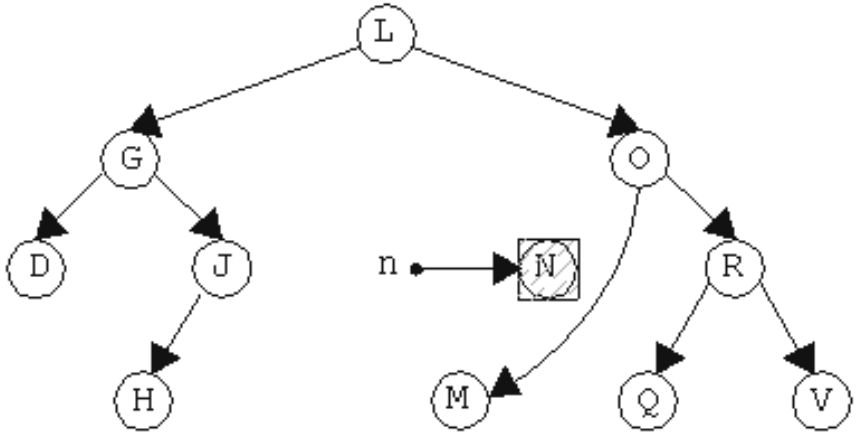
Διαγραφή κόμβου από ΔΔΑ -4-

- Απλή είναι, επίσης, και η δεύτερη περίπτωση, όπου ο κόμβος n έχει ένα παιδί.
- Το μόνο που χρειάζεται να κάνουμε σ' αυτήν την περίπτωση είναι να θέσουμε τον δείκτη του πατέρα τού n να δείχνει στο παιδί τού n.

Διαγραφή κόμβου από ΔΔΑ -5-

- Αν δηλαδή θέλουμε να διαγράψουμε τον κόμβο Ν του ίδιου ΔΔΑ:
- τότε αρκεί να θέσουμε τον αριστερό δείκτη του κόμβου Ο να δείχνει στον κόμβο Μ:





Διαγραφή κόμβου από ΔΔΑ -6-

- Στην πραγματικότητα, οι δυο παραπάνω περιπτώσεις μπορούν να συνδυαστούν σε μία, κατά την οποία ο προς διαγραφή κόμβος n έχει το πολύ ένα μη κενό υποδέντρο:
- Αν ο αριστερός δείκτης του n είναι NULL, τότε θέτουμε τον κατάλληλο δείκτη του πατέρα του n να δείχνει στο δεξί υποδέντρο του n (το οποίο, αν είναι κενό, σημαίνει ότι ο n είναι φύλλο).
- Διαφορετικά τον θέτουμε να δείχνει στο αριστερό υποδέντρο του n.

Διαγραφή κόμβου από ΔΔΑ -7-

```
        SubTree = n->LChild;
if (SubTree == NULL)
        SubTree = n->RChild;
if (Parent == NULL)    *Root = SubTree;
else if (Parent->LChild == n)
         Parent->LChild = SubTree;
     else
          Parent->RChild = SubTree;
free(n);
```

