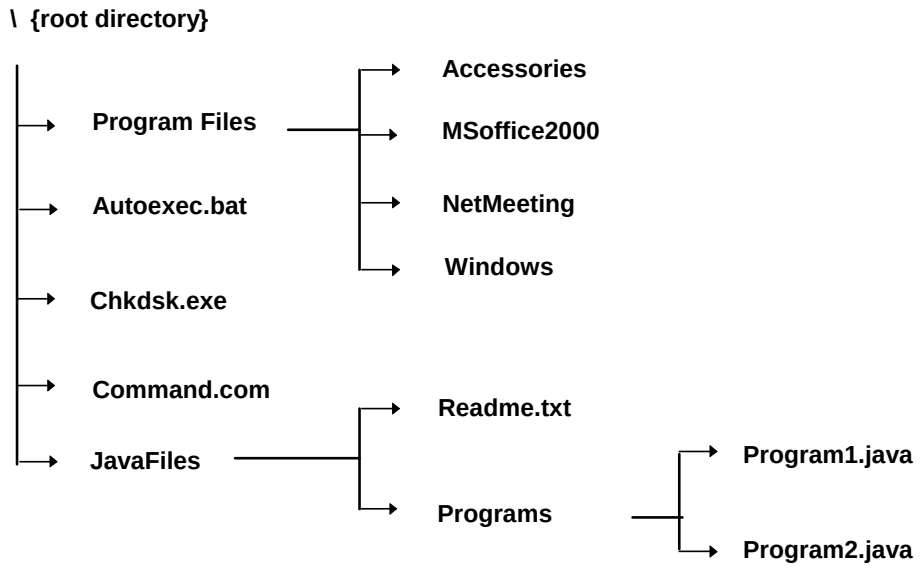
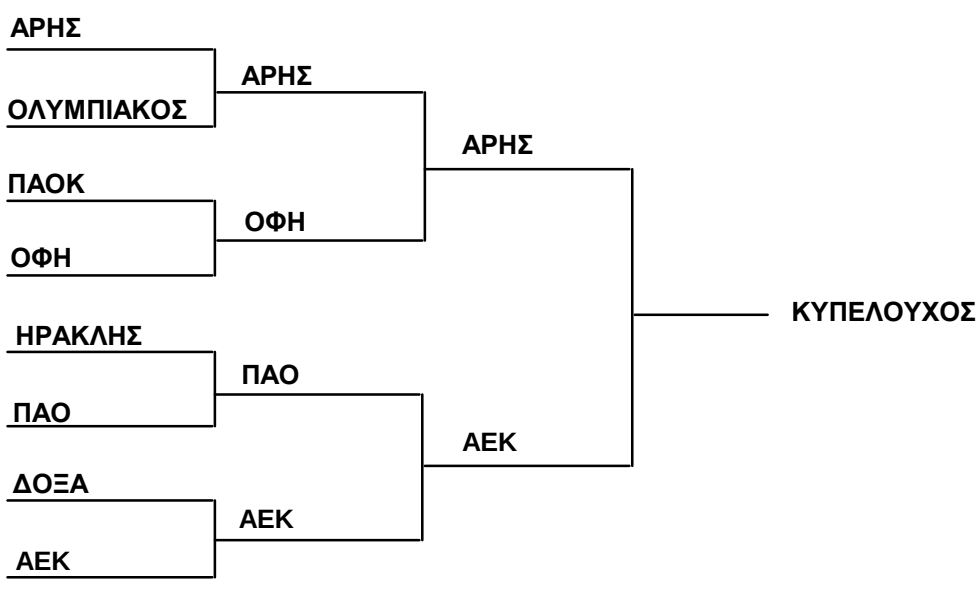
# 5 ΔΕΝΤΡΑ (*Trees*)

Οι περισσότερες δομές δεδομένων που εξετάσαμε μέχρι τώρα (λίστες, στοίβες, ουρές) ήταν γραμμικές (ή δομές δεδομένων μιας διάστασης). Στην παράγραφο αυτή θα ασχοληθούμε με τις μή-γραμμικές δομές δεδομένων που ονομάζονται δέντρα. Συναντάμε τα δέντρα σε πάρα πολλές εφαρμογές της επιστήμης των υπολογιστών. Κλασσικό παράδειγμα αποτελεί η οργάνωση του συστήματος των αρχείων, με τη μορφή δέντρου (βλέπε σχ. 5.1), που ακολουθούν τα περισσότερα λειτουργικά συστήματα (π.χ. M.Windows, UNIX κλπ). Τα δέντρα χρησιμοποιούνται επίσης πολύ συχνά, ως μοντέλα αναπαράστασης προβλημάτων της καθημερινής μας ζωής. Χαρακτηριστικά παραδείγματα αποτελούν το "γενεαλογικό" δέντρο και η δενδροειδής αναπαράσταση αγώνων κυπέλου, μεταξύ ομάδων (σχ. 5.2).




**Σχήμα 5.1 :** Δενδροειδής αναπαράσταση αρχείων στα Windows




**Σχήμα 5.2** : Αναπαράσταση αγώνων κυπέλου με μορφή δέντρου

Παρά το γεγονός ότι η έννοια του δέντρου γίνεται έμμεσα κατανοητή από έναν οποιονδήποτε οπτικό τρόπο αναπαράστασης του, η δομή δεδομένων δέντρο πρέπει να οριστεί με έναν αυστηρά μαθηματικό τρόπο. Υπάρχουν πολλοί ισοδύναμοι ορισμοί ενός δέντρου, δίνουμε στη συνέχεια δύο από αυτούς.

***Ορισμός 1:***

**Δέντρο (*tree*)** είναι ένα σύνολο **Τ** από **κόμβους (*nodes*)**, τέτοιο ώστε είτε:

**(α)** Το **Τ** είναι κενό ή

**(β)** Το **Τ** περιλαμβάνει ένα ξεχωριστό κόμβο, R, που ονομάζεται **ρίζα (*root* )** του **Τ** και οι υπόλοιποι **κόμβοι Τ – {R}** χωρίζονται σε μηδέν ή περισσότερα σύνολα κόμβων, **Τ1, Τ2, . . ., Τn**, που είναι ξένα μεταξύ τους και τα οποία είναι με τη σειρά τους δέντρα. **Τα Τ1, Τ2, . . ., Τn**, ονομάζονται υποδέντρα του Τ.

***Ορισμός 2:***

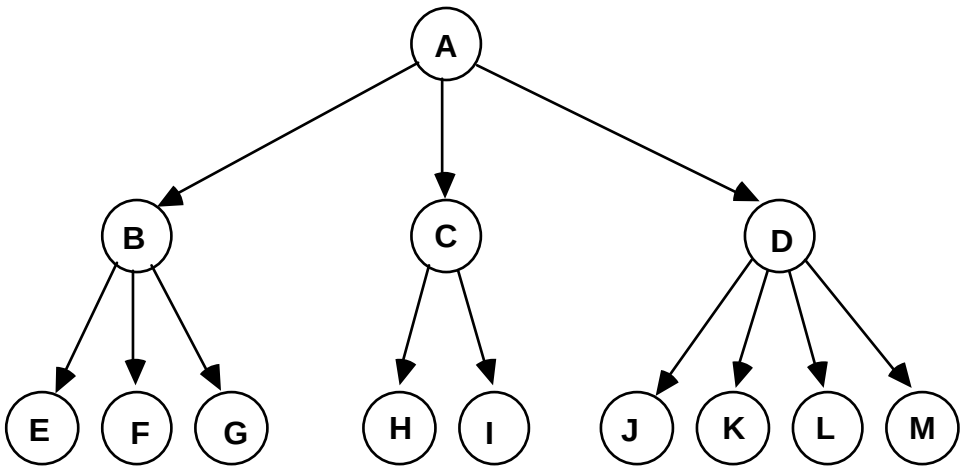
**Δέντρο** είναι μία συλλογή από στοιχεία, που ονομάζονται **κόμβοι**. Οι κόμβοι του δέντρου συνδέονται μεταξύ τους με τη βοήθεια **ακμών (*arcs*)** με βάση τους εξής κανόνες:

**(α)** Υπάρχει ένας και μόνον ένας κόμβος στον οποίο δεν καταλήγει καμία ακμή (**η Ρίζα (*Root*)** του δέντρου).

**(β)** Σε όλους τους υπόλοιπους κόμβους καταλήγει υποχρεωτικά μία και μόνο μία ακμή.

Ο πρώτος ορισμός είναι αναδρομικός και είναι καλό να τον έχει κανείς υπ' όψη του, όταν θα συζητήσουμε αργότερα, τις διάφορες ιδιότητες και τους αλγόριθμους επεξεργασίας των δέντρων. Ο δεύτερος ορισμός βρίσκεται σε μεγαλύτερη αντιστοιχία με τον οπτικό τρόπο αναπαράστασης ενός δέντρου.

Ας εξετάσουμε τους δύο ορισμούς χρησιμοποιώντας για παράδειγμα το δέντρο του σχήματος 5.3 :




**Σχήμα 5.3** : Συνήθης οπτική αναπαράσταση δέντρου

Σύμφωνα με τον δεύτερο ορισμό βλέπουμε ότι μόνο στον κόμβο *A*, που αποτελεί τη ρίζα του δέντρου δεν καταλήγει καμία ακμή, ενώ σε όλους τους άλλους κόμβους καταλήγει μία μόνο ακμή. Παρατηρήστε ότι συνήθως σχεδιάζουμε ένα δέντρο με τη

ρίζα του επάνω και τους υπόλοιπους κόμβους από κάτω (παρόλο που αυτό μοιάζει παράξενο !).

Σύμφωνα με τον πρώτο ορισμό το δέντρο αποτελείται από ένα σύνολο 13 κόμβων:

**T = { *A*, B, *C*, D, E, F, *G*, H, I, J, K, L, M }**

Η ρίζα του είναι ***A*** και το σύνολο **T - { *A* }** χωρίζεται σε τρία υποδέντρα:

**T1 = { B, E, F, *G* }, T2 = { *C*, H, I }** και **T3 = { D, J, K, L, M }**

Τα **T1**, **T2** και **T3** έχουν σαν ρίζα τον κόμβο **B** , ***C***, και **D** αντίστοιχα. Τα δε **σύνολα T1 - { B}**, **T2 - { *C* }** και **T3 - { D }** χωρίζονται αντίστοιχα στα υποδέντρα:

**T1.1 = { E }, T1.2 = { F }, T1.3 = { *G* }**
**T2.1 = { H }, T2.2 = { I }**
**T3.1 = { J }, T3.2 = { K }, T3.3 = { L }, T3.4 = { M }**

Κάθε ένα από τα τελευταία υποδέντρα είναι σύνολα που περιέχουν έναν μόνον κόμβο. Ο κόμβος αυτός αποτετελεί και τη ρίζα του αντίστοιχου υποδέντρου και αν αφαιρεθεί το σύνολο που προκύπτει είναι το κενό (περίπτωση (α) του ορισμού).

Μερικοί ακόμα ορισμοί-ιδιότητες των δέντρων είναι απαραίτητοι:

- Κάθε κόμβος (εκτός από τη ρίζα) έχει ακριβώς έναν κόμβο από πάνω του, ο οποίος ονομάζεται **πατέρας (*father*)**. Για παράδειγμα στο δέντρο του σχήματος 5.3, ο **A** είναι ο πατέρας του **B** και ο **D** είναι ο πατέρας του **K**.

- Οι κόμβοι που βρίσκονται ακριβώς κάτω από έναν κόμβο (συνδέονται με ακμές που ξεκινούν από αυτόν), ονομάζονται **παιδιά** του **(*children*)**. Πολλές φορές μπορεί να αναφερόμαστε (κατ' αναλογία με τα γενεαλογικά δέντρα) στους **απογόνους (*successors*)** ή **προγόνους (*ancestors*)** ενός κόμβου. Στο παράδειγμα του σχήματος 5.3, οι κόμβοι **E**, **F**, ***G*** είναι παιδιά του κόμβου **B**. Οι κόμβοι **H** και **I** είναι απόγονοι του **A** (όχι οι μοναδικοί) και ο κόμβος ***G*** έχει πρόγονο τον **A**.

- Ενας κόμβος ο οποίος δεν έχει κανένα μη-κενό υποδέντρο (κόμβος χωρίς παιδιά) ονομάζεται **τερματικός (*terminal*)** κόμβος ή **φύλλο (*leaf*)** του δέντρου. Στο δέντρο του σχήματος 5.3, τερματικοί κόμβοι είναι οι **E**, **F**, ***G***, **H**, **I**, **J**, **K**, **L** και **M**. Ολοι οι υπόλοιποι κόμβοι του δέντρου ονομάζονται **μή-τερματικοί (*non-terminals*)**. Πολλές φορές οι τερματικοί κόμβοι ενός δέντρου αναφέρονται και σαν **εξωτερικοί (*externals*)** ενώ οι μή τερματικοί σαν **εσωτερικοί (*internals*)**.

- Μία ακολουθία κόμβων, ενός δέντρου, οι οποίοι συνδέονται διαδοχικά μεταξύ τους με τη βοήθεια ακμών, ονομάζεται **μονοπάτι (*path*)**. Οι ακολουθίες **<A**, **B**, ***G*> και <A**, **D**, **K>** είναι δύο από τα μονοπάτια του δέντρου του σχήματος 5.3.

- Οι κόμβοι ενός δέντρου χωρίζονται σε **επίπεδα (*levels*)**. Ενας κόμβος βρίσκεται στο επίπεδο **N**, εάν **N** είναι ο αριθμός των ακμών του μονοπατιού, που συνδέει τη ρίζα με τον κόμβο αυτό. Στο παράδειγμα του σχήματος 5.3, ο ***A*** ανήκει στο επίπεδο **0**, οι **B**, ***C***, **D** ανήκουν στο επίπεδο **1**, ενώ οι κόμβοι **E**, **F**, ***G***, **H**, **I**, **J**, **K**, **L, M** αποτελούν το επίπεδο **2**.

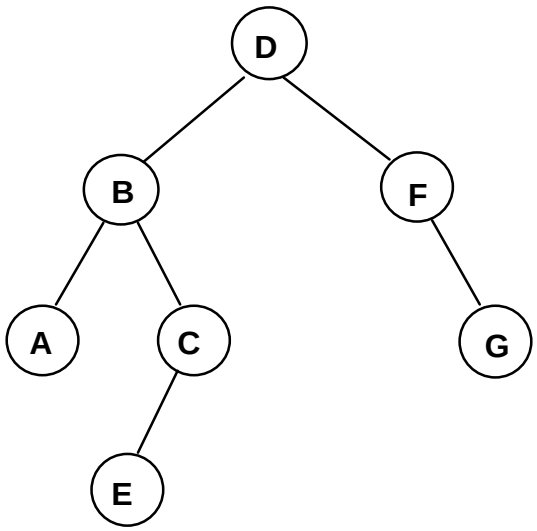
- Ορίζουμε σαν **ύψος (*heigth*)** ή **βάθος (*depth*)** ενός κόμβου τον αριθμό του επιπέδου στο οποίο βρίσκεται ο κόμβος αυτός. Το ύψος (ή βάθος) ενός δέντρου ταυτίζεται με το μέγιστο ύψος (ή βάθος) των κόμβων του δέντρου.

- Ονομάζουμε **βαθμό (*degree*)** ενός κόμβου τον αριθμό των υποδέντρων του

- Ονομάζουμε **δάσος (*forest*)** ένα σύνολο από **N >= 0** δέντρα που είναι ξένα μεταξύ τους.

## 5.1 Δυαδικά Δέντρα

***Ορισμός 3:***
**Δυαδικό δέντρο (*binary tree*)** είναι ένα δέντρο του οποίου κάθε κόμβος έχει το πολύ δύο υποδέντρα. Τα υποδέντρα του δυαδικού δέντρου ονομάζονται αριστερό και δεξιό υποδέντρο αντίστοιχα.




**Σχήμα 5.4** : Δυαδικό δέντρο

### 5.1.1 Υλοποίηση Δυαδικού Δέντρου με τη Βοήθεια Δεικτών

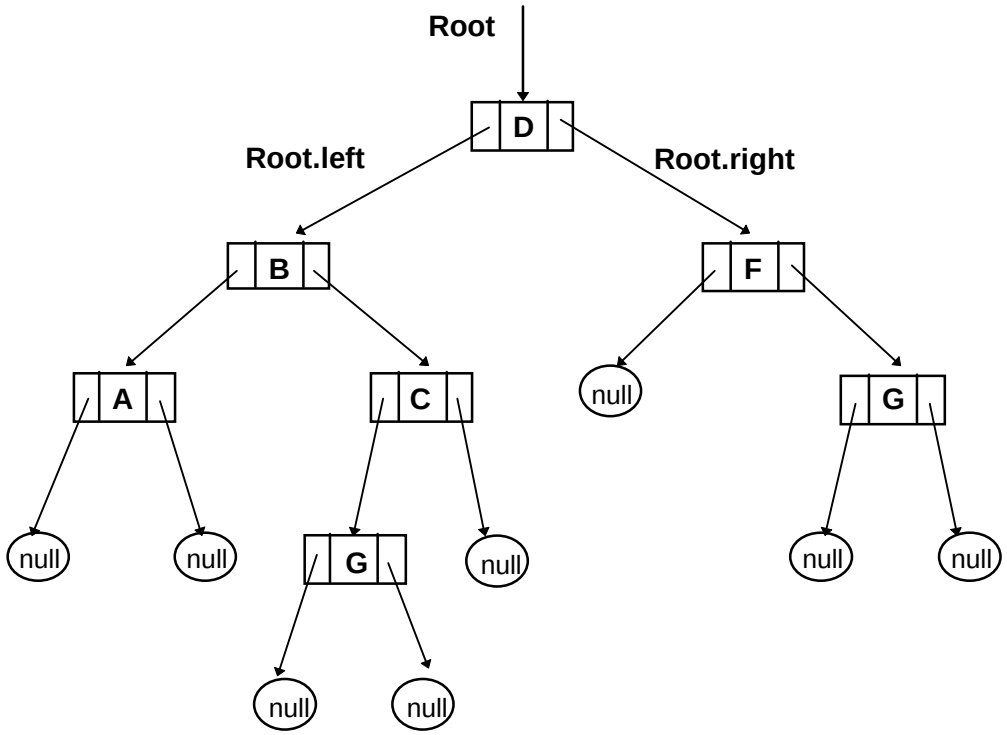
Μπορούμε να αναπαραστήσουμε ένα δυαδικό δέντρο με τη βοήθεια μιας δυναμικής δομής δεδομένων. Κάθε κόμβος του δέντρου αναπαρίσταται με τη βοήθεια μιας τριάδας η οποία περιλαμβάνει ένα πεδίο για την αποθήκευση της πληροφορίας του κόμβου και δύο πεδία τύπου δείκτη σε κόμβο δέντρου, που δείχνουν στο αριστερό και δεξιό υποδέντρο αντίστοιχα. Η τριάδα αυτή των δηλώσεων ορίζονται σαν μέλη της κλάσης **TreeNode**. Δίνεται επίσης στα πλαίσια της κλάσης η πράξη δημιουργίας ενός τερματικού κόμβου:

```
class TreeNode
{
    TreeNode left;
    int item;
    TreeNode right;

    public TreeNode(int data)
    {   item = data;
        left = right = null;
    }
   ..........
}
```

Για να αναφερθούμε σε ένα δυαδικό δέντρο χρησιμοποιούμε ένα δείκτη (pointer), ο οποίος δείχνει στην ρίζα του. Ο δείκτης αυτός (root) ορίζεται στα πλαίσια της κλάσης που υλοποιεί το δέντρο αυτό

Η χρήση δυναμικής δομής δεδομένων είναι τις περισσότερες φορές η πιο κατάλληλη για την αναπαράσταση ενός δυαδικού δέντρου και είναι αυτή που θα χρησιμοποιήσουμε για την υλοποίηση των διαφόρων πράξεων-λειτουργιών που ορίζονται πάνω στα δυαδικά δέντρα.

Στο σχήμα 5.6 φαίνεται η γραφική αναπαράσταση του δέντρου του σχήματος 5.4 με τη βοήθεια δεικτών. Παρατηρήστε ότι στην περίπτωση που το αριστερό ή το δεξιό υποδέντρο ενός κόμβου είναι κενό τότε η αντίστοιχη τιμή του δείκτη **left** ή **right** έχει την τιμή **null**.

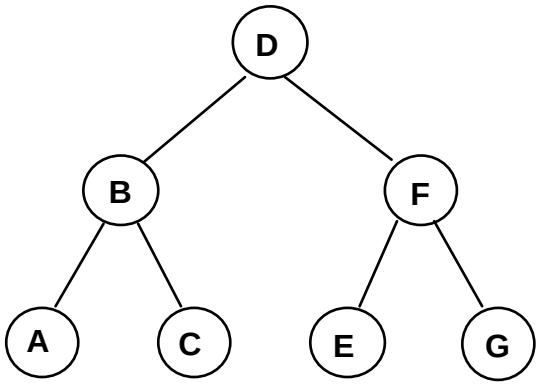



**Σχήμα 5.6**: Αναπαράσταση δυαδικού δέντρου με τη βοήθεια δεικτών

## 5.2 Βασικές Μέθοδοι Διέλευσης από τους Κόμβους ενός Δυαδικού Δέντρου

Οι περισσότεροι αλγόριθμοι χειρισμού δυαδικών δέντρων απαιτούν ένα συστηματικό τρόπο "επίσκεψης", όλων των κόμβων του δέντρου. Οι κυριότεροι τρόποι επίσκεψης είναι τρεις και περιγράφονται στη συνέχεια.

Έστω για παράδειγμα το δυαδικό δέντρο του παρακάτω σχήματος:




**1ος τρόπος: Ενθεματική Διέλευση (*Inorder Traversal* )**

**Βήμα 1** : Διέλευση Αριστερού υποδέντρου
**Βήμα 2** : Επίσκεψη ρίζας
**Βήμα 3** : Διέλευση Δεξιού υποδέντρου

Για το δέντρο του παράδειγματος έχουμε: **A B C D E F G**

Η παρακάτω αναδρομική μέθοδος της κλάσης **BSTree** (βλέπε και τμήμα κώδικα 1) υλοποιεί την ενθεματική διέλευση από τους κόμβους ενός δυαδικού δέντρου. Στην περίπτωση που ο αλγόριθμος επισκέπτεται έναν κόμβο το περιεχόμενό του τυπώνεται στην οθόνη:

```
private void inOrder(TreeNode node)
{   if (node == null)
        return;
    inOrder(node.left);
    System.out.print(node.item + " ");
    inOrder(node.right);
}
```

**2ος τρόπος: Προθεματική Διέλευση (*Preorder Traversal*)**

**Βήμα 1** : Επίσκεψη ρίζας
**Βήμα 2** : Διέλευση Αριστερού υποδέντρου
**Βήμα 3** : Διέλευση Δεξιού υποδέντρου

Για το δέντρο του παράδειγματος έχουμε: **D B A C F E G**

Η υλοποίηση της αναδρομικής μεθόδου **preOrder** είναι η εξής:

```
private void preOrder(TreeNode node)
{   if (node == null)
        return;
    System.out.print(node.item + " ");
    preOrder(node.left);
    preOrder(node.right);
}
```

**<u>3ος τρόπος</u>: Επιθεματική Διέλευση (*Postorder Traversal*)**

**Βήμα 1** : Διέλευση Αριστερού υποδέντρου
**Βήμα 2** : Διέλευση Δεξιού υποδέντρου
**Βήμα 3** : Επίσκεψη ρίζας

Για το δέντρο του παράδειγματος έχουμε: **A C B E G F D**

Η υλοποίηση της αναδρομικής μεθόδου **postOrder** είναι η εξής:

```
private void postOrder(TreeNode node)
{   if (node == null)
        return;
    postOrder(node.left);
    postOrder(node.right);
    System.out.print(node.item + " ");
}
```

## 5.3 Κατασκευή Δυαδικού Δέντρου Αναζήτησης

Το δυαδικό δέντρο αναζήτησης αποτελεί μία ειδική κατηγορία δυαδικού δέντρου, το οποίο κατασκευάζεται με βάση τον παρακάτω αλγόριθμο.

**ΒΗΜΑ 1**:

Το πρώτο δεδομένο χρησιμοποιείται για τη δημιουργία της ρίζας του δέντρου.

**ΒΗΜΑ 2**:

Τα επόμενα δεδομένα τοποθετούνται στο δέντρο, έτσι ώστε σε σχέση με οποιονδήποτε κόμβο να ισχύει το εξής: Όλες οι τιμές που βρίσκονται στο αριστερό υποδέντρο είναι μικρότερες από την τιμή του κόμβου και όλες οι τιμές που βρίσκονται στο δεξιό υποδέντρο είναι μεγαλύτερες.

Με βάση τα βήματα 1 και 2 ορίζεται η παρακάτω πράξη εισαγωγής στοιχείου σε ένα δυαδικό δέντρο:

```
public void insertNode(int d)
{
    if (root == null)
        root = new TreeNode(d);
    else
        root.insert(d);
}

public void insert(int d)
    {   if (d < item)
        {   if (left == null)
                left = new TreeNode(d);
            else
                left.insert(d);
        }
        else
            if (right == null)
                right = new TreeNode(d);
            else
                right.insert(d);
    }
```

Η μέθοδος **insertNode** ανήκει στην κλάση **BSTree** ενώ η μέθοδος **insert** είναι μέθοδος της κλάσης **TreeNode**. Η επαναληπτική χρησιμοποίηση της insertNode (για την εισαγωγή μιας ακολουθίας δεδομένων) δημιουργεί το δυαδικό δέντρο αναζήτησης. Ενα τέτοιο δέντρο αποτελεί μία δομή καταχώρησης πληροφορίας με ταξινομημένο τρόπο. Μπορούμε εύκολα να παρατηρήσουμε ότι, εάν εφαρμόσουμε τον ενθεματικό τρόπο διέλευσης από τους κόμβους ενός δυαδικού δέντρου αναζήτησης τα περιεχόμενα των κόμβων θα εκτυπωθούν σε αύξουσα τάξη. Ολόκληρη η υλοποίηση ενός κόμβου δυαδικού δέντρου καθώς και η υλοποίηση του δυαδικού δέντρου αναζήτησης δίνεται στα τμήματα κώδικα 1 και 2 αντίστοιχα.

```
class TreeNode

//Ορισμός κόμβου ενός Δυαδικού Δέντρου
{
      TreeNode left;
      int item;
      TreeNode right;

      public TreeNode(int data)
      {   item = data;
          left = right = null;
      }

      public void insert(int d)
      {   if (d < item)
          {   if (left == null)
                  left = new TreeNode(d);
              else
                  left.insert(d);
          }
          else
              if (right == null)
                  right = new TreeNode(d);
              else
                  right.insert(d);
      }
}
```

**Τμήμα Κώδικα 1:** Κόμβος Δυαδικού Δέντρου

```
import java.io.*;

public class BSTree

//Υλοποίηση ενός Δυαδικού Δέντρου Αναζήτησης
{
      TreeNode root;

      public BSTree( ) {
           root = null;
      }

      public void insertNode(int d)
      {
           if (root == null)
                root = new TreeNode(d);
           else
                root.insert(d);
      }

      public void inOrderTraversal( ) {
           inOrder(root);
      }

      private void inOrder(TreeNode node)
      {    if (node == null)
                return;
           inOrder(node.left);
           System.out.print(node.item + " ");
           inOrder(node.right);
      }
}
```

**Τμήμα Κώδικα 2:** Δυαδικό Δέντρο Αναζήτησης (ΔΔΑ)

Στο τμήμα κώδικα 3 φαίνεται πως μπορεί να χρησιμοποιηθεί το δυαδικό δέντρο αναζήτησης για την ταξινόμηση ενός πίνακα. Αν τροποποιήσουμε την inOrderTraversal που δόθηκε στα τμήματα κώδικα 1 και 2 κατάλληλα, μπορούμε αντί της εμφάνισης των δεδομένων στην οθόνη να τα τοποθετούμενα ξανά στον πίνακα.

```
public class BSTreeSort
{
      public static void main (String args[ ])
      {
           BSTree BT = new BSTree();
           int A[ ] = {47, 12, 19, 50, 17, 10, 69, 54, 42, 8};

           for (int i = 0; i < A.length; i++)
           BT.insertNode(A[i]);

           BT.inOrderTraversal( );
      }
}
```

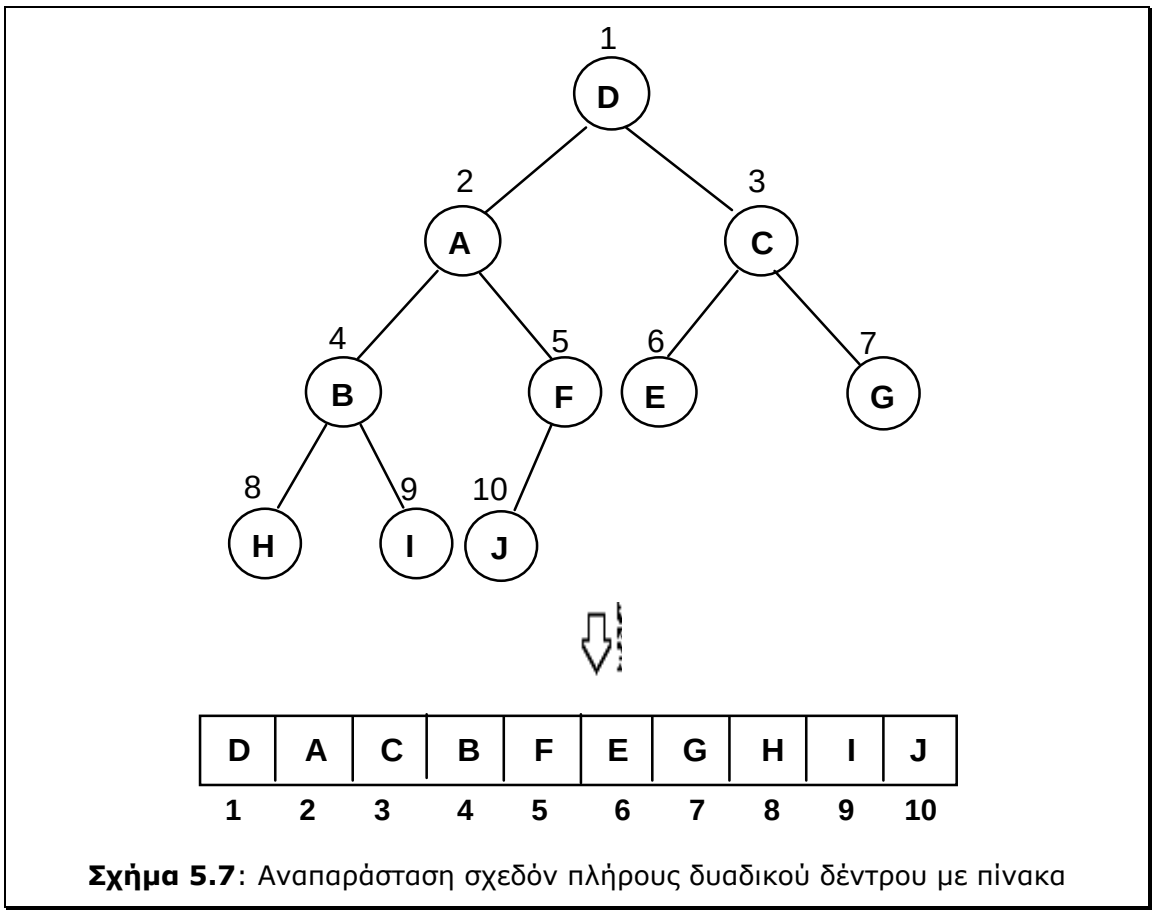
**Τμήμα Κώδικα 3**: Ταξινόμηση με τη βοήθεια ΔΔΑ

## 5.4 Υλοποίηση Σωρού (*Heap* ) με τη βοήθεια Δυαδικού Δέντρου

Ένα δυαδικό δέντρο βάθους **N** ονομάζεται **πλήρες (*complete*)** όταν έχει όλους τους κόμβους του επιπέδου **N** συμπληρωμένους.

Ένα δυαδικό δέντρο βάθους **N** ονομάζεται **σχεδόν πλήρες (*almost complete*)** όταν έχει όλους τους κόμβους του επιπέδου **N-1** συμπληρωμένους και οι κόμβοι που υπάρχουν στο **N-οστό** επίπεδο είναι τοποθετημένοι όσο το δυνατόν πιο αριστερά.

Η τοποθέτηση των κόμβων σε ένα σχεδόν πλήρες δυαδικό δέντρο είναι τέτοια που μας επιτρέπει να αριθμήσουμε με ένα συστηματικό τρόπο τους κόμβους αυτούς. Αυτή η συστηματική αρίθμηση (από πάνω προς τα κάτω και από αριστερά προς τα δεξιά κάνει δυνατή την αναπαράσταση ενός σχεδόν πλήρους δυαδικού δέντρου με τη βοήθεια ενός απλού γραμμικού πίνακα, όπως φαίνεται στο σχήμα 5.7. (Θεωρούμε ότι το στοιχείο με ενδείκτη 0 του πίνακα δεν χρησιμοποιείται)




**Σχήμα 5.7**: Αναπαράσταση σχεδόν πλήρους δυαδικού δέντρου με πίνακα

Ένα δυαδικό δέντρο ονομάζεται ***σωρός (heap)*** όταν ισχύουν επιπλέον οι παρακάτω δύο προϋποθέσεις:

**(α)** το δυαδικό δέντρο είναι σχεδόν πλήρες και
**(β)** οι τιμές των κόμβων είναι τοποθετημένες με τέτοιο τρόπο ώστε ο κάθε κόμβος πατέρας να έχει μεγαλύτερη τιμή από τα παιδιά του.

Οι βασικές πράξεις που ορίζονται για τη δομή δεδομένων σωρός, είναι η πράξη της εισαγωγής και η πράξη της διαγραφής ενός στοιχείου.

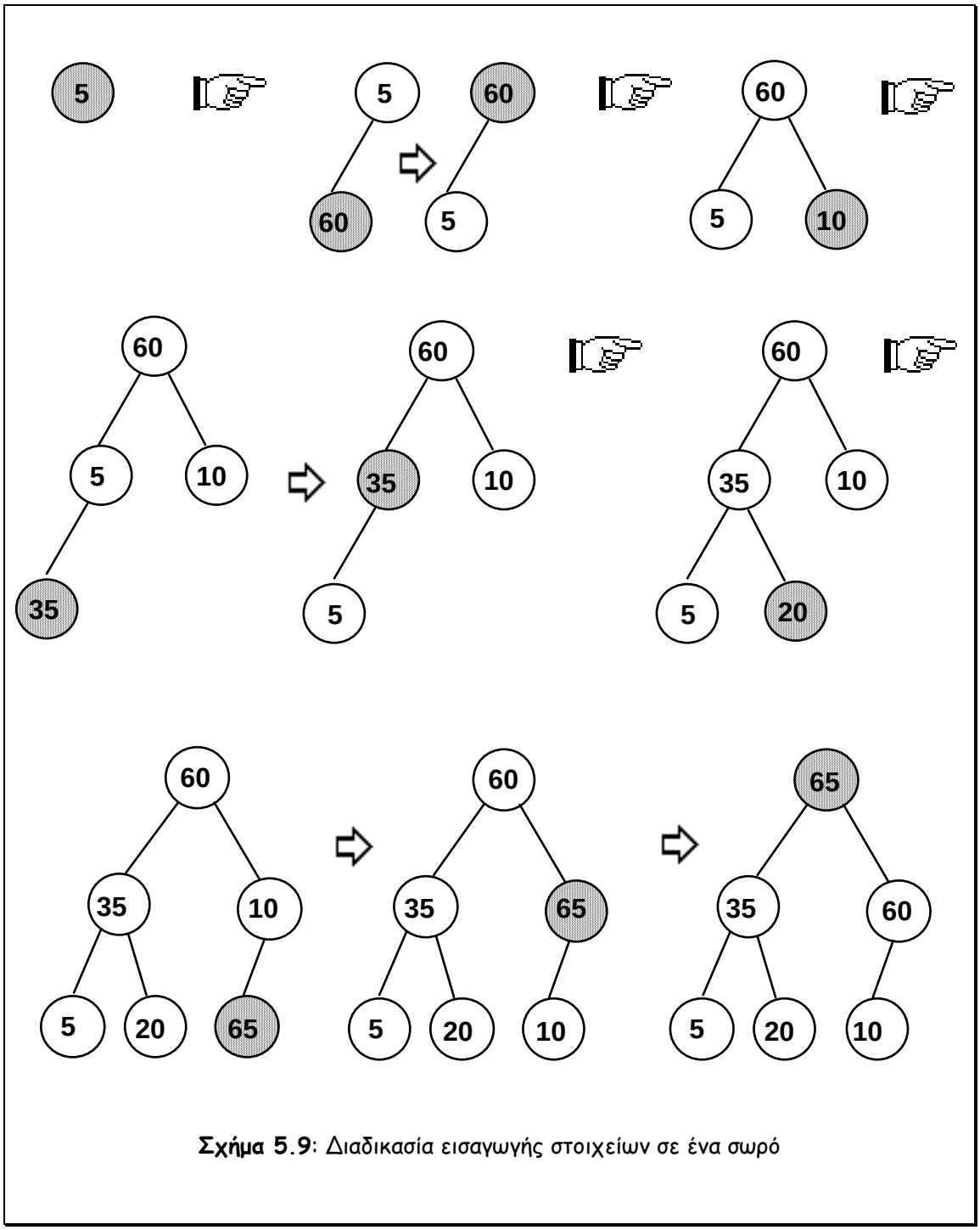
Η πράξη της εισαγωγής φροντίζει ώστε ο σωρός που προκύπτει να έχει το μεγαλύτερο στοιχείο του πάντοτε στην κορυφή του δέντρου (κορυφή του σωρού). Στο σχήμα 5.9 φαίνεται μία σειρά από διαδοχικές πράξεις εισαγωγής στοιχείων οι οποίες δημιουργούν ένα σωρό.

Η πράξη διαγραφής αφαιρεί πάντοτε το στοιχείο που βρίσκεται στην κορυφή του δέντρου και φροντίζει να αναδιατάσσει το δέντρο, έτσι ώστε το αμέσως μεγαλύτερο στοιχείο να βρεθεί στην κορυφή του σωρού. Η κλάση **Heap** που υλοποιεί το σωρό δίνεται στο τμήμα κώδικα 3.

```
public class Heap /* Ενδεικτική Υλοποίηση */
{
      private int[ ] btree;
      private int index;
      private int capacity;

      public Heap(int cap) {
           capacity = cap;
           btree = new int[capacity];
           index = 0;
      }

      public void heapInsert(int item) {
           int father, son;
           son = ++index;
           btree[son] = item;  // αρχικά το item στο τέλος της heap
```

```
        father = son/2;
        while( (son>1) && (btrre[son]>btree[father]) )
        {
            int b = btree[father];
            btree[father] = btree[son];
            btree[son] = b;
            son = father;
            father = son/2;
        }
    }

    public int heapDelete( )
    {
        int father, son;
        int oldItem, p;
        oldItem = btree[index];
        btree[index] = btree[1];
        p = btree[1];
        index--;
        father = 1;
        if ( (index > 2) && (btree[2] > btree[3]) )
            son = 2;
        else
            son = 3;
        while ( (son <= index) && (btree[son] > oldItem) )
        {
            btree[father] = btree[son];
            father = son;
            son = father * 2;
            if ( ( son+1 <= index) && (btree[son+1] > btree[son]) )
                son = son +1;
        }
        btree[father] = oldItem;
        return p;
    }
```

**Τμήμα Κώδικα 4**




**Σχήμα 5.9**: Διαδικασία εισαγωγής στοιχείων σε ένα σωρό