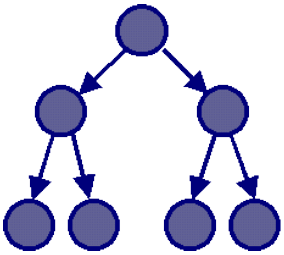
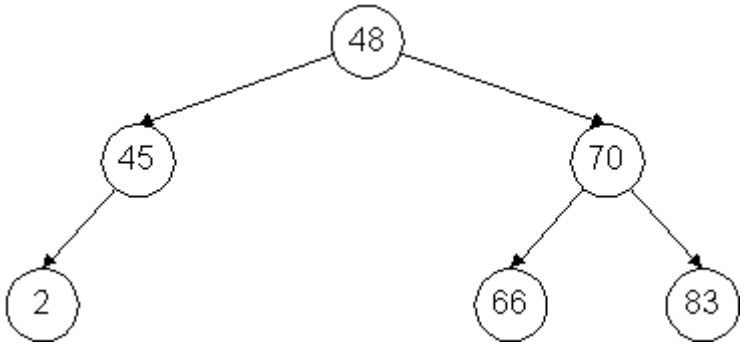


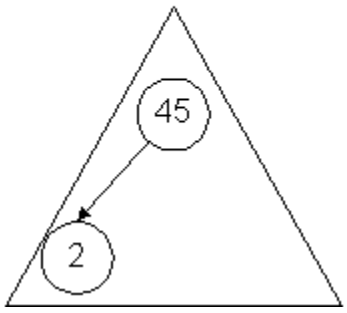
5.4 Τα ΔΔΑ ως Αναδρομικές Δομές Δεδομένων

Ένα δυαδικό δέντρο μπορεί πολύ φυσικά να οριστεί ως μια **αναδρομική δομή δεδομένων**(**recursive data structure**). Για παράδειγμα, έστω ότι έχουμε το παρακάτω δυαδικό δέντρο:

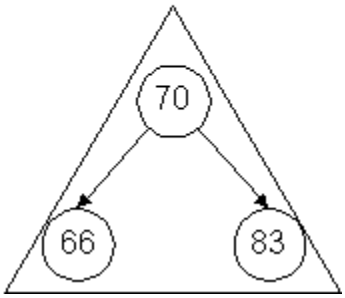


Στη ρίζα του δέντρου βρίσκεται ο αριθμός 48 και υπάρχουν δείκτες προς τους κόμβους με αριθμούς 45 και 70, καθένας από τους οποίους είναι ρίζα ενός δυαδικού υποδέντρου, όπως δείχνουν και τα ακόλουθα σχήματα:

Αριστερό Υποδένδρο



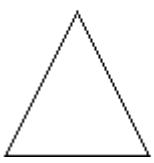
Δεξί υποδένδρο



Ας πάρουμε τώρα το αριστερό υποδέντρο, που έχει ρίζα τον κόμβο με τον αριθμό 45 και ένα αριστερό παιδί, αλλά δεν έχει δεξί. Μπορούμε να θεωρήσουμε ότι αυτός ο κόμβος έχει δείκτες προς δυο δυαδικά υποδέντρα, ένα δεξί υποδέντρο και ένα αριστερό υποδέντρο (κενό), με την προϋπόθεση ότι επιτρέπονται τα κενά δέντρα:

Αριστερό Υποδένδρο

Δεξί υποδένδρο



Επειδή το αριστερό υποδέντρο αποτελείται μόνο από έναν κόμβο, τον κόμβο με τον αριθμό 2, θα έχει κενό δεξί υποδέντρο και κενό αριστερό υποδέντρο. Έτσι, λοιπόν, ένα δυαδικό δέντρο μπορεί να οριστεί αναδρομικά ως εξής:

ΑΝΑΔΡΟΜΙΚΟΣ ΟΡΙΣΜΟΣ ΕΝΟΣ ΔΥΑΔΙΚΟΥ ΔΕΝΤΡΟΥ

Ένα δυαδικό δέντρο είτε

α. είναι κενό

είτε

β. αποτελείται από έναν κόμβο, που ονομάζεται ρίζα, ο οποίος έχει δείκτες προς δύο δυαδικά υποδέντρα, που ονομάζονται αριστερό υποδέντρο και δεξί υποδέντρο.

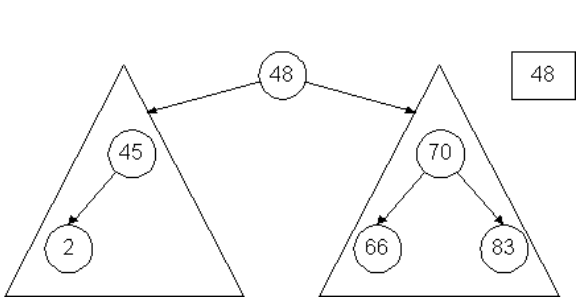
Εξαιτίας της αναδρομικής φύσης των δυαδικών δέντρων, πολλές από τις βασικές λειτουργίες τους μπορούν να εκτελεστούν πολύ απλά χρησιμοποιώντας αναδρομικούς αλγόριθμους.