Διαγραφή κόμβου από ΔΔΑ -8-

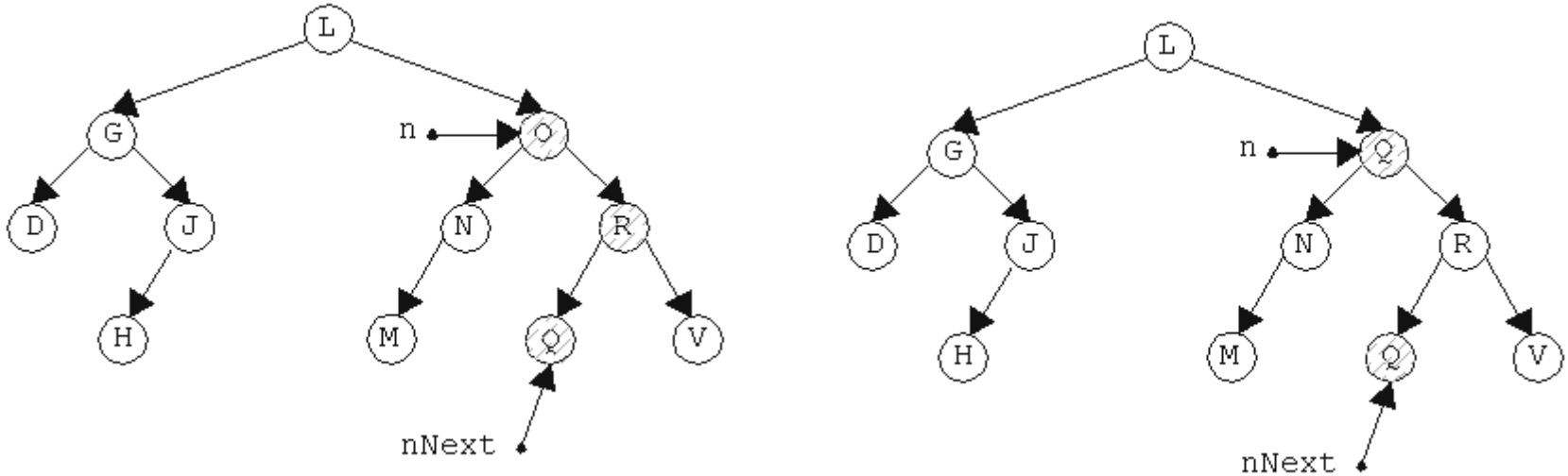
- Η περίπτωση διαγραφής ενός κόμβου με δύο παιδιά, μπορεί να αναχθεί σε μια από τις δύο προηγούμενες, αν αντικαταστήσουμε το περιεχόμενο του κόμβου n με το αντίστοιχο του ενδοδιατεταγμένου επόμενού του και στη συνέχεια διαγράψουμε αυτόν τον επόμενο.
- Ο ενδοδιατεταγμένος επόμενος ενός κόμβου ενός ΔΔΑ είναι ο επόμενος σε μια ενδοδιατεταγμένη διάσχιση του ΔΔΑ.

Διαγραφή κόμβου από ΔΔΑ -9-

- Για παράδειγμα, αν πρόκειται να διαγράψουμε τον κόμβο Ο στο ΔΔΑ:
- μπορούμε να εντοπίσουμε τον ενδοδιατεταγμένο επόμενό του ξεκινώντας από το δεξί παιδί του κόμβου Ο και κατεβαίνοντας αριστερά μέχρι να συναντήσουμε κόμβο χωρίς αριστερό υποδέντρο.

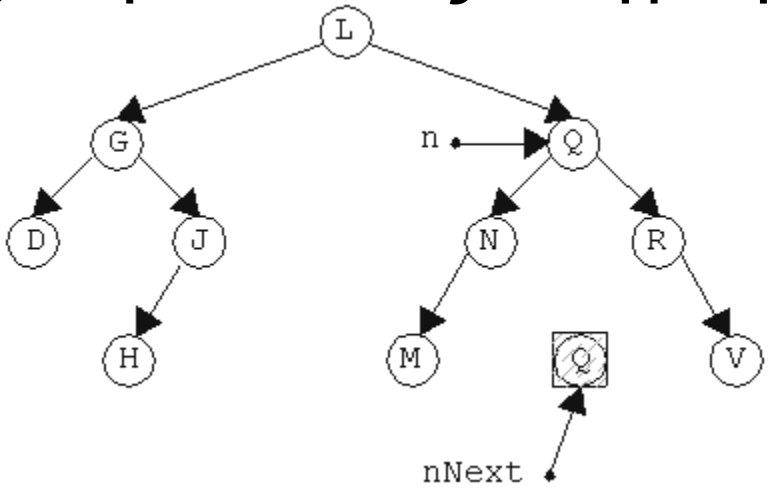
Διαγραφή κόμβου από ΔΔΑ -10-

- Για τον κόμβο O ο ενδοδιατεταγμένος επόμενος είναι ο κόμβος Q:
- Αν αντικαταστήσουμε τα περιεχόμενα του κόμβου O με αυτά του κόμβου Q, όπως δείχνει το παρακάτω σχήμα:



Διαγραφή κόμβου από ΔΔΑ -11-

- το μόνο που μένει είναι να διαγράψουμε τον κόμβο στον οποίο δείχνει ο δείκτης nNext.
- Ο κόμβος αυτός θα έχει το πολύ ένα παιδί, επομένως η διαγραφή του θα γίνει με έναν από τους τρόπους που περιγράψαμε για τις δυο πρώτες περιπτώσεις διαγραφής:



Διαδικασία διαγραφής -1-

void BSTDelete(BinTreePointer *Root,
BinTreeElementType KeyValue)

/*Δέχεται: Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.

