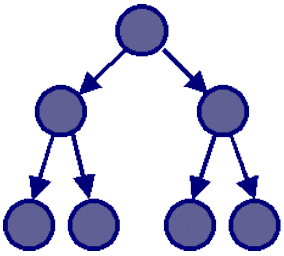
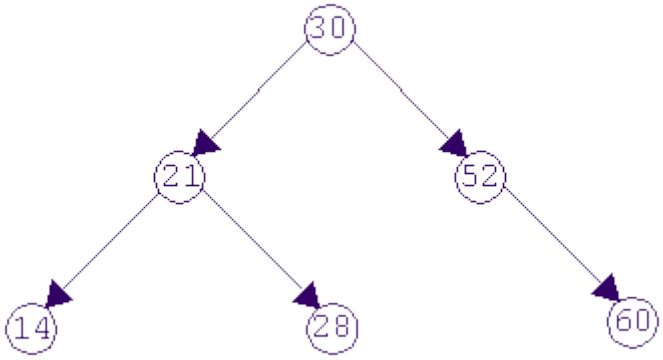


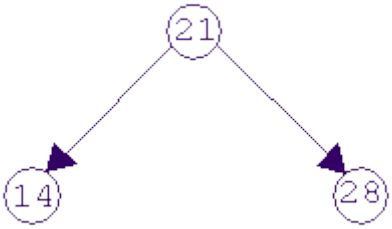
5.3 Δυαδικά Δέντρα Αναζήτησης & Υλοποίηση ΔΔΑ με δείκτες

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.

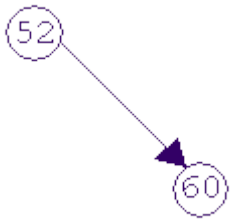
Έστω, για παράδειγμα ότι αναζητούμε τον αριθμό 14. Ξεκινώντας από την ρίζα του δυαδικού δέντρου αναζήτησης, βλέπουμε ότι ο αριθμός 14 είναι μικρότερος του 30, επομένως συμπεραίνουμε ότι ο αριθμός που αναζητούμε βρίσκεται στα αριστερά της ρίζας ή αλλιώς στο αριστερό **υποδέντρο** (**subtree**) με ρίζα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



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.

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.

Παρακάτω φαίνεται ένας **αλγόριθμος για την αναζήτηση σε ένα δυαδικό δέντρο αναζήτησης**. 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 LocPtrνα πάρει τιμή NULL, υποδηλώνοντας ένα κενό υποδέντρο, οπότε το στοιχείο δεν βρίσκεται στο δέντρο. 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.

ΑΝΑΖΗΤΗΣΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ

/*Δέχεται:	Ένα ΔΔΑ με το δείκτη *Root* να δείχνει στη ρίζα του και μια τιμή *KeyValue*.
Λειτουργία:	Αναζητά στο ΔΔΑ, με τη βοήθεια του δείκτη *LocPtr* που δείχνει στον τρέχοντα κάθε φορά κόμβο του 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.*/

1. *LocPtr* ← *Root* /*ενημέρωση του δείκτη *LocPtr* ώστε να δείχνει στη ρίζα *Root* του ΔΔΑ*/
   Found ← **FALSE**

2. **Όσο** *Found* = **FALSE και** *LocPtr* != **NULL επανάλαβε**

/*όσο δεν βρέθηκε το στοιχείο και ο δείκτης *LocPtr* έχει τιμή διάφορη του NULL, δηλαδή δεν δείχνει σε ένα κενό υποδέντρο*/

Αν *KeyValue* < Data(*LocPtr*) **τότε**

/*αν η τιμή της *KeyValue* είναι μικρότερη από την τιμή του πεδίου κλειδιού του τρέχοντος κόμβου του ΔΔΑ, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης *LocPtr*/

LocPtr ← *LocPtr ->LChild* /*θέσε στον δείκτη *LocPtr* την τιμή του αριστερού δεσμού *LChild* του τρέχοντος κόμβου *LocPtr,* προκειμένου η αναζήτηση να συνεχιστεί στο αριστερό υποδέντρο του τρέχοντος κόμβου*/

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

/*αν η τιμή της *KeyValue* είναι μεγαλύτερη από την τιμή του πεδίου κλειδιού του τρέχοντος κόμβου του ΔΔΑ, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης *LocPtr*/

LocPtr ← *LocPtr ->RChild* /*θέσε στον δείκτη *LocPtr* την τιμή του δεξιού δεσμού *RChild* του τρέχοντος κόμβου *LocPtr,* προκειμένου η αναζήτηση να συνεχιστεί στο δεξί υποδέντρο του τρέχοντος κόμβου*/

Αλλιώς

Found ← **TRUE** /*η τιμή *KeyValue* βρέθηκε στον κόμβο που δείχνει ο δείκτης *LocPtr*/

Τέλος_αν

Τέλος_επανάληψης

Πριν προχωρήσουμε στην BSTSearch που υλοποιεί την αναζήτηση σε ένα ΔΔΑ, υπενθυμίζουμε ότι η υλοποίηση ενός ΔΔΑ είναι η ίδια μ' αυτή ενός δυαδικού δένδρου όπως αυτή ορίστηκε στην ενότητα «5.2 Εισαγωγή στα Δέντρα & Δυαδικά Δέντρα». Μπορούμε να έχουμε πρόσβαση σε οποιονδήποτε κόμβο του ΔΔΑ αν διατηρούμε έναν δείκτη στην ρίζα του, οπότε με βάση αυτήν την παραδοχή υλοποιείται και η διαδικασία CreateBST που δημιουργεί ένα κενό ΔΔΑ. Στη συνέχεια δίνουμε τις απαραίτητες δηλώσεις για την υλοποίηση ενός ΔΔΑ, όπως και τη διαδικασία CreateBST.

```
typedef int BinTreeElementType;                    /*ο τύπος των στοιχείων των
                                                     κόμβων, ενδεικτικά int*/
  typedef struct BinTreeNode *BinTreePointer;
  struct BinTreeNode {
                 BinTreeElementType Data;
                 BinTreePointer LChild, RChild;
 };
```

Η διαδικασία CreateBST υλοποιεί τη δημιουργία ενός κενού ΔΔΑ.

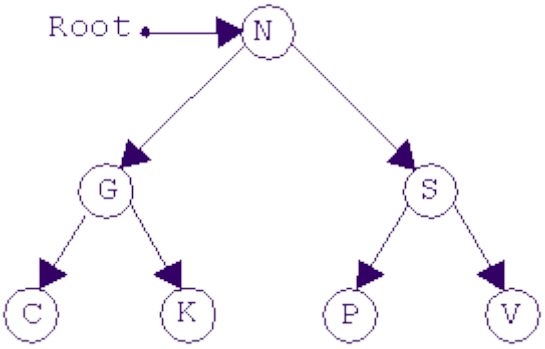
```
void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη (NULL) Root
*/
{
    *Root = NULL;
}
```

Στη συνέχεια παρατίθεται η διαδικασία BSTSearch που υλοποιεί την αναζήτηση σε ένα δυαδικό δέντρο αναζήτησης.

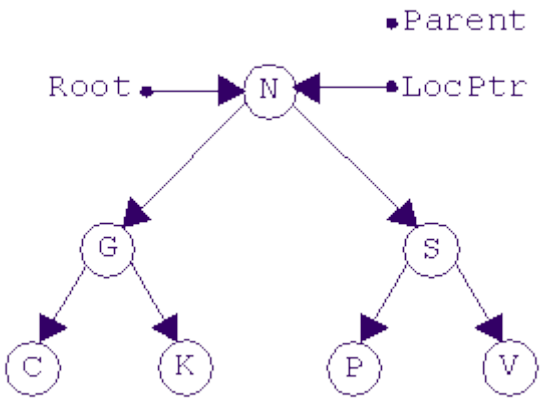
```
void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean
               *Found, BinTreePointer *LocPtr);
/*Δέχεται:        Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή
                 KeyValue.
  Λειτουργία:    Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο 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.*/
{
    (*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;
    }
}
```

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. Στην πραγματικότητα 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.

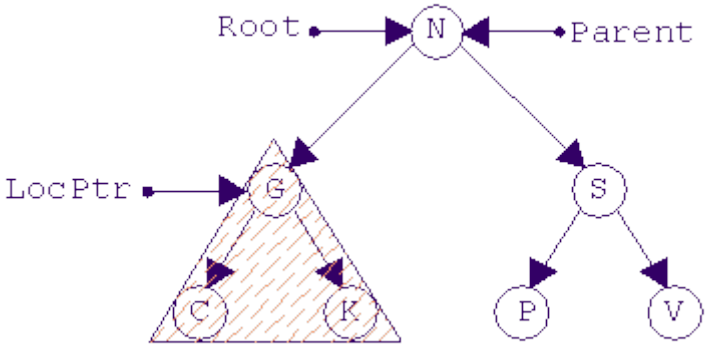
Έστω ότι έχει ήδη κατασκευαστεί το παρακάτω ΔΔΑ



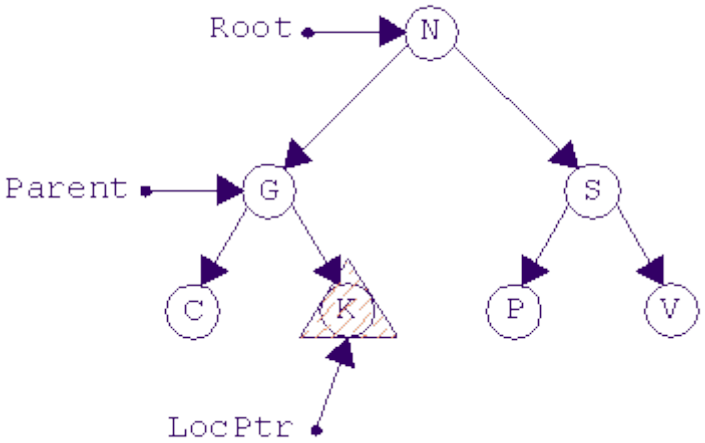
και επιθυμούμε να εισάγουμε το γράμμα M. Ξεκινάμε από την ρίζα και συγκρίνουμε το γράμμα N, που βρίσκεται στη ρίζα, με το M, που αναζητούμε. Οι δείκτες Root, Parent και LocPtr δείχνουν όπως φαίνεται στο ακόλουθο σχήμα. Ο δείκτης Root θα δείχνει πάντα τη ρίζα του δέντρου, ο δείκτης LocPtr θα δείχνει τον τρέχοντα κόμβο και ο δείκτης Parent θα δείχνει τον πατέρα του τρέχοντος κόμβου.



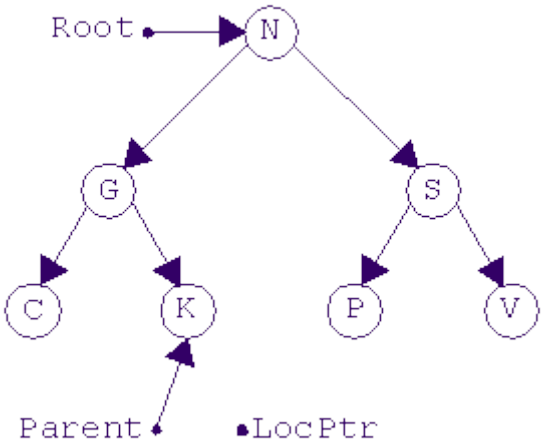
Επειδή ισχύει M<N, συνεχίζουμε την αναζήτηση στο αριστερό υποδέντρο, που έχει ρίζα το γράμμα G, όπως φαίνεται και στο σχήμα:



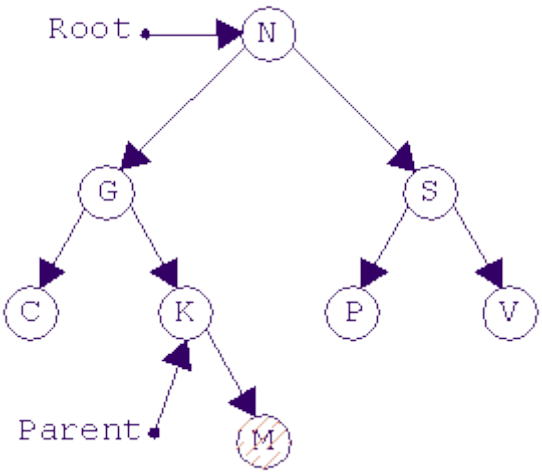
Τώρα συγκρίνουμε το M με το περιεχόμενο της ρίζας του παραπάνω υποδέντρου και, επειδή M>G, κατεβαίνουμε στο δεξί υποδέντρο με ρίζα το γράμμα K:



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:



Παρακάτω φαίνεται ένας **αλγόριθμος για την εισαγωγή ενός στοιχείου σε ένα ΔΔΑ**:

ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ

/*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.*/

1. *LocPtr* ← *Root* /*θέσε στο δείκτη *LocPtr,* ο οποίος δείχνει στον τρέχοντα κάθε φορά κόμβο του ΔΔΑ, την τιμή του δείκτη *Root* που δείχνει στη ρίζα του ΔΔΑ*/

   Parent ← **NULL** /*θέσε στο δείκτη *Parent*, ο οποίος δείχνει στον πατέρα του τρέχοντος κόμβου, την τιμή NULL*/

   Found ← **FALSE** /*θέσε στη μεταβλητή *Found*, η οποία δείχνει αν το στοιχείο *Item* υπάρχει ήδη στο ΔΔΑ, την τιμή FALSE*/

2. **Όσο** *Found* = **FALSE και** *LocPtr* != **NULL επανάλαβε**

   /*όσο δεν βρέθηκε το στοιχείο και ο δείκτης *LocPtr* έχει τιμή διάφορη του NULL, δηλαδή δεν δείχνει σε ένα κενό υποδέντρο*/

   Parent ← *LocPtr*

   Αν *Item* < Data(*LocPtr*) **τότε** /*αν η τιμή της *Item* είναι μικρότερη από την τιμή του πεδίου κλειδιού του τρέχοντος κόμβου του ΔΔΑ, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης *LocPtr**/

      LocPtr ← *LocPtr ->LChild* /*θέσε στο δείκτη *LocPtr* την τιμή του αριστερού δεσμού *LChild* του τρέχοντος κόμβου *LocPtr,* προκειμένου η αναζήτηση να συνεχιστεί στο αριστερό υποδέντρο του τρέχοντος κόμβου*/

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

/*αν η τιμή της *Item* είναι μεγαλύτερη από την τιμή του πεδίου κλειδιού του τρέχοντος κόμβου του ΔΔΑ, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης *LocPtr**/

LocPtr ← *LocPtr ->RChild* /*θέσε στον δείκτη *LocPtr* την τιμή του δεξιού δεσμού *RChild* του τρέχοντος κόμβου *LocPtr,* προκειμένου η αναζήτηση να συνεχιστεί στο δεξί υποδέντρο του τρέχοντος κόμβου*/

Αλλιώς

Found ← **TRUE** /*το στοιχείο *Item* υπάρχει ήδη στο ΔΔΑ*/

Τέλος_αν

Τέλος_επανάληψης

3. **Αν** *Found* = **TRUE τότε**

Γράψε 'Το στοιχείο υπάρχει ήδη στο δέντρο'

Αλλιώς

Πάρε ένα νέο κόμβο στον οποίο δείχνει ο δείκτης *LocPtr*

Data(*LocPtr*) ← *Item* /*θέσε στο πεδίο κλειδιού του νέου κόμβου στον οποίο δείχνει ο *LocPtr* την τιμή της *Item**/

LocPtr ->LChild ← **NULL** /*θέσε στον αριστερό δεσμό *LChild* του κόμβου που δείχνει ο *LocPtr* την τιμή NULL*/

LocPtr ->RChild ← **NULL** /*θέσε στον δεξί δεσμό *RChild* του κόμβου που δείχνει ο *LocPtr* την τιμή NULL*/

Αν *Parent* = **NULL** τότε /*αν ο δείκτη *Parent* που δείχνει σ τον πατέρα του κόμβου του δείκτη *LocPtr* έχει την τιμή NULL, δηλαδή αν το ΔΔΑ είναι κενό*/

Root ← *LocPtr* /*θέσε στο δείκτη *Root* που δείχνει στη ρίζα του ΔΔΑ την τιμή του δείκτη *LocPtr**/

Αλλιώς

Αν *Item* < Data(*Parent*) **τότε** /*αν η τιμή της *Item* είναι μικρότερη από την τιμή του πεδίου κλειδιού του κόμβου *Parent**/

Parent ->LChild ← *LocPtr* /*ενημέρωσε τον αριστερό δεσμό *LChild* του κόμβου που δείχνει ο *Parent* ώστε να δείχνει στο νέο κόμβο *LocPtr**/

Αλλιώς

Parent ->RChild ← *LocPtr* /*ενημέρωσε τον δεξί δεσμό *RChild* του κόμβου που δείχνει ο *Parent* ώστε να δείχνει στο νέο κόμβο *LocPtr**/

Τέλος_αν

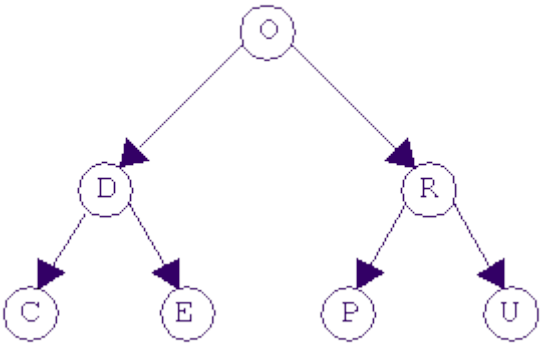
Τέλος_αν

Τέλος_αν

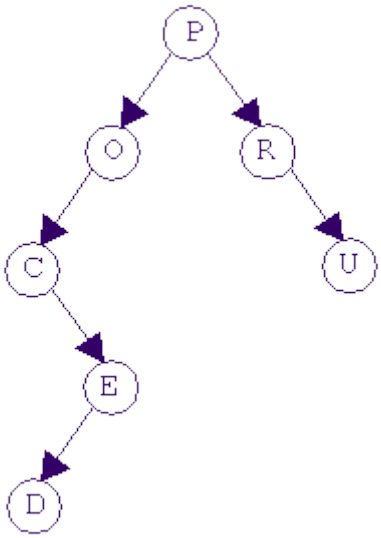
Παρακάτω φαίνεται η διαδικασία BSTInsert για την εισαγωγή ενός στοιχείου σε ένα ΔΔΑ:

```
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.*/
{
  BinTreePointer LocPtr,          /*δείκτης αναζήτησης*/
                 Parent;          /*δείκτης προς τον πατέρα του τρέχοντος
                                  κόμβου*/
  boolean Found;                  /*δείχνει αν το στοιχείο Item υπάρχει ήδη στο
                                  ΔΔΑ*/
    LocPtr = *Root;
    Parent = NULL;
    Found = FALSE;
    while (!Found && LocPtr != NULL)
    {
        Parent = LocPtr;
        if (Item < LocPtr ->Data)
            LocPtr = LocPtr ->LChild;
        else if (Item > LocPtr ->Data)
            LocPtr = LocPtr ->RChild;
        else
            Found = TRUE;
    }
    if (Found)
        printf("Το %c είναι στοιχείο του ΔΔΑ\n", Item);
    else
    {
        LocPtr = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        LocPtr ->Data = Item;
        LocPtr ->LChild = NULL;
        LocPtr ->RChild = NULL;
        if (Parent == NULL)
            *Root = LocPtr;
        else if (Item < Parent ->Data)
            Parent ->LChild = LocPtr;
        else
            Parent ->RChild = LocPtr;
    }
}
```

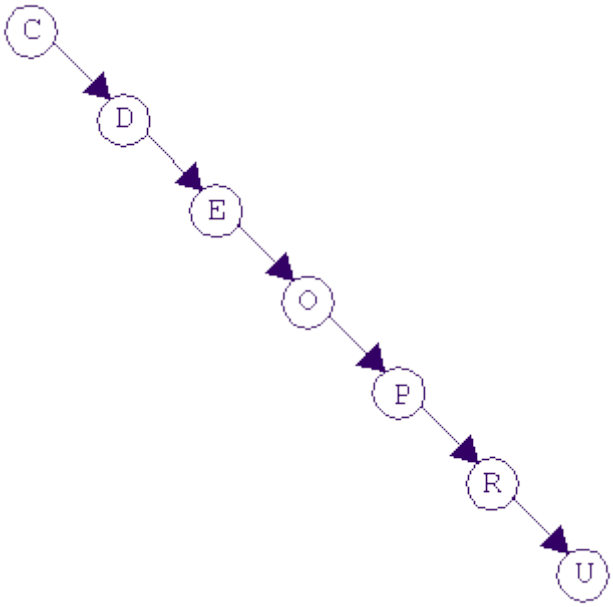
Η σειρά με την οποία εισάγονται τα στοιχεία σε ένα ΔΔΑ καθορίζει και τη δομή του δέντρου. Αν για παράδειγμα θέλουμε να εισαγάγουμε τα γράμματα O, R, D, E, P, U, C, E, R με αυτήν την σειρά σε ένα ΔΔΑ, θα προκύψει το παρακάτω **ισοζυγισμένο δέντρο**(**balanced tree**):



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



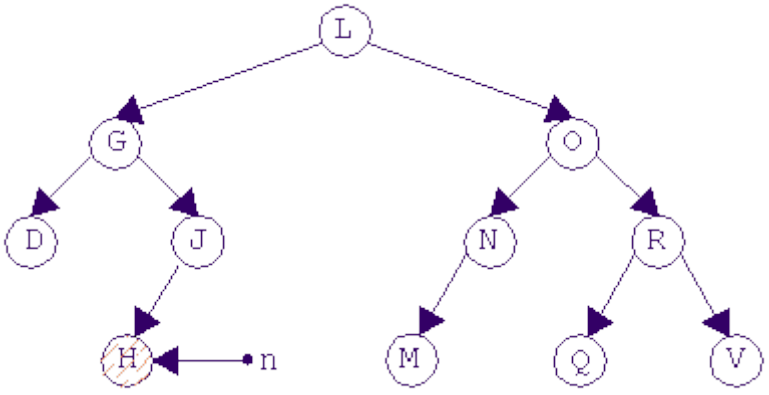
Τέλος, αν τα εισάγουμε με αλφαβητική σειρά, δηλαδή C, D, E, E, O, P, R, R, U, τότε το δέντρο εκφυλίζεται σε συνδεδεμένη λίστα:



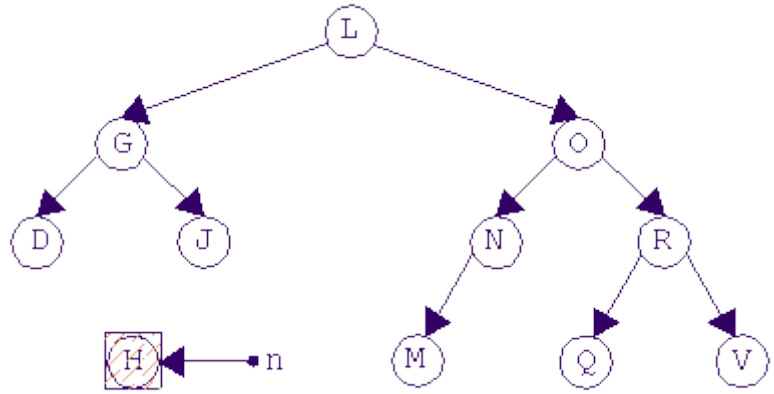
Για την **διαγραφή ενός κόμβου n από ένα ΔΔΑ** υπάρχουν τρεις περιπτώσεις:

1. Ο n είναι φύλλο
2. Ο n έχει ένα παιδί
3. Ο n έχει δύο παιδιά

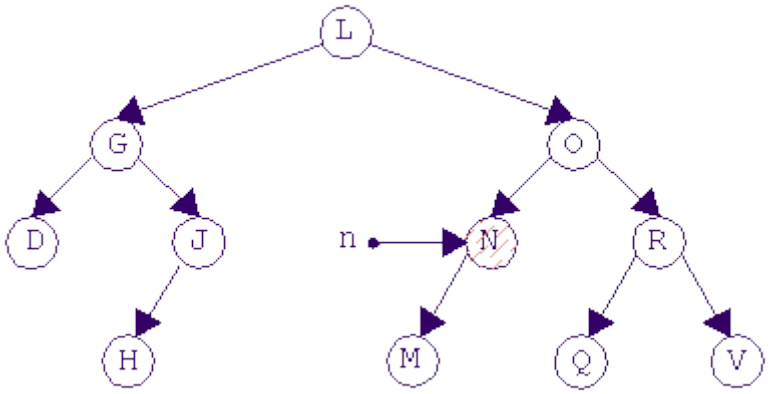
Όταν ο κόμβος που θέλουμε να διαγράψουμε είναι φύλλο, τότε απλά θέτουμε τον αριστερό ή δεξιό δείκτη του πατέρα του n, ανάλογα με το αν ο n είναι το αριστερό ή το δεξί παιδί του πατέρα του, ίσο με NULL. Αν, για παράδειγμα, θέλουμε να διαγράψουμε τον κόμβο H από το παρακάτω δέντρο:



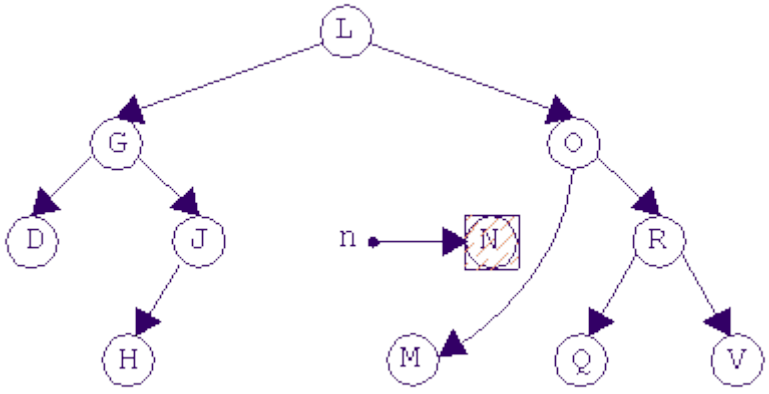
τότε θέτουμε απλά τον αριστερό δείκτη του κόμβου J ίσο με NULL και πετάμε τον κόμβο H, όπως φαίνεται στο σχήμα που ακολουθεί:



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. Αν δηλαδή θέλουμε να διαγράψουμε τον κόμβο N του ίδιου ΔΔΑ:



τότε αρκεί να θέσουμε τον αριστερό δείκτη του κόμβου O να δείχνει στον κόμβο M:



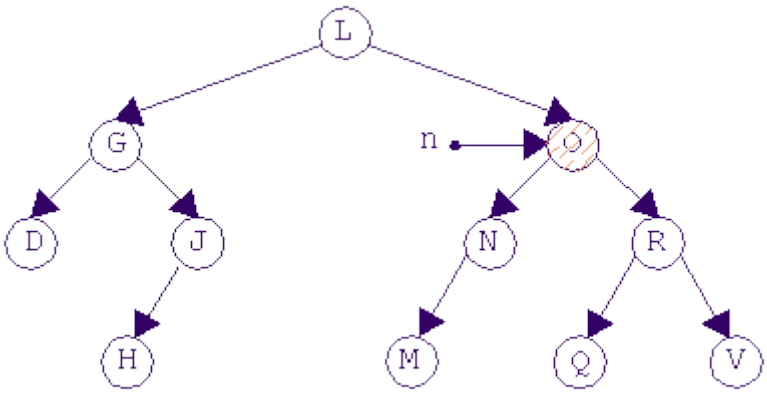
Στην πραγματικότητα, οι δυο παραπάνω περιπτώσεις μπορούν να συνδυαστούν σε μία, κατά την οποία ο προς διαγραφή κόμβος n έχει το πολύ ένα μη κενό υποδέντρο. Αν ο

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. Οι αντίστοιχες εντολές είναι:

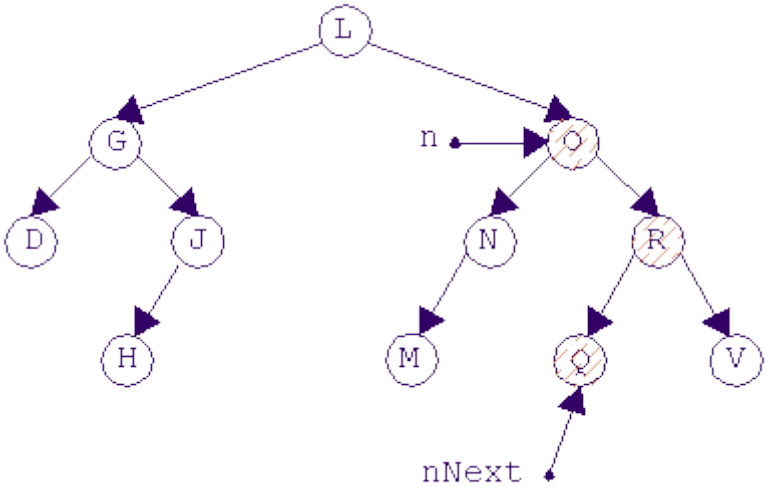
```
SubTree = n->LChild;
if (SubTree == NULL)
    SubTree = n->RChild;
    if (Parent == NULL)                    //* θα διαγραφεί η ρίζα του ΔΔΑ *)
      *Root = SubTree;
    else if (Parent->LChild == n)
            Parent->LChild = SubTree;
         else
            Parent->RChild = SubTree;
```

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.

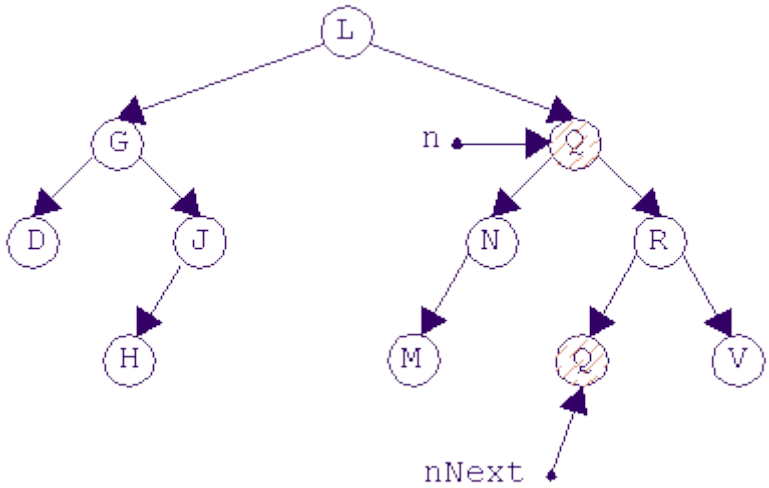
Για παράδειγμα, αν πρόκειται να διαγράψουμε τον κόμβο Ο στο ΔΔΑ:



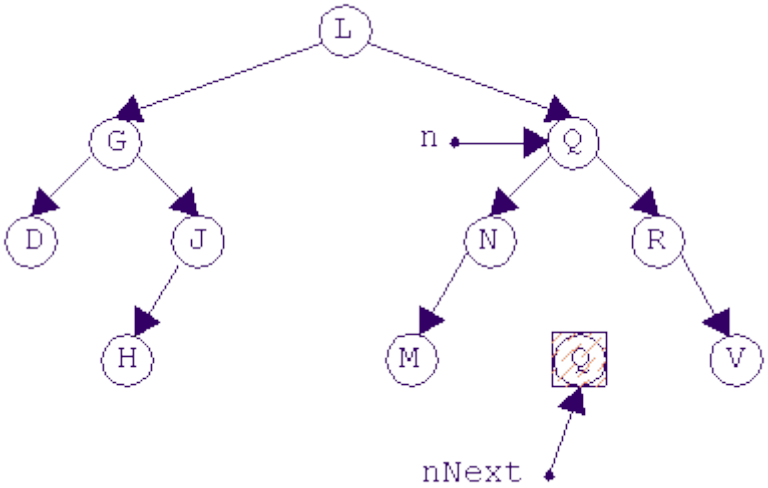
μπορούμε να εντοπίσουμε τον ενδοδιατεταγμένο επόμενό του ξεκινώντας από το δεξί παιδί του κόμβου Ο και κατεβαίνοντας αριστερά μέχρι να συναντήσουμε κόμβο χωρίς αριστερό υποδέντρο. Για τον κόμβο Ο ο ενδοδιατεταγμένος επόμενος είναι ο κόμβος Q:



Αν αντικαταστήσουμε τα περιεχόμενα του κόμβου O με αυτά του κόμβου Q, όπως δείχνει το παρακάτω σχήμα:



το μόνο που μένει είναι να διαγράψουμε τον κόμβο στον οποίο δείχνει ο δείκτης n Next. 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:



Στη συνέχεια παρουσιάζεται ο **αλγόριθμος διαγραφής στοιχείου από ένα ΔΔΑ** και για τις τρεις περιπτώσεις:

ΔΙΑΓΡΑΦΗ ΣΤΟΙΧΕΙΟΥ ΑΠΟ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ

/*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.*/

Κλήση του τροποποιημένου/*δες παρακάτω*/ **αλγόριθμου αναζήτησης σε ΔΔΑ** /*ο οποίος δέχεται το δείκτη *Root* που δείχνει στη ρίζα του ΔΔΑ και μια τιμή *KeyValue* και επιστρέφει μέσω της λογικής μεταβλητής *Found* την τιμή TRUE ή FALSE ανάλογα με το αν βρέθηκε ή όχι η τιμή *KeyValue* στο ΔΔΑ, τον δείκτη *n* στον κόμβο που περιέχει αυτή την τιμή και τον δείκτη *Parent* στον πατέρα του *n*/

Αν *Found* = **FALSE τότε** /*αν η τιμή *KeyValue* δεν βρέθηκε στο ΔΔΑ*/

Γράψε 'Το στοιχείο δεν βρίσκεται στο ΔΔΑ'

Αλλιώς /*η τιμή *KeyValue* βρέθηκε στο ΔΔΑ*/

Αν *n ->LChild* != **NULL και** *n ->RChild* != **NULL τότε**

/*αν ο αριστερός *LChild* και ο δεξιός *RChild* δεσμός του κόμβου του *n* στον οποίο βρέθηκε η τιμή *KeyValue* έχουν τιμή διάφορη του NULL, δηλαδή αν ο κόμβος αυτός έχει δύο παιδιά*/

/*εντοπισμός του ενδοδιατεταγμένου επόμενου κόμβου και του πατέρα του*/

nNext ← n ->RChild /*θέσε στο δείκτη *nNext* την τιμή του δεξιού δεσμού *RChild* του *n*, δηλαδή ο *nNext* δείχνει στο δεξί παιδί του κόμβου που

πρόκειται να διαγραφεί από το ΔΔΑ*/

Parent ← *n* /*θέσε στο δείκτη *Parent* την τιμή του δείκτη n, δηλαδή ο Parent είναι πλέον ο πατέρας του *nNext**/

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

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

Parent ← *nNext* /*με τις 2 διπλανές εντολές ενημερώνεται ο *nNext* (2η εντολή) ώστε να δείχνει στο αριστερό παιδί του κόμβου που έδειχνε μέχρι τώρα, καθώς επίσης και ο *Parent* (1η εντολή)ώστε να εξακολουθήσει να είναι ο πατέρας του *nNext**/

nNext ← *nNext ->LChild*

Τέλος_επανάληψης

Data(*n*) ← Data(*nNext*) /*αντιγραφή των περιεχομένων του *nNext* στον *n**/

n ← *nNext* /*αλλαγή του *n* ώστε να δείχνει στον επόμενο*/

Τέλος_αν

/*Συνεχίζουμε με την περίπτωση που ο κόμβος έχει το πολύ 1 παιδί*/

Subtree ← *n ->LChild* /*θέσε στον δείκτη *Subtree* την τιμή του αριστερού δεσμού *LChild* του *n*, δηλαδή ο *Subtree* δείχνει στο αριστερό παιδί του *n**/

Αν *Subtree* = **NULL τότε** /*αν ο δείκτης *Subtree* έχει την τιμή NULL, δηλαδή αν ο κόμβος *n* δεν έχει αριστερό παιδί*/

Subtree ← *n ->RChild* /*θέσε στον δείκτη *Subtree* την τιμή του δεξιού δεσμού *RChild* του *n*, δηλαδή ο *Subtree* δείχνει στο δεξί παιδί του *n**/

Τέλος_αν

Αν *Parent* = **NULL τότε** /*αν ο δείκτης *Parent* έχει την τιμή NULL, δηλαδή αν ο κόμβος *n* δεν έχει πατέρα ή αλλιώς αν ο κόμβος *n* είναι η ρίζα του ΔΔΑ*/

Root ← *Subtree* /*θέσε στον δείκτη *Root* την τιμή του δείκτη *Subtree,* δηλαδή η ρίζα του ΔΔΑ θα είναι πλέον το παιδί του *n**/

Αλλιώς_αν *Parent ->LChild* = *n* **τότε**

/*αν ο κόμβος *n* που πρόκειται να διαγραφεί είναι το αριστερό παιδί του πατέρα *Parent**/

Parent ->LChild ← *Subtree* /*θέσε στον αριστερό δεσμό του δείκτη *Parent* την τιμή του *Subtree*, δηλαδή ο πατέρας *Parent* του κόμβου που πρόκειται να διαγραφεί θα έχει πλέον ως αριστερό παιδί το παιδί του κόμβου που πρόκειται να διαγραφεί*/

Αλλιώς /*ο κόμβος *n* που πρόκειται να διαγραφεί είναι το δεξί παιδί του πατέρα *Parent**/

Parent ->RChild ← *Subtree* /*θέσε στον δεξιό δεσμό του δείκτη *Parent* την τιμή του *Subtree,* δηλαδή ο πατέρας *Parent* του κόμβου που πρόκειται

να διαγραφεί θα έχει πλέον ως δεξί παιδί το παιδί του κόμβου που πρόκειται να διαγραφεί*/

Τέλος_αν

Τέλος_αν

ΑΝΑΖΗΤΗΣΗ ΣΕ ΔΥΑΔΙΚΟ ΔΕΝΤΡΟ ΑΝΑΖΗΤΗΣΗΣ (2)

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

Λειτουργία: Αναζητά στο ΔΔΑ, με τη βοήθεια του δείκτη *LocPtr* που δείχνει στον τρέχοντα κάθε φορά κόμβο του 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.*/

1. *LocPtr* ← *Root* /*ενημέρωση του δείκτη *LocPtr* ώστε να δείχνει στη ρίζα *Root* του ΔΔΑ*/

   Parent ← **NULL**

   Found ← **FALSE**

2. **Όσο** *Found* = **FALSE και** *LocPtr* != **NULL επανάλαβε**

   /*όσο δεν βρέθηκε το στοιχείο και ο δείκτης *LocPtr* έχει τιμή διάφορη του NULL, δηλαδή δεν δείχνει σε ένα κενό υποδέντρο*/

   Αν *KeyValue* < Data(*LocPtr*) **τότε**

   /*αν η τιμή της *KeyValue* είναι μικρότερη από την τιμή του πεδίου κλειδιού του τρέχοντος κόμβου του ΔΔΑ, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης *LocPtr**/

   Parent ← *LocPtr*

   LocPtr ← *LocPtr ->LChild* /*θέσε στον δείκτη *LocPtr* την τιμή του αριστερού δεσμού *LChild* του τρέχοντος κόμβου *LocPtr,* προκειμένου η αναζήτηση να συνεχιστεί στο αριστερό υποδέντρο του τρέχοντος κόμβου*/

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

   /*αν η τιμή της *KeyValue* είναι μεγαλύτερη από την τιμή του πεδίου κλειδιού του τρέχοντος κόμβου του ΔΔΑ, δηλαδή του κόμβου στον οποίο δείχνει ο δείκτης *LocPtr**/

   Parent ← *LocPtr*

LocPtr ← *LocPtr ->RChild*	/*θέσε στον δείκτη *LocPtr* την τιμή του δεξιού δεσμού *RChild* του τρέχοντος κόμβου *LocPtr*, προκειμένου η αναζήτηση να συνεχιστεί στο δεξί υποδέντρο του τρέχοντος κόμβου*/
Αλλιώς	
Found ← **TRUE**	/*η τιμή *KeyValue* βρέθηκε στον κόμβο που δείχνει ο δείκτης *LocPtr**/
Τέλος_αν	
Τέλος_επανάληψης	

Η διαδικασία BSTDelete που ακολουθεί υλοποιεί τη λειτουργία τον αλγορίθμου της διαγραφής που παρουσιάστηκε παραπάνω:

```
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.*/
{
  BinTreePointer n,                          /*δείχνει στον κόμβο που περιέχει
                                               την τιμή KeyValue*/
                 Parent,                     /*πατέρας του n ή του nNext*/
                 nNext,                      /*ενδοδιατεταγμένος επόμενος του n*/
                 Subtree;                    /*δείκτης προς υποδέντρο του n*/
  boolean Found;                             /*TRUE αν η KeyValue βρεθεί*/

   BSTSearch2(*Root, KeyValue, &Found , &n, &Parent);
   if (!Found)
     print("Το στοιχείο %d δεν βρίσκεται στο ΔΔΑ\n", KeyValue);
   else
     {
       if (n ->LChild != NULL && n ->RChild != NULL)

         /*κόμβος με δύο παιδιά*/
         {
           /*Βρες τον ενδοδιατεταγμένο επόμενο και τον πατέρα του*/
           nNext = n->RChild;
           Parent = n;
           while (nNext-> LChild!= NULL)
             {
               Parent = nNext;
               nNext = nNext->LChild;
             }
           /*Αντιγραφή των περιεχομένων του nNext στον n και αλλαγή του n ώστε να
           δείχνει στον επόμενο*/
           n->Data = nNext->Data;
           n = nNext;
         }
       /*Συνεχίζουμε με την περίπτωση που ο κόμβος έχει το πολύ 1 παιδί*/
       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);
      }
}
```

Η διαδικασία BSTSearch2 είναι η παρακάτω:

```
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.*/
{
  (*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;
  }
}
```

Όλες οι συναρτήσεις περιλαμβάνονται στη διασύνδεση BstADT.h και στην υλοποίηση της BstADT.c, που υλοποιούν τον ΑΤΔ Δυαδικό Δέντρο Αναζήτησης και μπορεί να χρησιμοποιηθεί σε ένα πρόγραμμα C με την εντολή

#include "BstADT.h";

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. Το πρόγραμμα ValUsers.c χρησιμοποιεί το BstUsADT.h. Η διαφορά του από το BstUsADT.h είναι στον τύπο του BTreeElementType, το οποίο τώρα είναι μια εγγραφή με πεδία Id και Password. Το πεδίο Id είναι το κλειδί με βάση το οποίο γίνεται η αναζήτηση. Τα στοιχεία των έγκυρων χρηστών είναι αποθηκευμένα στο αρχείο Users.txt.