Έστω, για παράδειγμα, η λειτουργία της διάσχισης, κατά την οποία μετακινούμαστε μέσα στο δυαδικό δέντρο και επισκεπτόμαστε κάθε κόμβο ακριβώς μια φορά. Υποθέτουμε ότι η σειρά με την οποία επισκεπτόμαστε κάθε κόμβο δεν έχει σημασία, αλλά είναι σημαντικό να επισκεφτούμε όλους τους κόμβους και να επεξεργαστούμε τις πληροφορίες που περιέχονται σ' αυτούς ακριβώς μια φορά.

Ένα απλό αναδρομικό σχήμα είναι να διασχίσουμε το δυαδικό δέντρο ως εξής:

1. Επίσκεψη της ρίζας και επεξεργασία των περιεχομένων της.
2. Διάσχιση του αριστερού υποδέντρου.
3. Διάσχιση του δεξιού υποδέντρου.

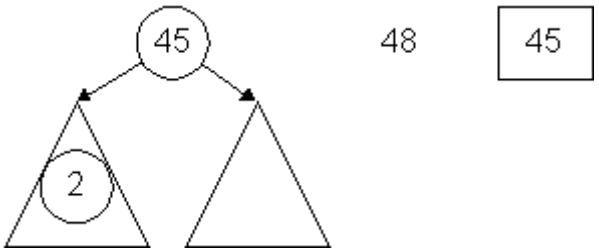
Έτσι, λοιπόν, για να διασχίσουμε το παραπάνω δέντρο, εμφανίζοντας τα περιεχόμενα κάθε κόμβου που επισκεπτόμαστε, ξεκινάμε από τη ρίζα και εμφανίζουμε τον αριθμό 48, όπως δείχνει και το σχήμα που ακολουθεί. Εν συνεχεία πρέπει να διασχίσουμε πρώτα το αριστερό υποδέντρο και έπειτα το δεξί. Όταν ολοκληρωθεί η διάσχιση και των δύο υποδέντρων θα έχουμε διασχίσει όλο το δυαδικό δέντρο.



Σχήμα Α

Τώρα το πρόβλημα έχει περιοριστεί στο να διασχίσουμε δύο μικρότερα δυαδικά δέντρα. Παίρνουμε πρώτα το αριστερό υποδέντρο (Σχήμα Β) και επισκεπτόμαστε τη ρίζα του, οπότε εμφανίζεται ο αριθμός 45. Στη συνέχεια θα διασχίσουμε πρώτα το αριστερό υποδέντρο του και έπειτα το δεξί.

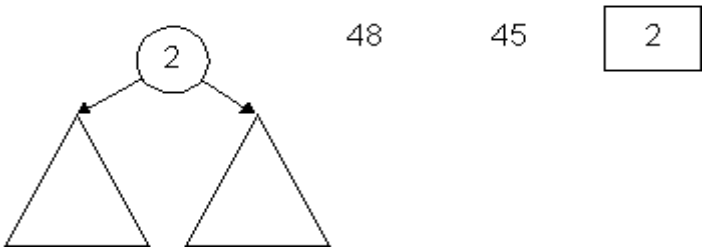
Έξοδος



Σχήμα Β

Για να διασχίσουμε το αριστερό υποδέντρο, επισκεπτόμαστε τη ρίζα του και εμφανίζουμε τον αριθμό 2. Τώρα πρέπει να διασχίσουμε πρώτα το αριστερό και στη συνέχεια το δεξί υποδέντρο.

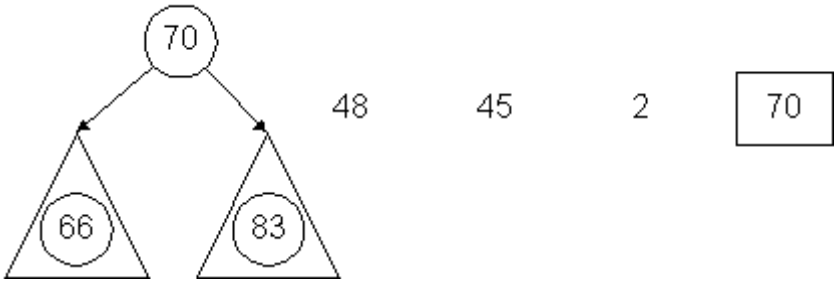
Έξοδος



Σχήμα Γ

Όπως φαίνεται και από το παραπάνω σχήμα, και τα δύο υποδέντρα είναι κενά οπότε δεν χρειάζεται να κάνουμε τίποτα, δηλαδή η διάσχιση του δέντρου του Σχήματος Γ έχει ολοκληρωθεί. Τώρα πρέπει να διασχίσουμε το δεξί υποδέντρο του δέντρου του Σχήματος Β. Το υποδέντρο αυτό είναι κενό, οπότε και πάλι δεν κάνουμε τίποτα. Επομένως, η διάσχιση του αριστερού υποδέντρου της ρίζας του αρχικού δέντρου έχει ολοκληρωθεί και μένει να διασχίσουμε το δεξί υποδέντρο. Ξεκινάμε πάλι από τη ρίζα αυτού του υποδέντρου και εμφανίζουμε τον αριθμό 70, που είναι αποθηκευμένος εκεί, όπως φαίνεται και στο Σχήμα Δ. Στη συνέχεια θα διασχίσουμε το αριστερό και το δεξί υποδέντρο.

Έξοδος



Σχήμα Δ

Πρώτα θα διασχίσουμε το αριστερό υποδέντρο του παραπάνω σχήματος, οπότε εμφανίζουμε τον αριθμό 66, που βρίσκεται στη ρίζα του. Ακολουθεί η διάσχιση του αριστερού υποδέντρου και του δεξιού υποδέντρου. Όπως φαίνεται και από το Σχήμα Ε, τα υποδέντρα αυτά είναι κενά, πράγμα που σημαίνει ότι δεν έχουμε να κάνουμε τίποτα και η διάσχιση του δέντρου του Σχήματος Ε έχει ολοκληρωθεί.

Έξοδος

Σχήμα Ε

Τώρα μας μένει να διασχίσουμε το δεξί υποδέντρο του Σχήματος Δ. Εμφανίζουμε τον αριθμό 83 της ρίζας του και στη συνέχεια πρέπει να διασχίσουμε το αριστερό και το δεξί υποδέντρο, όπως φαίνονται και στο Σχήμα Ζ. Επειδή και τα δύο αυτά δέντρα είναι κενά, δεν κάνουμε τίποτα. Η διάσχιση του δέντρου του Σχήματος Ζ έχει ολοκληρωθεί.

Έξοδος

Σχήμα Ζ

Με τα παραπάνω, έχουμε ολοκληρώσει τη διάσχιση του δέντρου του Σχήματος Δ, που αποτελεί το δεξί υποδέντρο του αρχικού μας δέντρου κι επομένως ολοκληρώσαμε τη διάσχιση ολόκληρου του δέντρου.

Όπως φαίνεται, λοιπόν, από το παράδειγμα αυτό, η διάσχιση ενός δυαδικού δέντρου αναδρομικά απαιτεί τρία βασικά βήματα, τα οποία συμβολίζουμε ως N, L και R:

N (Node): Επίσκεψη ενός κόμβου.
L (Left): Διάσχιση του αριστερού υποδέντρου ενός κόμβου.
R (Right): Διάσχιση του δεξιού υποδέντρου ενός κόμβου.

Παρόλο που στο παράδειγμά μας εκτελέσαμε τα βήματα με αυτή τη σειρά, στην πραγματικότητα έχουμε έξι διατάξεις όσον αφορά τη σειρά εκτέλεσης των βημάτων:

- LNR
- NLR
- LRN
- NRL
- RNL
- RLN

Αν, για παράδειγμα εφαρμοστεί η διάταξη LNR, τότε ο αντίστοιχος αλγόριθμος διάταξης είναι:

Αν το δυαδικό δέντρο είναι κενό **τότε**
  Μην κάνεις τίποτα
Αλλιώς
  L:Διάσχιση του αριστερού υποδέντρου
  N:Επίσκεψη της ρίζας
  R:Διάσχιση του δεξιού υποδέντρου
Τέλος_αν

Αν εφαρμόσουμε τον παραπάνω αλγόριθμο στο δυαδικό δέντρο που χρησιμοποιήσαμε και προηγουμένως, η διάσχισή του εμφανίζει τους κόμβους με τη σειρά 2, 45, 48, 66, 70, 83.

Από τις παραπάνω διατάξεις οι πιο σημαντικές είναι οι τρεις πρώτες, όπου πρώτα διασχίζεται το αριστερό υποδέντρο και μετά το δεξί, και έχουν τις εξής συγκεκριμένες ονομασίες:

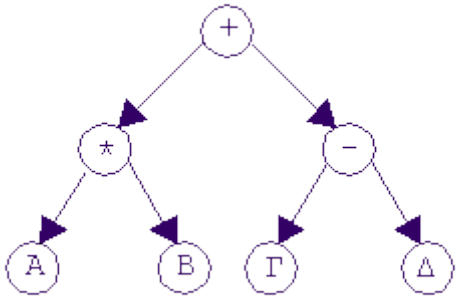
- LNR: ενδοδιατεταγμένη (inorder)
- NLR: προδιατεταγμένη (preorder)
- LRN: μεταδιατεταγμένη (postorder)

Στη συνέχεια θα δούμε γιατί αυτές οι διατάξεις είναι σημαντικές καθώς και τον λόγο που ονομάστηκαν έτσι.

Μια αριθμητική έκφραση μπορεί να παρασταθεί με ένα δυαδικό δέντρο, στο οποίο τα φύλλα περιέχουν τελεστέους και οι εσωτερικοί κόμβοι τελεστές. Ένα τέτοιο δέντρο ονομάζεται **δέντρο παράστασης** (**expression tree**). Για παράδειγμα, η αριθμητική παράσταση

Α * Β + Γ - Δ

μπορεί να παρασταθεί με το ακόλουθο δυαδικό δέντρο:



όπου κάθε τελεστέος παριστάνεται σαν ένα παιδί ενός κόμβου πατέρα, που παριστάνει τον αντίστοιχο τελεστή.

Με μια ενδοδιατεταγμένη διάσχιση του παραπάνω δέντρου παράστασης προκύπτει η ενδοθεματική έκφραση:

Α * Β + Γ - Δ

Μια προδιατεταγμένη διάσχιση του ίδιου δέντρου οδηγεί στην προθεματική έκφραση:

+ * Α Β - Γ Δ

Τέλος, μια μεταδιατεταγμένη διάσχιση του δέντρου μας δίνει την μεταθεματική έκφραση:

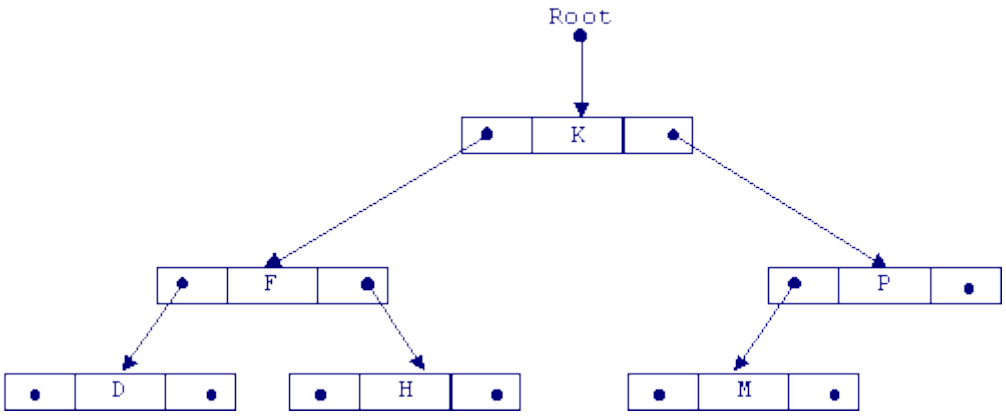
Α Β * Γ Δ - +

Είναι πολύ εύκολο να κατασκευαστεί μια αναδρομική διαδικασία για να υλοποιήσει οποιαδήποτε από τις παραπάνω διατάξεις. Για παράδειγμα, η παρακάτω διαδικασία υλοποιεί την ενδοδιατεταγμένη διάσχιση ενός δυαδικού δέντρου:

```
void RecBSTInorder(BinTreePointer Root);

/*Δέχεται:     Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.

  Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
              επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.

  Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.*/

{
  if (Root != NULL)
  {
        RecBSTInorder(Root-> LChild);                              /*L*/
        /*Προτάσεις για την επεξεργασία του στοιχείου Root->Data*/ /*N*/
        RecBSTInorder(Root-> RChild);
                                                                   /*R*/
  }
}
```

Για τα άλλα ήδη διατάξεων μπορούν να κατασκευαστούν διαδικασίες απλά αλλάζοντας τη σειρά των λειτουργιών L, N και R.

Για να δούμε στην πράξη πώς λειτουργεί η παραπάνω διαδικασία, έστω ότι έχουμε το παρακάτω δυαδικό δέντρο:



Ο πίνακας που ακολουθεί δείχνει αναλυτικά την ενέργεια της διαδικασίαςl norderTraversal για το παραπάνω δυαδικό δέντρο:

Περιεχόμενα τρέχοντος κόμβου	Ενέργεια	Έξοδος
K	Κλήση της InorderTraversal με δείκτη στη ρίζα (F) του αριστερού υποδέντρου.	
F	Κλήση της InorderTraversal με δείκτη στη ρίζα (D) του αριστερού υποδέντρου.	
D	Κλήση της InorderTraversal με δείκτη (NULL) στη ρίζα του αριστερού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
D	Εμφάνισε τα περιεχόμενα του κόμβου.	D
D	Κλήση της InorderTraversal με δείκτη στη ρίζα (NULL) του δεξιού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
D	Επιστροφή στον κόμβο πατέρα.	
F	Εμφάνισε τα περιεχόμενα του κόμβου.	F
F	Κλήση της InorderTraversal με δείκτη στη ρίζα (H) του δεξιού υποδέντρου.	
H	Κλήση της InorderTraversal με δείκτη στη ρίζα (NULL) του αριστερού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
H	Εμφάνισε τα περιεχόμενα του κόμβου.	H
H	Κλήση της InorderTraversal με δείκτη στη ρίζα (NULL) του δεξιού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
H	Επιστροφή στον κόμβο πατέρα.	
F	Επιστροφή στον κόμβο πατέρα.	
K	Εμφάνισε τα περιεχόμενα του κόμβου.	K
K	Κλήση της InorderTraversal με δείκτη στη ρίζα (P) του δεξιού υποδέντρου.	
P	Κλήση της InorderTraversal με δείκτη στη ρίζα (M) του αριστερού υποδέντρου.	
M	Κλήση της InorderTraversal με δείκτη στη ρίζα (NULL) του αριστερού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
M	Εμφάνισε τα περιεχόμενα του κόμβου.	M

M	Κλήση της InorderTraversal με δείκτη στη ρίζα (NULL) του δεξιού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
M	Επιστροφή στον κόμβο πατέρα.	
P	Εμφάνισε τα περιεχόμενα του κόμβου.	P
P	Κλήση της InorderTraversal με δείκτη στη ρίζα (NULL) του δεξιού υποδέντρου.	
τίποτα	Τίποτα, επιστροφή στον κόμβο πατέρα.	
P	Επιστροφή στον κόμβο πατέρα.	
K	Τέλος της διαδικασίας. Η διάσχιση ολοκληρώθηκε.	

Η ενδοθεματική διάσχιση που εκτελέσαμε για το παραπάνω δέντρο, εμφανίζει τα περιεχόμενα των κόμβων με αλφαβητική σειρά:

D, F, H, K, M, P

κι αυτό συμβαίνει γιατί, στην πραγματικότητα, πρόκειται όχι για ένα απλό δυαδικό δέντρο, αλλά για ένα ΔΔΑ. Έτσι, λοιπόν, για κάθε κόμβο η τιμή που είναι αποθηκευμένη στο αριστερό παιδί του είναι μικρότερη από την τιμή του κόμβου αυτού και η τελευταία είναι μικρότερη από αυτήν που είναι αποθηκευμένη στο δεξί παιδί του κόμβου. Επομένως, όλες οι τιμές που βρίσκονται σε κόμβους του αριστερού υποδέντρου ενός κόμβου είναι μικρότερες από την τιμή του κόμβου αυτού και η τιμή του κόμβου είναι με τη σειρά της μικρότερη από όλες τις τιμές των κόμβων του δεξιού υποδέντρου του. Αφού, λοιπόν, μια ενδοθεματική διάσχιση είναι μια διάσχιση με διάταξη LNR, συμπεραίνουμε ότι η επίσκεψη των κόμβων γίνεται με αλφαβητική σειρά.

Στη συνέχεια θα κατασκευάσουμε και θα υλοποιήσουμε ένα **αλγόριθμο αναζήτησης ενός ΔΔΑ αναδρομικά**. Ήδη κατασκευάσαμε ένα αλγόριθμο αναζήτησης ενός ΔΔΑ και υλοποιήσαμε την διαδικασία BSTSearch, σύμφωνα με την οποία ξεκινάμε από την ρίζα, την συγκρίνουμε με το ζητούμενο στοιχείο και, αν δεν είναι ίσο με αυτήν, τότε συνεχίζουμε την αναζήτηση στο αριστερό υποδέντρο, στην περίπτωση που η ρίζα είναι μεγαλύτερη, ή με το δεξί υποδέντρο αν είναι μικρότερη. Αν το υποδέντρο με το οποίο συνεχίζουμε είναι κενό, τότε το στοιχείο δεν υπάρχει στο δέντρο, ενώ, αν δεν είναι κενό, εκτελούμε αναζήτηση στο υποδέντρο *με τον ίδιο ακριβώς τρόπο* που κάναμε και στο αρχικό δέντρο. Αυτό δείχνει ότι στην πραγματικότητα σκεφτόμαστε αναδρομικά κι επομένως μπορούμε να χρησιμοποιήσουμε ένα αναδρομικό αλγόριθμο και την αντίστοιχη αναδρομική διαδικασία:

ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΔΡΟΜΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ ΣΕ ΕΝΑ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ

/*Δέχεται: Ένα ΔΔΑ με το δείκτη *Root* να δείχνει στη ρίζα του, την τιμή *KeyValue* που αναζητείται, τη λογική μεταβλητή *Found* που δηλώνει αν βρέθηκε ή όχι η τιμή *KeyValue* στο ΔΔΑ, και το δείκτη *LocPtr* που δείχνει στον κόμβο που περιέχει την τιμή *KeyValue*, εφόσον η αναζήτηση είναι επιτυχής.

Λειτουργία: Εκτελεί αναδρομική αναζήτηση στο ΔΔΑ για 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.*/

Αν *Root* = **NULL τότε** /*αν ο δείκτης *Root* που δείχνει στη ρίζα του ΔΔΑ έχει την τιμή NULL, δηλαδή αν το ΔΔΑ είναι κενό*/

Found ← **FALSE** /*θέσε στη μεταβλητή *Found,* η οποία δηλώνει αν βρέθηκε ή όχι η τιμή *KeyValue* στο ΔΔΑ, την τιμή FALSE*/

Αλλιώς /*δηλαδή αν το ΔΔΑ δεν είναι κενό*/

Αν *KeyValue* < Data(*Root*)**τότε** /*αν το ζητούμενο στοιχείο *Keyvalue* είναι μικρότερο από την τιμή της ρίζας *Root* του τρέχοντος υποδέντρου που ανιχνεύεται*/

Εφάρμοσε τον αλγόριθμο της αναδρομικής αναζήτησης στο αριστερό υποδέντρο /*δηλαδή στο υποδέντρο που δείχνει ο αριστερός δεσμός *LChild* του τρέχοντα κόμβου, ο οποίος δεικτοδοτείται από τον *Root**/

Αλλιώς_αν *KeyValue* > Data(*Root*) **τότε** /*αν το ζητούμενο στοιχείο *Keyvalue* είναι μεγαλύτερο από την τιμή της ρίζας *Root* του τρέχοντος υποδέντρου που ανιχνεύεται*/

Εφάρμοσε τον αλγόριθμο της αναδρομικής αναζήτησης στο δεξί υποδέντρο /*δηλαδή στο υποδέντρο που δείχνει ο δεξιός δεσμός *RChild* του τρέχοντα κόμβου, ο οποίος δεικτοδοτείται από τον *Root**/

Αλλιώς

Found ← **TRUE** /*θέσε στη μεταβλητή *Found* την τιμή TRUE εφόσον η τιμή *KeyValue* βρέθηκε στο ΔΔΑ*/

LocPtr ← *Root* /*θέσε στο δείκτη *LocPtr* την τιμή του δείκτη *Root*, δηλαδή ο *LocPtr* δείχνει στον κόμβο του ΔΔΑ που περιέχει την τιμή *KeyValue**/

Τέλος_αν

Τέλος_αν

```
void RecBSTSearch(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:     Εκτελεί αναδρομική αναζήτηση στο ΔΔΑ για 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.*/
{
   if (Root == NULL) /*κενό δέντρο*/
     *Found = FALSE;
   else                      /*μη κενό δέντρο*/
     if (KeyValue < Root->Data)
       RecBSTSearch(Root->LChild, KeyValue, &(*Found), &(*LocPtr));
     else if (KeyValue > Root->Data)
       RecBSTSearch(Root->RChild, KeyValue, &(*Found), &(*LocPtr));
     else
       {
         *Found = TRUE;
         *LocPtr = Root;
       }
}
```

Ομοίως, μπορούμε να κατασκευάσουμε και έναν **αναδρομικό αλγόριθμο για την εισαγωγή ενός στοιχείου σε ένα ΔΔΑ**:

ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΕΙΣΑΓΩΓΗΣ ΣΤΟΙΧΕΙΟΥ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ

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

Λειτουργία: Εισάγει αναδρομικά το στοιχείο *Item* στο ΔΔΑ.

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

Αν *Root* = **NULL τότε** /*αν ο δείκτης *Root* που δείχνει στη ρίζα του ΔΔΑ έχει την τιμή NULL, δηλαδή αν το ΔΔΑ είναι κενό*/

Πάρε ένα νέο κόμβο δεικτοδοτούμενο από τον *Root*

Data(*Root*) ← *Item* /*θέσε στο τμήμα δεδομένου του κόμβου που δεικτοδοτείται από τον *Root* την τιμή της *Item**/

Root->LChild ← **NULL**

Root->RChild ← **NULL** /*θέσε στο αριστερό (*LChild*) και στο δεξί (*RChild*) τμήμα δεσμού του κόμβου του δείκτη *Root* την τιμή NULL, εφόσον ο νέος κόμβος δεν έχει παιδιά*/

Αλλιώς

Αν *Item* < Data(*Root*) **τότε**

/*αν η τιμή της *Item* είναι μικρότερη από την τιμή της ρίζας *Root* του τρέχοντος υποδέντρου*/

Εφάρμοσε τον αναδρομικό αλγόριθμο της εισαγωγής στοιχείου στο αριστερό υποδέντρο

/*δηλαδή, συνέχισε να αναζητάς τη θέση που θα εισαχθεί το νέο στοιχείο *Item,* στο υποδέντρο που δείχνει ο αριστερός δεσμός *LChild* του τρέχοντα κόμβου, ο οποίος δεικτοδοτείται από τον *Root**/

Αλλιώς_αν *Item* > Data(*Root*) **τότε**

/*αν η τιμή της *Item* είναι μεγαλύτερη από την τιμή της ρίζας *Root* του τρέχοντος υποδέντρου*/

Εφάρμοσε τον αναδρομικό αλγόριθμο της εισαγωγής στοιχείου στο δεξί υποδέντρο

/*δηλαδή, συνέχισε να αναζητάς τη θέση που θα εισαχθεί το νέο στοιχείο *Item,* στο υποδέντρο που δείχνει ο δεξιός δεσμός *RChild* του τρέχοντα κόμβου, ο οποίος δεικτοδοτείται από τον *Root**/

Αλλιώς

Γράψε 'Το στοιχείο υπάρχει ήδη στο ΔΔΑ'

Τέλος_αν

Τέλος_αν

Η αντίστοιχη αναδρομική διαδικασία για την εισαγωγή ενός στοιχείου σε ένα ΔΔΑ φαίνεται παρακάτω:

```
void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item);
/*Δέχεται:        Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο
                  Item.
  Λειτουργία:     Εισάγει αναδρομικά το στοιχείο Item στο ΔΔΑ.
  Επιστρέφει:     Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του.*/
{
    if (BSTEmpty(*Root)) {
        (*Root) = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        (*Root)->Data = Item;
        (*Root)->LChild = NULL;
        (*Root)->RChild = NULL;
```

```
    }
    else
        if (Item < (*Root)-> Data)
            RecBSTInsert(&(*Root)->LChild, Item);
        else if (Item > (*Root)->Data)
            RecBSTInsert(&(*Root)->RChild, Item);
        else
            printf("Το %d στοιχείο βρίσκεται στο δέντρο\n", Item);
}
```

Έμεινε τώρα να κατασκευάσουμε έναν **αναδρομικό αλγόριθμο για τη λειτουργία της διαγραφής ενός κόμβου από ένα ΔΔΑ**. Στη συνέχεια παρουσιάζεται ο αλγόριθμος:

ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗΣ ΣΤΟΙΧΕΙΟΥ ΑΠΟ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ

/*Δέχεται: Ένα ΔΔΑ με το δείκτη *Root* να δείχνει στη ρίζα του και μια τιμή *KeyValue*. Την πρώτη φορά που εκτελείται ο αλγόριθμος ο δείκτης *Root* δείχνει στη ρίζα του αρχικού ΔΔΑ, ενώ σε κάθε επόμενη επανάληψη ο δείκτης *Root* δείχνει σε κάποιο υποδέντρο αυτού, το οποίο είναι προφανές ότι αποτελεί και αυτό ΔΔΑ.

Λειτουργία: Προσπαθεί αναδρομικά 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.*/

Αν *Root* = **NULL τότε** /*το ΔΔΑ είναι κενό*/

Γράψε 'Το στοιχείο δεν βρίσκεται στο ΔΔΑ'

Αλλιώς

/*αναζήτησε αναδρομικά τον κόμβο που περιέχει την τιμή *KeyValue* και διάγραψέ τον*/

Αν *KeyValue* < Data(*Root*) **τότε** /*αν η τιμή της *KeyValue* είναι μικρότερη από την τιμή της ρίζας *Root* του ΔΔΑ*/

Εφάρμοσε τον αναδρομικό αλγόριθμο διαγραφής στο αριστερό υποδέντρο

/*ο δείκτης *Root* θα έχει ως τιμή κατά την κλήση του αλγορίθμου τον αριστερό δεσμό της ρίζας του τρέχοντος δέντρου, δηλαδή *Root->LChild*/

Αλλιώς_αν *KeyValue* > Data(*Root*) **τότε**

/*αν η τιμή της *KeyValue* είναι μεγαλύτερη από την τιμή της ρίζας *Root* του ΔΔΑ*/

Εφάρμοσε τον αναδρομικό αλγόριθμο διαγραφής στο δεξί υποδέντρο /*ο δείκτης *Root* θα έχει ως τιμή κατά την κλήση του αλγορίθμου τον δεξιό δεσμό της ρίζας του τρέχοντος

δέντρου, δηλαδή *Root->RChild**/

Αλλιώς /*η τιμή της *KeyValue* είναι ίση με την τιμή της ρίζας *Root* του τρέχοντος υποδέντρου*/

Αν *Root->LChild* = **NULL τότε**

/*αν ο αριστερός δεσμός *LChild* της ρίζας *Root* έχει την τιμή NULL, δηλαδή αν η ρίζα δεν έχει αριστερό παιδί*/

/*ενημέρωση του δείκτη *Root* ώστε, μετά τη διαγραφή του κόμβου στον οποίο δείχνει, να δείχνει στο δεξί παιδί του */

TempPtr ← Root->RChild /*ενημέρωσε τον δείκτη *TempPtr* ώστε να δείχνει στο δεξί παιδί της ρίζας *Root* */

Να επιστρέψεις στη δεξαμενή των διαθέσιμων κόμβων τον κόμβο στον οποίο δείχνει ο *Root*

Root ← TempPtr /*θέσε στον δείκτη *Root* την τιμή του *TempPtr,* δηλαδή ο *Root* δείχνει πλέον στο δεξί παιδί του κόμβου που διαγράφηκε*/

Αλλιώς_αν *Root->RChild* = **NULL τότε**

/*αν ο δεξιός δεσμός *RChild* της ρίζας *Root* έχει την τιμή NULL, δηλαδή αν η ρίζα δεν έχει δεξί παιδί*/

/*ενημέρωση του δείκτη *Root* ώστε, μετά τη διαγραφή του κόμβου στον οποίο δείχνει, να δείχνει στο αριστερό παιδί του */

TempPtr ← Root->LChild /*ενημέρωσε τον δείκτη *TempPtr* ώστε να δείχνει στο αριστερό παιδί της ρίζας *Root* */

Να επιστρέψεις στη δεξαμενή των διαθέσιμων κόμβων τον κόμβο στον οποίο δείχνει ο *Root*

Root ← TempPtr /*θέσε στον δείκτη *Root* την τιμή του *TempPtr,* δηλαδή ο *Root* δείχνει πλέον στο αριστερό παιδί του κόμβου που διαγράφηκε*/

Αλλιώς /*ο κόμβος *Root* έχει 2 παιδιά*/

TempPtr ← Root->RChild /*θέσε στον δείκτη *TempPtr* την τιμή του δεξιού δεσμού *RChild* της ρίζας *Root*, δηλαδή ο *TempPtr* δείχνει στο δεξί παιδί της ρίζας του κόμβου *Root* που πρόκειται να διαγραφεί*/

Όσο *TempPtr->LChild* != **NULL επανάλαβε**

/*όσο το μονοπάτι των αριστερών υποδέντρων δεν έχει τελειώσει*/

TempPtr ← TempPtr->LChild /*ενημέρωσε τον δείκτη *TempPtr* ώστε να δείχνει στο αριστερό παιδί του κόμβου που έδειχνε μέχρι τώρα*/

Τέλος_Επανάληψης

Data(*Root*) ← Data(*TempPtr*) /*θέσε στο τμήμα δεδομένου του κόμβου που δείχνει ο *Root* την τιμή του τμήματος δεδομένου του *TempPtr*/

Εφάρμοσε τον αναδρομικό αλγόριθμο διαγραφής στο δεξί υποδέντρο

/*ο δείκτης *Root* θα έχει ως τιμή κατά την κλήση του αλγορίθμου τον δεξιό δεσμό της ρίζας του τρέχοντος υποδέντρου, δηλαδή *Root->RChild,* ενώ η *KeyValue* θα έχει ως τιμή την τιμή του κόμβου του δείκτη *TempPtr*/

Τέλος_αν

Τέλος_αν

Τέλος_αν

Η διαδικασία RecBSTDelete υλοποιεί τον αναδρομικό αλγόριθμο που παρουσιάστηκε παραπάνω:

```
void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue);
/*Δέχεται:          Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή
                    KeyValue.
  Λειτουργία:       Προσπαθεί αναδρομικά 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.*/
{
  BinTreePointer TempPtr;
    if (BSTEmpty(*Root))
      printf("Το στοιχείο δεν βρίσκεται στο ΔΔΑ\n")
    else
        /*αναζήτησε αναδρομικά τον κόμβο που περιέχει την τιμή KeyValue και διάγραψέ
        τον*/
        if (KeyValue < (*Root)->Data)
               RecBSTDelete(&((*Root)->LChild), KeyValue);/* αριστερό υποδένδρο */
        else
               if (KeyValue > (*Root)->Data)
                  RecBSTDelete(&((*Root)->RChild), KeyValue);/* δεξί  υποδένδρο */
               else                /* Το KeyValue βρέθηκε διαγραφή κόμβου */
                   if ((*Root)->LChild == NULL)
                   {
                       TempPtr = (*Root)->RChild; /* δεν έχει αριστερό παιδί */
                       free(Root);
                       *Root = TempPtr;
                    }
                    else if ((*Root)->RChild == NULL)
                    {
```

```
                        TempPtr = (*Root)->LChild;/* έχει αριστερό παιδί αλλά όχι δεξί */
                        free(Root);
                        *Root = TempPtr;
                    }
                        else                              /* έχει 2 παιδιά */
                        {
                            /* εύρεση inorder απογόνου του */
                             TempPtr = (*Root)->RChild;
                             while (TempPtr-> LChild != NULL)
                                  TempPtr = TempPtr->LChild;
                                /*αντιγραφή των περιεχομένων του τρέχοντος κόμβου
                                  στο κόμβο προς διαγραφή και διαγραφή του
                                  τρέχοντος κόμβου */
                            (*Root)->Data = TempPtr->Data;
                            RecBSTDelete(&((*Root)->RChild), (*Root)->Data);
                        }
}
```

Όλες οι παραπάνω συναρτήσεις περιλαμβάνονται στο BstRecADT.c, που υλοποιεί τον ΑΤΔ Δυαδικό Δέντρο Αναζήτησης με αναδρομή και μπορεί να χρησιμοποιηθεί σε ένα πρόγραμμα C με την εντολή

#include "BstRecADT.h";

Το πρόγραμμα TestBSTRecADT.c χρησιμοποιεί τη διασύνδεση BstRecADT.h.

```
// filename TestBSTRec.c

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "BstRecADT.h"

void menu(int *choice);

main()
{
   int choice, ch;
   BinTreePointer ARoot, LocPtr;
   BinTreeElementType AnItem;
   boolean Found;

   CreateBST(&ARoot);
   do
   {
      menu(&choice);
      switch(choice)
      {
```

```
        case 1:    do
                   {
                      printf("Δώσε το στοιχείο για εισαγωγή στο ΔΔΑ: ");
                      scanf("%d", &AnItem);
                      RecBSTInsert(&ARoot, AnItem);
                      printf("\n Continue Y/N: ");
                      do
                      {
                         scanf("%c", &ch);
                      } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
                    } while (toupper(ch)!='N');
                    break;
        case 2:    if (BSTEmpty(ARoot))
                       printf("\n Άδειο δένδρο\n");
                    else
                       RecBSTInorder(ARoot);
                    break;
        case 3:    if (BSTEmpty(ARoot))
                       printf("Άδειο δένδρο\n");
                    else {
                       printf("Δώσε το στοιχείο για αναζήτηση: ");
                       scanf("%d", &AnItem);
                       RecBSTSearch(ARoot, AnItem, &Found, &LocPtr);
                       if (Found==FALSE)
                           printf("Item %d not in tree \n",AnItem);
                       else
                           printf("Item %d is in tree \n",AnItem);
                   }
                   break;
        case 4:    if (BSTEmpty(ARoot))
                       printf("\n Empty tree\n");
                   else {
                       printf("Δώσε το στοιχείο για διαγραφή : ");
                       scanf("%d", &AnItem);
                       RecBSTDelete(&ARoot, AnItem);
                   }
                   break;
        case 5: if (BSTEmpty(ARoot))
                       printf("\n Άδειο δένδρο\n");
                   else
                       RecBSTPreorder(ARoot);
                   break;
        case 6: if (BSTEmpty(ARoot))
                       printf("\n Άδειο δένδρο\n");
                else
                       RecBSTPostorder(ARoot);
                break;
     }
  } while (choice!=7);
```

```
   return 0;
}

void menu(int *choice)
{
    printf("\n                  MENOY                  \n");
    printf("-----------------------------------------------\n");
    printf("1. Εισαγωγή στοιχείου στο ΔΔΑ\n");
    printf("2. Διάσχιση ΔΔΑ inorder\n");
    printf("3. Αναζήτηση στοιχείου στο ΔΔΑ\n");
    printf("4. Διαγραφή στοιχείου από το ΔΔΑ\n");
    printf("5. Διάσχιση ΔΔΑ preorder\n");
    printf("6. Διάσχιση ΔΔΑ postorder\n");
    printf("7. Τέλος\n");
    printf("\nΕπέλεξε : ");
    do
    {
        scanf("%d", choice);
    } while (*choice<1 && *choice>6);
}
```

Το πρόγραμμα TestBSTADT.c χρησιμοποιεί τη διασύνδεση BstADT.h και την αντίστοιχη υλοποίηση της BstADT.c και δείχνει τη χρήση όλων των λειτουργιών του ΔΔΑ, η υλοποίηση των οποίων γίνεται χωρίς αναδρομή όπως είδαμε στην ενότητα 5.3.