Λειτουργία: Προσπαθεί να βρει έναν κόμβο στο ΔΔΑ που να περιέχει την τιμή KeyValue στο πεδίο κλειδί του τμήματος δεδομένων του και, αν τον βρει, τον διαγράφει από το ΔΔΑ.

Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του.*/

Διαδικασία διαγραφής -2-

```
{
  BinTreePointer n,
   /*δείχνει το κόμβο που έχει τη τιμή keyValue
   Parent,
   /*δείχνει το κόμβο πατέρα του n ή nNext
   nNext,
   /*ενδοδιατεταγμένος επομενος του n
   Subtree;
   /*δείκτης προς το υποδένδρο με ρίζα το n
  boolean Found;
```

Διαδικασία διαγραφής -3-

```
 BSTSearch2(*Root, KeyValue, &Found, n, Parent);
if (! Found) printf('Το στοιχείο δεν βρίσκεται στο ΔΔΑ')
else
{
        if (n->LChild != NULL) && (n->RChild != NULL)  {
                nNext =n->RChild; Parent =n;
                while (nNext->LChild != NULL)  {
                        Parent = nNext; nNext = nNext->LChild
                }
                n->Data = nNext->Data; n = nNext
        }
```

Διαδικασία διαγραφής -4-

```
        SubTree = n->LChild;
if (SubTree == NULL)
        SubTree = n->RChild;
if (Parent == NULL)    *Root = SubTree;
else if (Parent->LChild == n)
         Parent->LChild = SubTree;
     else
          Parent->RChild = SubTree;
free(n);
```

Διαδικασία αναζήτησης σε ΔΔΑ (2)

void BSTSearch2(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found, BinTreePointer *LocPtr, BinTreePointer *Parent)

/*Δέχεται: Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.

Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του και τον πατέρα του κόμβου αυτού.

Επιστρέφει: Η Found έχει τιμή TRUE, ο δείκτης LocPtr δείχνει στον κόμβο που περιέχει την τιμή KeyValue και ο Parent δείχνει στον πατέρα αυτού του κόμβου, αν η αναζήτηση είναι επιτυχής.

Διαφορετικά η Found έχει τιμή FALSE.*/

Διαδικασία αναζήτησης σε ΔΔΑ (2)

```
{
   *LocPtr = Root; *Parent=NULL; *Found = FALSE;
   while (!(*Found) && *LocPtr != NULL) {
      if (KeyValue < (*LocPtr)->Data) {
         *Parent=*LocPtr; *LocPtr = (*LocPtr)->LChild; }
      else if (KeyValue > (*LocPtr)->Data) {
                *Parent=*LocPtr;
                *LocPtr = (*LocPtr)->RChild; }
         else *Found = TRUE;
   }
}
```

Εφαρμογή

- Ως παράδειγμα της χρήσης του ΑΤΔ Δυαδικό Δέντρο Αναζήτησης, θεωρούμε το πρόβλημα της οργάνωσης μιας συλλογής από κωδικούς χρηστών (user-ids) και συνθηματικά (passwords).
- Κάθε φορά που ένας χρήστης μπαίνει στο σύστημα εισάγοντας έναν κωδικό χρήστη και ένα συνθηματικό, πρέπει να γίνει έλεγχος των στοιχείων αυτών από το σύστημα για να επιβεβαιωθεί ότι είναι νόμιμος χρήστης.
- Επειδή αυτή η επιβεβαίωση χρήστη θα πρέπει να γίνεται πολλές φορές κάθε μέρα, χρειάζεται οι πληροφορίες αυτές να είναι δομημένες με τέτοιο τρόπο ώστε να μπορούν να ερευνηθούν εύκολα. Παράλληλα, πρέπει να είναι δυναμική δομή, επειδή νέοι χρήστες θα προστίθενται στο σύστημα. Επομένως, μια κατάλληλη δομή είναι ένα ΔΔΑ.

Τέλος Ενότητας

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης