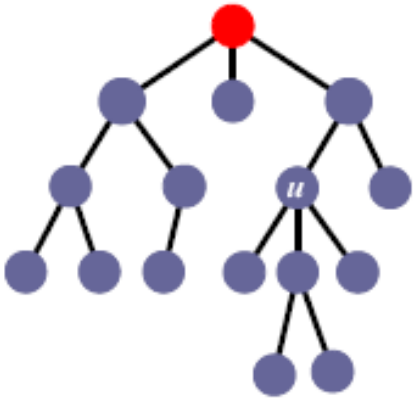
Δομές Δεδομένων (Data Structures)

Δένδρα (Trees)

- Βασικές Έννοιες.
- Δυαδικά Δένδρα.
- Δυαδικά Δένδρα Αναζήτησης.
- AVL Δένδρα.

Δένδρα: Βασικές Έννοιες

- ❑ Ορισμοί
- ❑ Λειτουργίες
- ❑ Υλοποιήσεις
- ❑ ΑΤΔ



✓ **Δένδρο: μοντέλο ιεραρχικής δομής.**

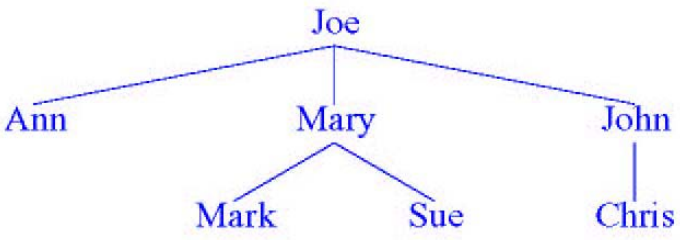
✓ **Εφαρμογές: γενεαλογικά δένδρα, οργάνωση επιχειρήσεων, ιεραρχική οργάνωση, ταχύτερη πρόσβαση σε κείμενο, … κλπ.**

Δένδρα: Ορισμοί

(αναδρομικός ορισμός)

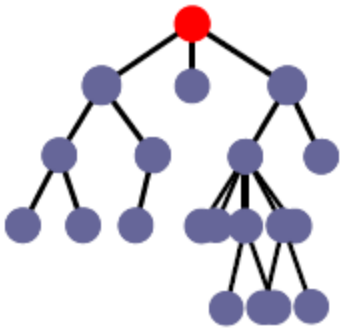
Ένα δένδρο t είναι ένα πεπερασμένο μη κενό σύνολο στοιχείων. Ένα από τα στοιχεία αυτά ονομάζεται ρίζα, ενώ τα υπόλοιπα στοιχεία (αν υπάρχουν) επιμερίζονται σε δένδρα που ονομάζονται υποδένδρα του t.

- ✓ **Βαθμός ενός στοιχείου είναι ο αριθμός των παιδιών που έχει.**
- ✓ **Βαθμός του δένδρου είναι ο μέγιστος βαθμός των στοιχείων του.**



Δένδρα: Ορισμοί

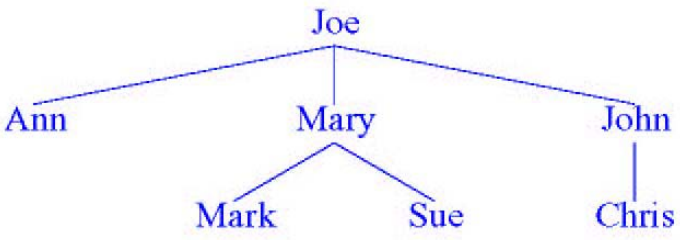
- Γράφημα ακυκλικό και συνεκτικό.
- Δέντρο με *n* κορυφές έχει $m = n - 1$ ακμές.
- **Ρίζα:** κόμβος χωρίς πρόγονο. Δέντρο με ρίζα : ιεραρχία.
- **Φύλλο:** κόμβος χωρίς απογόνους.
- **Πρόγονοι *t*:** κόμβοι στο (μοναδικό) μονοπάτι *t* προς ρίζα.
- **Απόγονοι *t*:** κόμβοι σε μονοπάτια από *t* προς φύλλα.
- **Υποδέντρο *t*:** Δέντρο αποτελούμενο από *t* και απογόνους του.
- **Επίπεδο *t*:** μήκος μονοπατιού από *t* προς ρίζα.
- **Ύψος:** μέγιστο επίπεδο κόμβου (φύλλου). Μέγιστη απόσταση από ρίζα.
- **Βαθμός *t* :** αριθμός παιδιών *t*.



Δένδρα: Βασικές Λειτουργίες

Γενικές πράξεις ADT (Abstract Data Type):

- element(*k*): επιστρέφει το στοιχείο του κόμβου *k*.
- father(*k*): επιστρέφει το δείκτη προς τον πατέρα του κόμβου *k*.
- children(*k*): επιστρέφει το δείκτη προς τα παιδιά του κόμβου *k*.



Δυαδικά Δένδρα

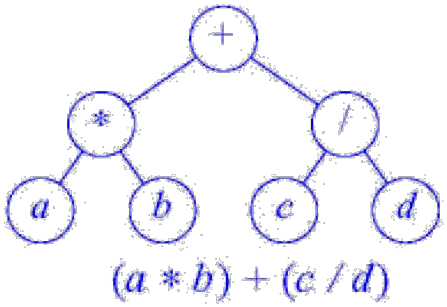
(αναδρομικός ορισμός)

Ένα δυαδικό δένδρο (binary tree, BT) t είναι μία πεπερασμένη (πιθανώς κενή) συλλογή στοιχείων. Όταν το δυαδικό δένδρο δεν είναι κενό, τότε έχει μία ρίζα και τα υπόλοιπα στοιχεία (αν υπάρχουν) επιμερίζονται σε δύο δυαδικά δένδρα που ονομάζονται το αριστερό και το δεξιό υποδένδρο του t.

Ιδιότητες:

- Το σχέδιο ενός δυαδικού δένδρου με n στοιχεία ($n > 0$) έχει ακριβώς n-1 ακμές.

- Ένα δυαδικό δένδρο ύψους h ($h \geq 0$) έχει τουλάχιστον h και το πολύ 2^h-1 στοιχεία.

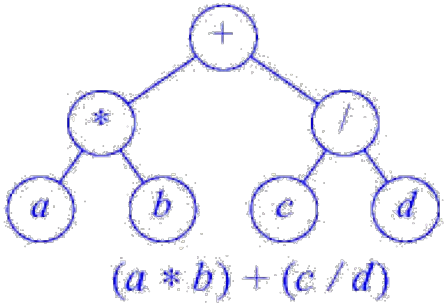


Δυαδικά Δένδρα: Πλήρες - Συμπληρωμένα

❑ **Πλήρες (full) δυαδικό δένδρο** ύψους *h* είναι εκείνο το δυαδικό δένδρο, το οποίο περιέχει ακριβώς 2^h-1 στοιχεία.

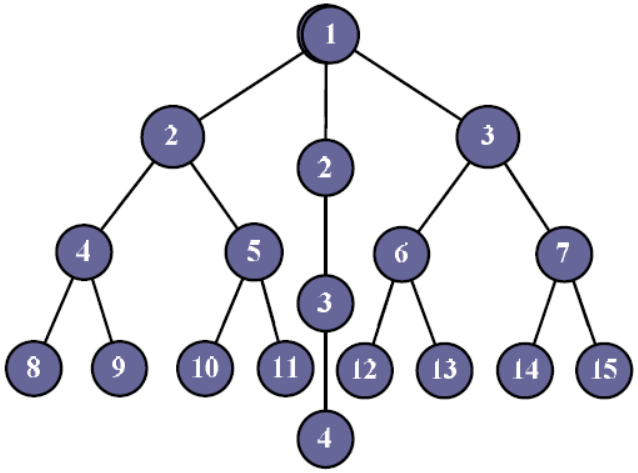
➢ Μπορούμε να αριθμήσουμε από 1 έως 2^h-1 τα στοιχεία ενός πλήρους δυαδικού δένδρου ύψους h, ξεκινώντας από το επίπεδο 1 προς το επίπεδο h και, μέσα σε κάθε επίπεδο, από αριστερά προς τα δεξιά.

❑ **Συμπληρωμένο (complete) δυαδικό δένδρο** ύψους *h* είναι εκείνο το δυαδικό δένδρο, το οποίο προκύπτει από ένα πλήρες δυαδικό δένδρο ύψους *h* εάν διαγράψουμε *k* στοιχεία με αρίθμηση 2^h-*i*, για $1 \le i \le k$ ($k \ge 0$).



Δυαδικά Δένδρα: Πλήρες - Συμπληρωμένα

- **Ύψος *h* :**
  $h+1 \leq$ #κορυφών $\leq 2^{h+1} - 1$
  - $h+1$ επίπεδα, ≥ 1 κορ. / επίπ
  - $\leq 2^i$ κορυφές στο επίπεδο i .
    $1 + 2 + \ldots + 2^h = 2^{h+1} - 1$
- **#κορυφών n :**
  $\log_2(n+1) - 1 \leq h \leq n - 1$
- **Πλήρες** (full) : $n = 2^{h+1} - 1$



Δυαδικά Δένδρα: Πλήρες - Συμπληρωμένα

- Το πλήθος των φύλλων σε ένα μη άδειο πλήρες δυαδικό δένδρο είναι κατά 1 μεγαλύτερο από το πλήθος των εσωτερικών κόμβων (2h-1 και 2h-1-1, αντίστοιχα)

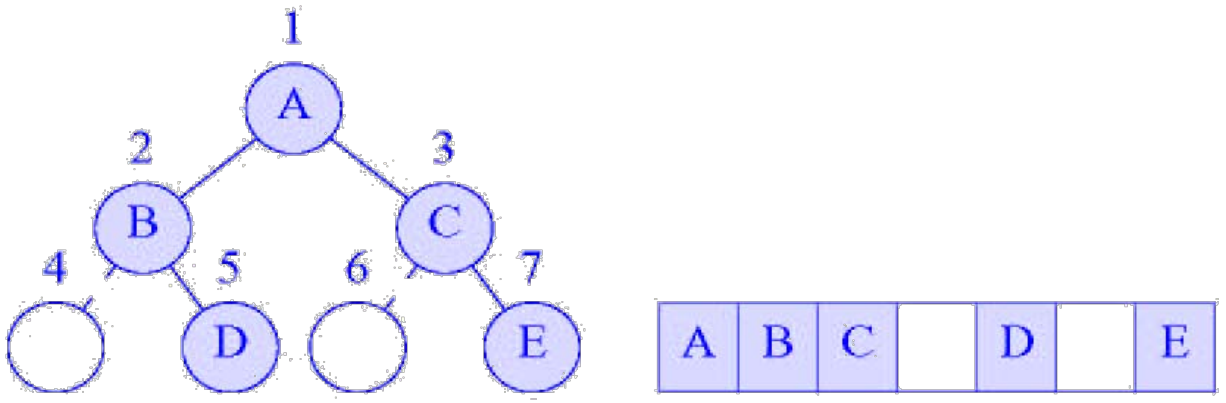
- **Ιδιότητες:** Έστω i ($1 \le i \le n$) ο αριθμός ενός στοιχείου ενός συμπληρωμένου δυαδικού δένδρου:
  - Αν i = 1, το στοιχείο είναι η ρίζα, αλλιώς είναι παιδί του κόμβου με αριθμό int[i/2].
  - Το αριστερό του παιδί έχει αριθμό $2i$ (αν $2i \le n$, αλλιώς δεν έχει αριστερό παιδί).
  - Το δεξιό του παιδί έχει αριθμό $2i+1$ (αν $2i+1 \le n$, αλλιώς δεν έχει δεξιό παιδί).

- Αποδείξεις των παραπάνω ιδιοτήτων με μαθηματική επαγωγή.

Δυαδικά Δένδρα: Αναπαράσταση με Πίνακες

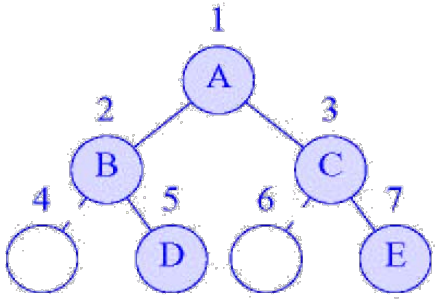
- Υλοποίηση με πίνακα. Βασίζεται στην προηγούμενη ιδιότητα (των συμπληρωμένων δυαδικών δένδρων)

- Πρόβλημα: σπατάλη χώρου όταν λείπουν πολλά στοιχεία (για να γίνει το δένδρο πλήρες)
  - Ένα δένδρο με *n* στοιχεία θα μπορούσε να απαιτήσει πίνακα μεγέθους μέχρι και 2^n-1 (η περίπτωση των δεξιών λοξών δυαδικών δένδρων)



Δυαδικά Δένδρα: Στατική Αναπαράσταση

Παράδειγμα:

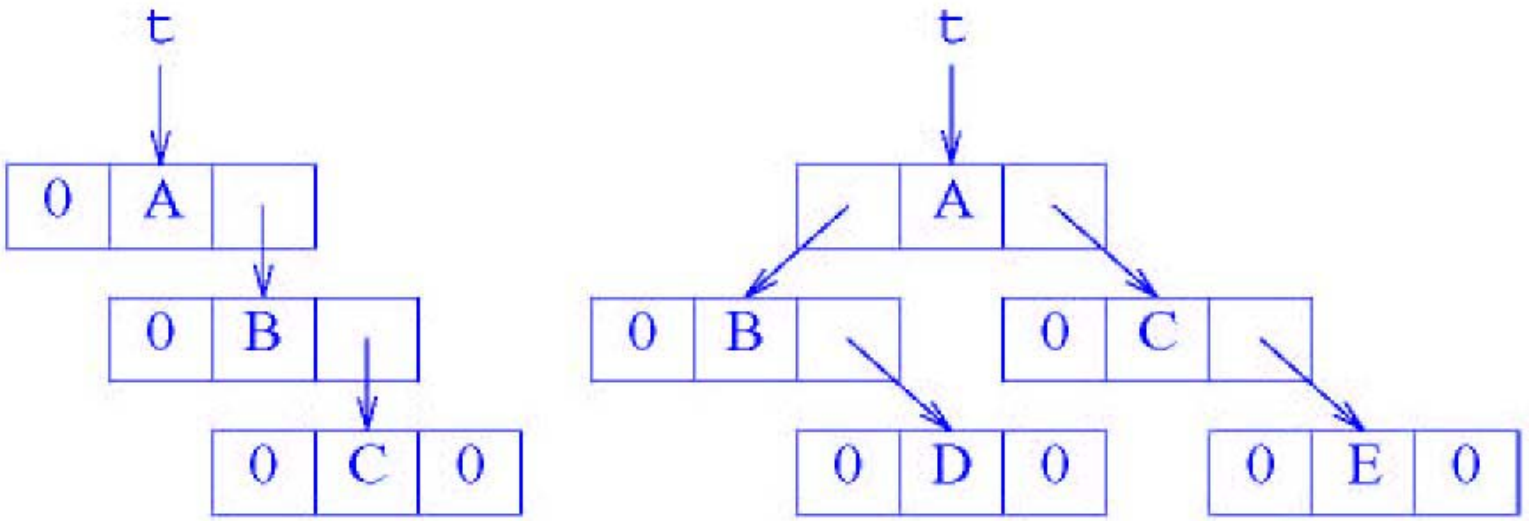


A	B	C		D		E

Θέση	1	2	3	4	5	6	7
Γονιός	--	1	1	2	2	3	3
Αριστερό παιδί	2	4	6	--	--	--	--
Δεξί παιδί	3	5	7	--	--	--	--
Αριστερός αδελφός	--	--	2	--	4	--	6
Δεξιός αδελφός	--	3	--	5	--	7	--

Δυαδικά Δένδρα: Συνδεδεμένη Αναπαράσταση

- ✓ Υλοποίηση με δείκτες: Η πιο συνηθισμένη υλοποίηση.
- ✓ Κάθε στοιχείο αντιστοιχεί σε ένα κόμβο ο οποίος περιλαμβάνει ένα πεδίο δεδομένων (data) και δύο πεδία συνδέσμων (LeftChild, RightChild)



Παράδειγμα: Κλάση Δυαδικών Δένδρων

```
class BinaryTreeNode {
    public:
        BinaryTreeNode() {LeftChild = RightChild = 0;}
        BinaryTreeNode(const T& e)
                {data = e; LeftChild = RightChild = 0;}
        BinaryTreeNode(const T& e, BinaryTreeNode *l,
                BinaryTreeNode *r)
        data = e; LeftChild = l; RightChild = r;}
    private:
        T data;
        BinaryTreeNode<T> *LeftChild,  // left subtree
                *RightChild; // right subtree
}
```

Δυαδικά Δένδρα (BT): Λειτουργίες

- **Συνήθεις πράξεις BT:**
  - **GetHeight():** Προσδιορισμός ύψους δένδρου.
  - **GetNumberElements():** Προσδιορισμός στοιχείων δένδρου.
  - **CopyTree():** Αντιγραφή δένδρου.
  - **DeleteTree():** Διαγραφή δένδρου.
  - **FindifTree():** Έλεγχος ΔΔ αν είναι BT.
  - **OutputTree():** Παρουσίαση δένδρου (οθόνη ή printer).

- Όλα τα παραπάνω εκτελούνται με συστηματικό τρόπο με τη λειτουργία διάσχισης του δένδρου. Κάθε διαδικασία επίσκεψης όλων των κόμβων ενός δένδρου, ακριβώς μια φορά τον καθένα, ονομάζεται **διάσχιση (traversal).**

Δυαδικά Δένδρα: Διάσχιση

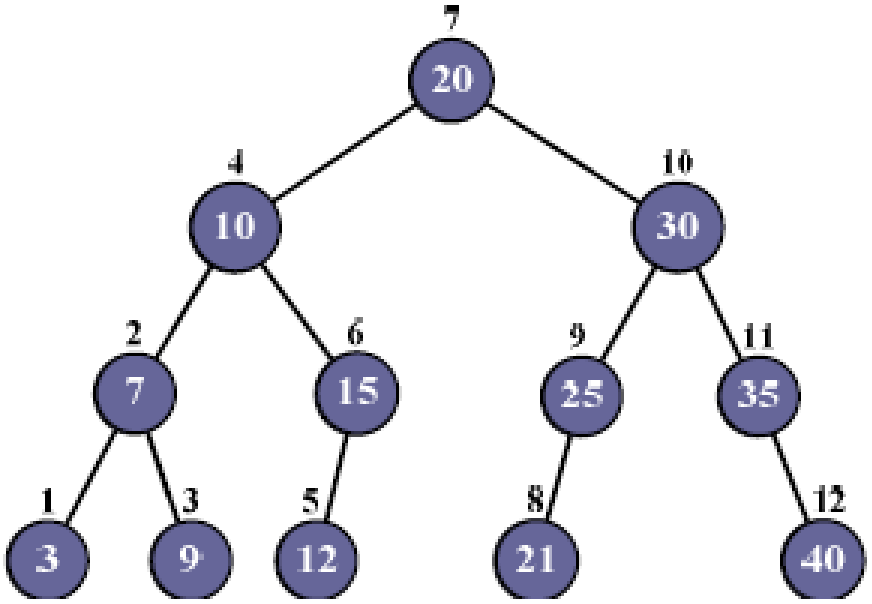
- Διάσχιση:
  - **Προδιατεταγμένη** διάσχιση (**preorder**): Για κάθε κόμβο, επισκεπτόμαστε πρώτα τον ίδιο τον κόμβο, έπειτα τους κόμβους του αριστερού του υποδένδρου και στη συνέχεια τους κόμβους του δεξιού του υποδένδρου.
  - **Μεταδιατεταγμένη** διάσχιση (**postorder**): Για κάθε κόμβο, επισκεπτόμαστε πρώτα τους κόμβους του αριστερού του υποδένδρου, έπειτα τους κόμβους του δεξιού του υποδένδρου και στη συνέχεια τον ίδιο τον κόμβο.
  - **Ενδοδιατεταγμένη** διάσχιση (**inorder**): Για κάθε κόμβο, επισκεπτόμαστε πρώτα τους κόμβους του αριστερού του υποδένδρου, έπειτα τον ίδιο τον κόμβο και στη συνέχεια τους κόμβους του δεξιού του υποδένδρου.
  - **Κατά σειρά επιπέδων** (**level order**): Επισκεπτόμαστε τους κόμβους κατά επίπεδα, από πάνω (τη ρίζα) προς τα κάτω, και μέσα σε ένα επίπεδο από αριστερά προς τα δεξιά.

- Οι 3 πρώτες μέθοδοι είναι αναδρομικές ενώ η $4^{η}$ είναι επαναληπτική

Διάσχιση: InOrder

- Ενδο-διατεταγμένη (inorder) διέλευση:
  - Αριστερό – Ρίζα – Δεξί.
  - Κόμβος εξετάζεται μετά από κόμβους αριστερού υποδέντρου και πριν από κόμβους δεξιού υποδέντρου.

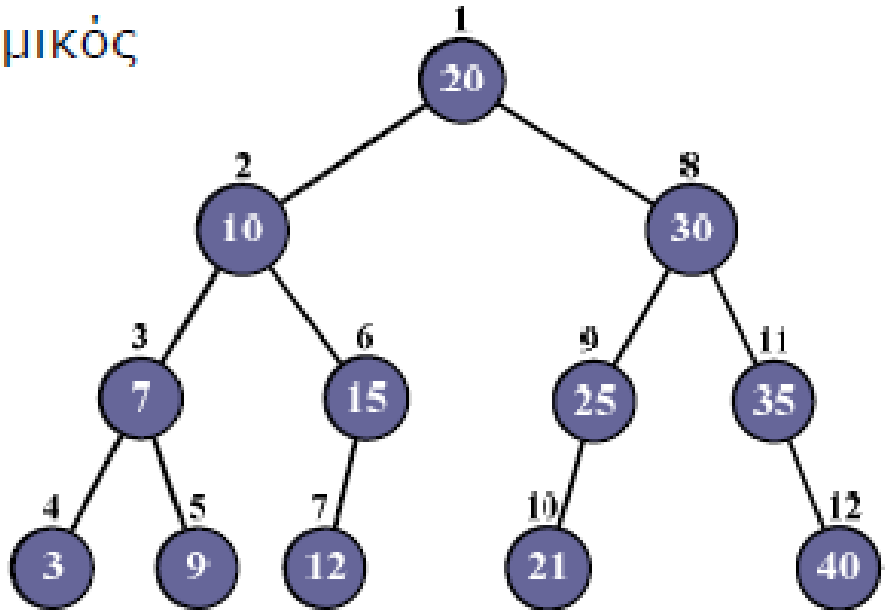
```
void inOrder(tnode *x) {
    if (x == NULL) return;
    inOrder(x->left);
    printf(" %d ", x->key);
    inOrder(x->right);   }

inOrder(root);
```



Διάσχιση: preOrder

- Προ-διατεταγμένη (preorder) διέλευση:
  - Ρίζα – Αριστερό – Δεξί.
  - Κόμβος εξετάζεται πριν από κόμβους αριστερού και δεξιού υποδέντρου.
  - Χρόνος : Θ(n) – γραμμικός

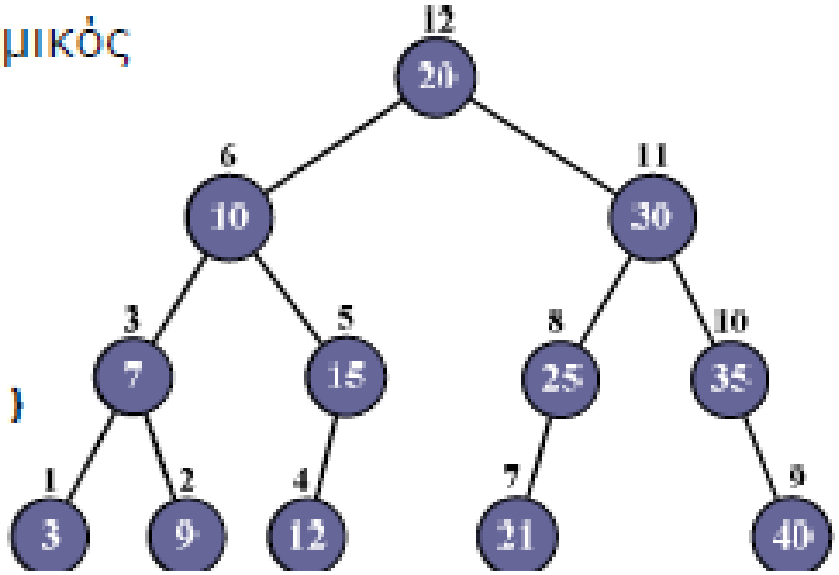
```
void preOrder(tnode *x) {
     if (x == NULL) return;
     printf(" %d ", x->key);
     preOrder(x->left);
     preOrder(x->right);   }

preOrder(root);
```



Διάσχιση: postOrder

- Μετά-διατεταγμένη (postOrder) διέλευση:
  - Αριστερό – Δεξί – Ρίζα.
  - Κόμβος εξετάζεται μετά από κόμβους αριστερού και δεξιού υποδέντρου.
    - Χρόνος : Θ(n) – γραμμικός

```
void postOrder(tnode *x) {
    if (x == NULL) return;
    postOrder(x->left);
    postOrder(x->right);
    printf(" %d ", x->key); }

postOrder(root);
```



Διάσχιση: Παράδειγμα

- **Προδιατεταγμένη:**

  A, B, D, C, E, G, F, H, I

- **Μεταδιατεταγμένη:**

  D, B, G, E, H, I, F, C, A

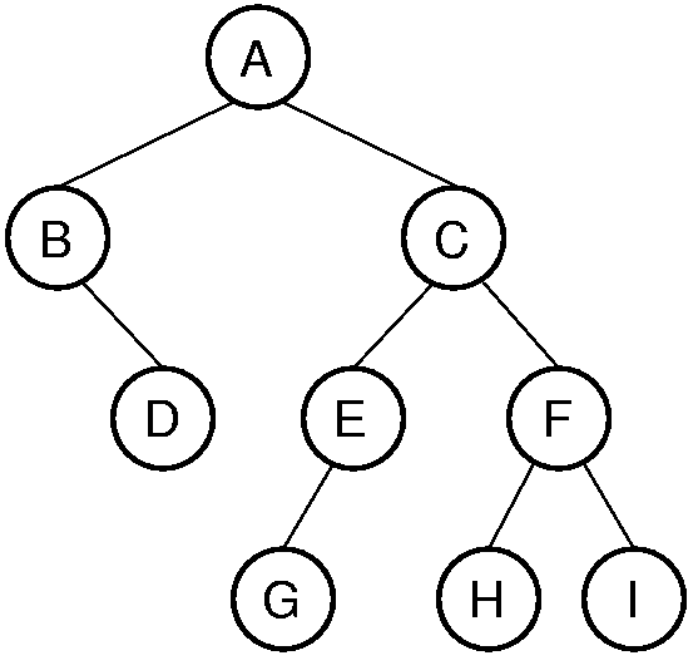
- **Ενδοδιατεταγμένη:**

  B, D, A, G, E, C, H, F, I

- **Κατά σειρά επιπέδων:**

  A, B, C, D, E, F, G, H, I

  – δένδρων)



Διάσχιση: Παραδείγματα Συναρτήσεων

```
void PreOrder(BinaryTreeNode<T> *t)
{// Preorder traversal of *t.
  if (t) {Visit(t); PreOrder(t->LeftChild);
    PreOrder(t->RightChild); }
}

void InOrder(BinaryTreeNode<T> *t)
{// Inorder traversal of *t.
  if (t) {InOrder(t->LeftChild);
    Visit(t); InOrder(t->RightChild); }
}

void PostOrder(BinaryTreeNode<T> *t)
{// Postorder traversal of *t.
  if (t) {PostOrder(t->LeftChild);
    PostOrder(t->RightChild); Visit(t); }
}
```

Διάσχιση: Παραδείγματα Συναρτήσεων

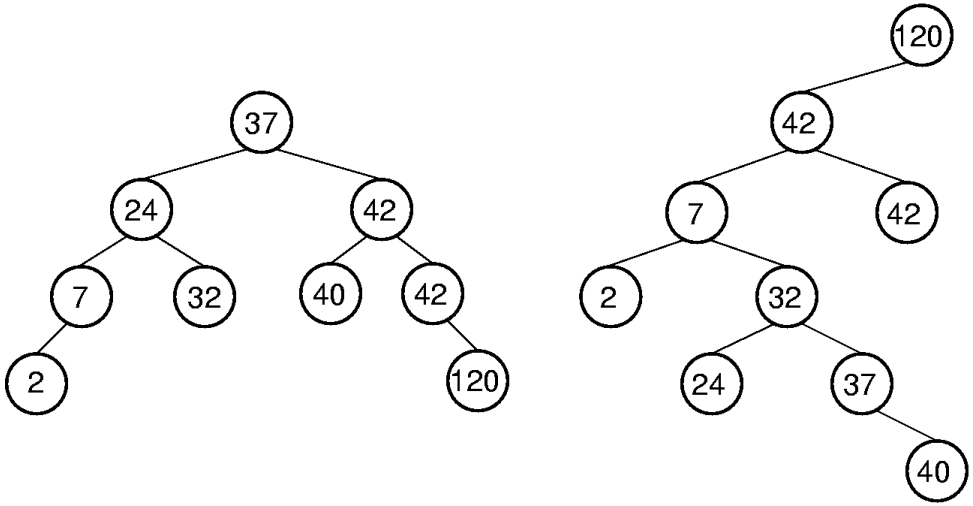
```
void LevelOrder(BinaryTreeNode<T> *t)
{// Level-order traversal of *t.
  LinkedQueue<BinaryTreeNode<T>*> Q;
  while (t) {
    Visit(t);   // visit t

    // put t's children on queue
    if (t->LeftChild) Q.Add(t->LeftChild);
    if (t->RightChild) Q.Add(t->RightChild);

    // get next node to visit
    try {Q.Delete(t);}
    catch (OutOfBounds) {return;}
  }
}
```

Δυαδικά Δένδρα Αναζήτησης (ΔΔΑ)

Ορισμός: Ένα ΔΔΑ (Binary Search Tree, BST) είναι δυαδικό δένδρο με διακριτά κλειδιά (τιμές) και τις εξής ιδιότητες:

- Τα κλειδιά (αν υπάρχουν) στο αριστερό υποδένδρο της ρίζας είναι μικρότερα από το κλειδί της ρίζας.
- Τα κλειδιά (αν υπάρχουν) στο δεξιό υποδένδρο της ρίζας είναι μεγαλύτερα από το κλειδί της ρίζας.
- Το αριστερό και το δεξιό υποδένδρο είναι επίσης ΔΔΑ.

Στόχος: **να μειώσουμε τους χρόνους ενημέρωσης και αναζήτησης σε λιγότερο από Θ(n).**



ΔΔΑ: Παράδειγμα Κλάσης

AbstractDataType *BSTree* {

instances: binary trees, each node has an element with a key field; all keys are distinct; keys in the left subtree of any node smaller than the key in the node; those in the right subtree are larger;

operations

Create (): create an empty binary search tree

Search (*k*, *e*): return in *e* the element with key *k*; return false if the operation fails, return true if it succeeds

Insert (*e*): insert element *e* into the search tree

Delete (*k*, *e*): delete the element with key *k* and return it in *e*

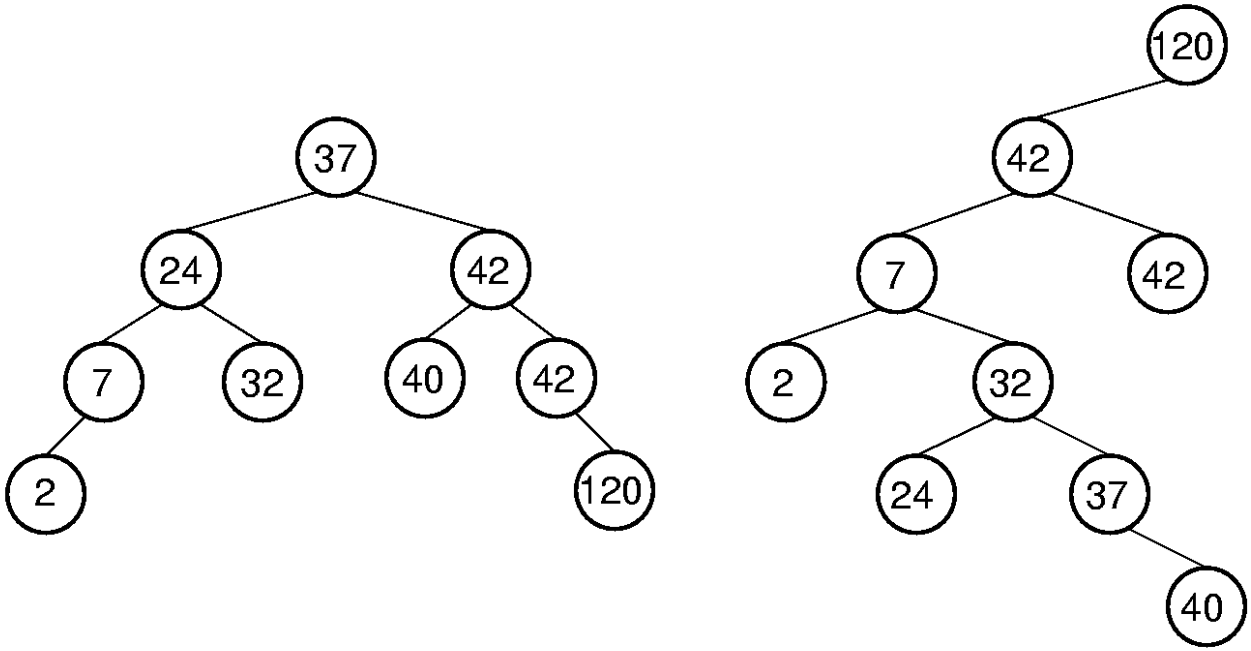
Ascend (): Output all elements in ascending order of key

}

ΔΔΑ: Ύψος

- **Το ύψος ενός ΔΔΑ με n στοιχεία μπορεί να φτάσει μέχρι και *n*.**

- **Στη γενική περίπτωση όμως (όταν οι εισαγωγές και οι διαγραφές γίνονται τυχαία), το ύψος είναι O(log*n*).**



ΔΔΑ: Αναζήτηση Στοιχείου

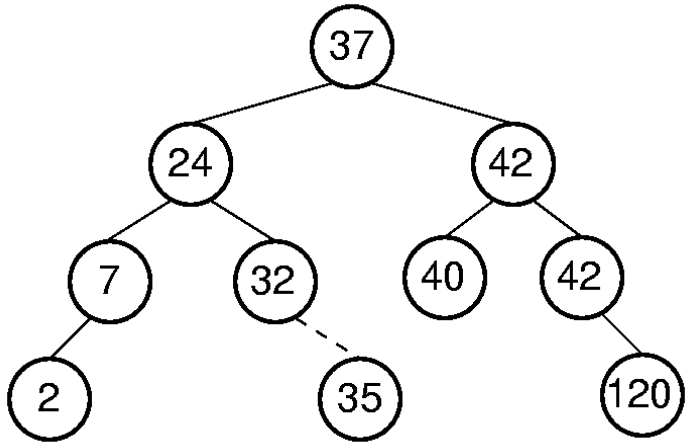
```
bool BSTree<E,K>::Search(const K& k, E &e) const
{// Search for element that matches k.
  // pointer p starts at the root and moves through
  // the tree looking for an element with key k
  BinaryTreeNode<E> *p = root;
  while (p)                  // examine p->data
    if (k < p->data) p = p->LeftChild;
    else if (k > p->data) p = p->RightChild;
        else {               // found element
            e = p->data;
            return true;}
  return false;
}
```

Κόστος αναζήτησης = κόστος κατάβασης = O(h)

ΔΔΑ: Εισαγωγή Στοιχείου

Βασική ιδιότητα ΔΔΑ:
- Η εισαγωγή γίνεται πάντα σε κάποιο (νέο) φύλλο

Διαδικασία:
- Αναζήτηση του στοιχείου (οπότε καταλήγουμε σε κόμβο-φύλλο).
- Εισαγωγή του ως παιδί εκείνου του κόμβου.



Κόστος = Κόστος αναζήτησης + κόστος 'συγκόλλησης'
νέου κόμβου στον πατέρα-κόμβο = O(h) + O(1) = O(h)

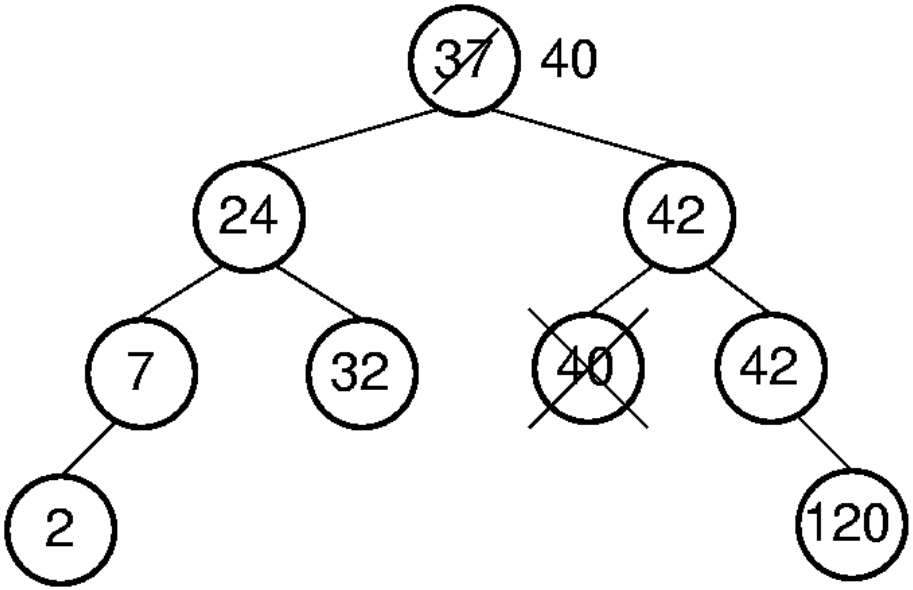
ΔΔΑ: Εισαγωγή Στοιχείου

```
BSTree<E,K>& BSTree<E,K>::Insert(const E& e)
{// Insert e if not duplicate.
  BinaryTreeNode<E> *p = root,          // search pointer
                    *pp = 0;            // parent of p
  // find place to insert
  while (p) {                           // examine p->data
    pp = p;
    // move p to a child
    if (e < p->data) p = p->LeftChild;
    else if (e > p->data) p = p->RightChild;
        else throw BadInput(); // duplicate
    }
  // get a node for e and attach to pp
  BinaryTreeNode<E> *r = new BinaryTreeNode<E> (e);
  if (root) {// tree not empty
    if (e < pp->data) pp->LeftChild = r;
    else pp->RightChild = r;}
  else // insertion into empty tree
      root = r;

  return *this;
}
```

ΔΔΑ: Διαγραφή Στοιχείου

Διαφορετικές περιπτώσεις:

1. Ο κόμβος p (που περιέχει το στοιχείο) είναι φύλλο.
2. Το p έχει μόνο ένα μη κενό υποδένδρο.
3. Το p έχει ακριβώς δύο μη κενά υποδένδρα.



ΔΔΑ: Διαγραφή Στοιχείου

```
BSTree<E,K>& BSTree<E,K>::Delete(const K& k, E& e)
{// Delete element with key k and put it in e.

  // set p to point to node with key k
  BinaryTreeNode<E> *p = root, // search pointer
                    *pp = 0;   // parent of p
  while (p && p->data != k){// move to a child of p
    pp = p;
    if (k < p->data) p = p->LeftChild;
    else p = p->RightChild;
    }
   if (!p) throw BadInput(); // no element with key k

   e = p->data;  // save element to delete

  // restructure tree,  handle case when p has two children
  if (p->LeftChild && p->RightChild) {// two children
    // convert to zero or one child case
    // find largest element in left subtree of p
    BinaryTreeNode<E> *s = p->LeftChild,
                      *ps = p;  // parent of s
    ...
```

ΔΔΑ: Διαγραφή Στοιχείου

```
  ...
  while (s->RightChild) {// move to larger element
      ps = s;
      s = s->RightChild;}

  // move largest from s to p
  p->data = s->data;
                        p = s;
                        pp = ps;}
  // p has at most one child
  // save child pointer in c
  BinaryTreeNode<E> *c;
  if (p->LeftChild) c = p->LeftChild;
  else c = p->RightChild;

  // delete p
  if (p == root) root = c;
  else {// is p left or right child of pp?
      if (p == pp->LeftChild)
          pp->LeftChild = c;
      else pp->RightChild = c;}
  delete p;

  return *this;
}
```

Δένδρα AVL

Δένδρα AVL **(**Adelson, Velskii, Landis, 1962)

Ορισμός: Ένα δυαδικό δένδρο καλείται δένδρο AVL αν και μόνο αν τα ύψη των δύο (αριστερού και δεξιού) κάθε εσωτερικού κόμβου διαφέρουν το πολύ κατά 1.

$$|h_L - h_R| \leq 1$$

Βασική Ιδιότητα:

- **Κάθε πράξη** (διάσχιση, ένθεση, διαγραφή, …) εκτελείται σε χρόνο (μέση περίπτωση): $O(\log n)$.
- Απόδειξη: χρησιμοποιώντας ιδιότητες των αριθμών Fibonacci.

Δένδρα AVL

Τρόπος Λειτουργίας

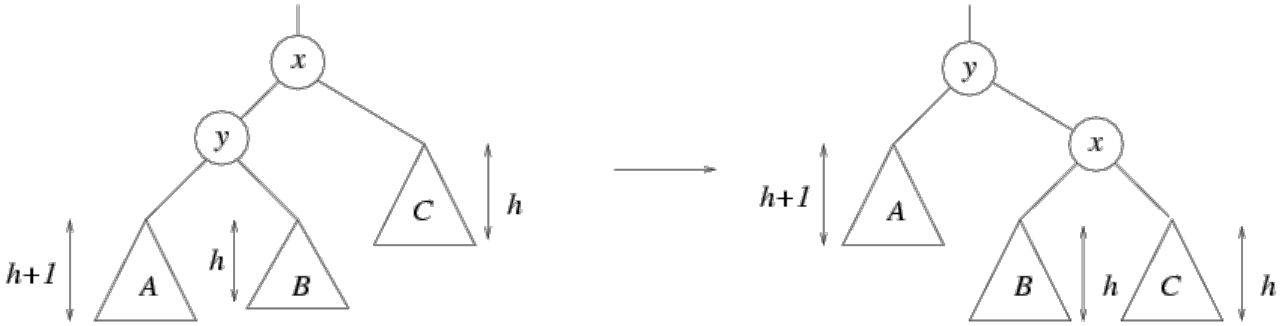
- Η αναζήτηση στοιχείου-κόμβου είναι παρόμοια όπως και σε ένα απλό δυαδικό δένδρο αναζήτησης.
- Μετά από κάθε εισαγωγή και διαγραφή πρέπει να γίνει έλεγχος για το αν ικανοποιείται ο περιορισμός του AVL (διαφορά των υψών των υποδένδρων).
- Σε περίπτωση που δεν υπάρχει πρόβλημα στη δομή του δένδρου, τότε δεν απαιτείται καμία άλλη ενέργεια.
- Διαφορετικά θα πρέπει να γίνουν δομικές αλλαγές στο δένδρο ώστε να προκύψει πάλι ένα δένδρο AVL.

Δομικές αλλαγές – περιστροφές: Έχουμε δύο είδη περιστροφών

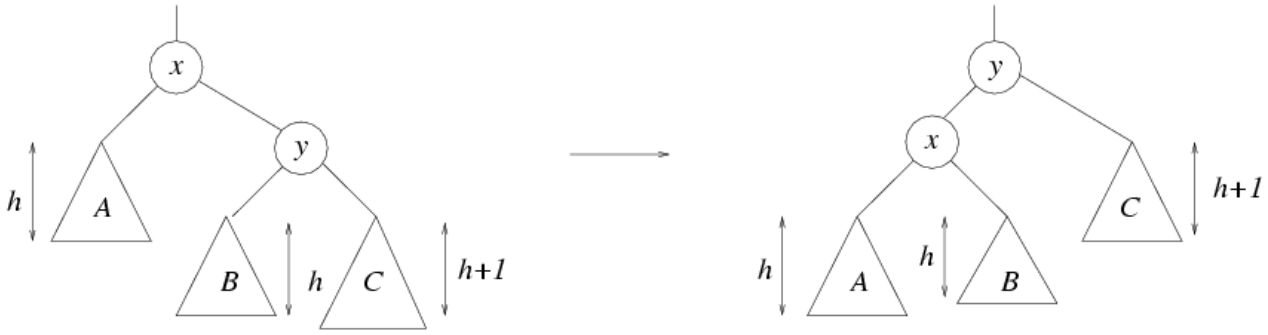
- **Απλή περιστροφή:** Περιστρέφονται δύο κόμβοι.
- **Διπλή περιστροφή:** Περιστρέφονται τρεις κόμβοι.

Δένδρα AVL: Απλή Περιστροφή

Περίπτωση 1:

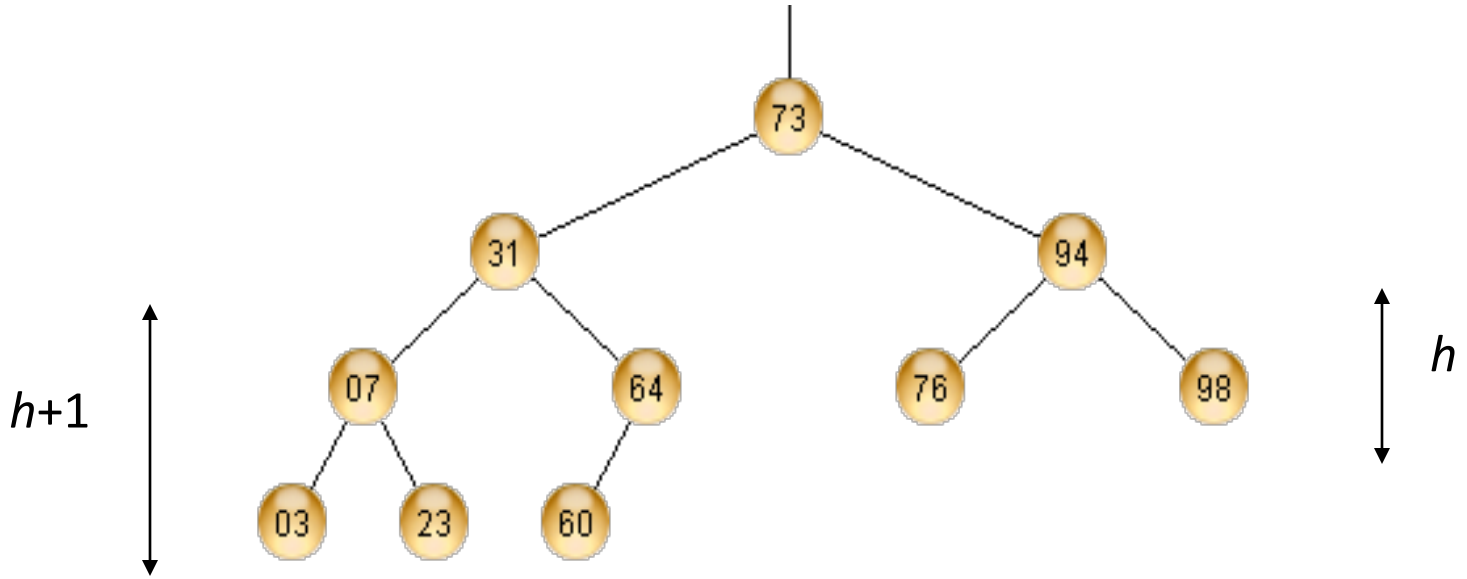


Περίπτωση 2:



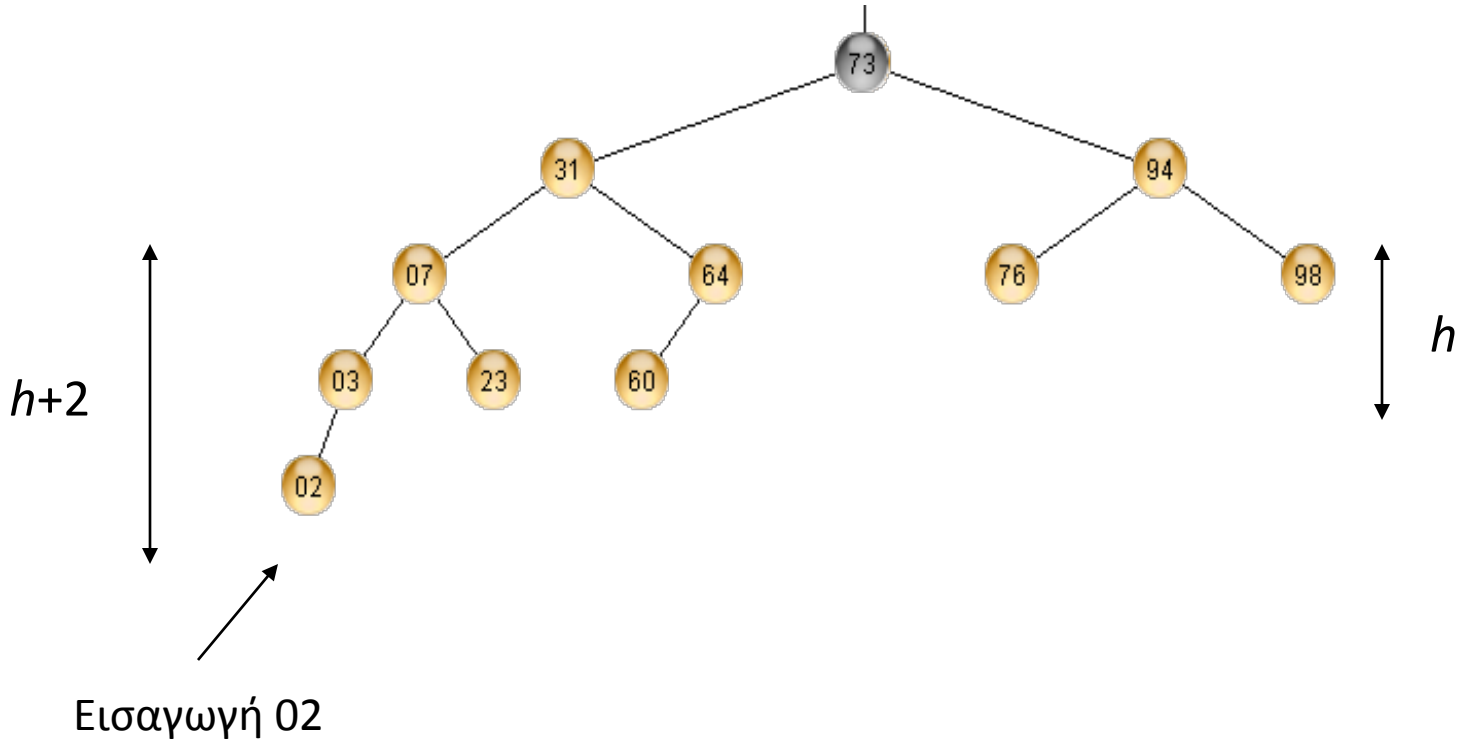
Δένδρα AVL: Εισαγωγή - Απλή Περιστροφή

Παράδειγμα: Αρχικό Δένδρο AVL



Δένδρα AVL: Εισαγωγή - Απλή Περιστροφή

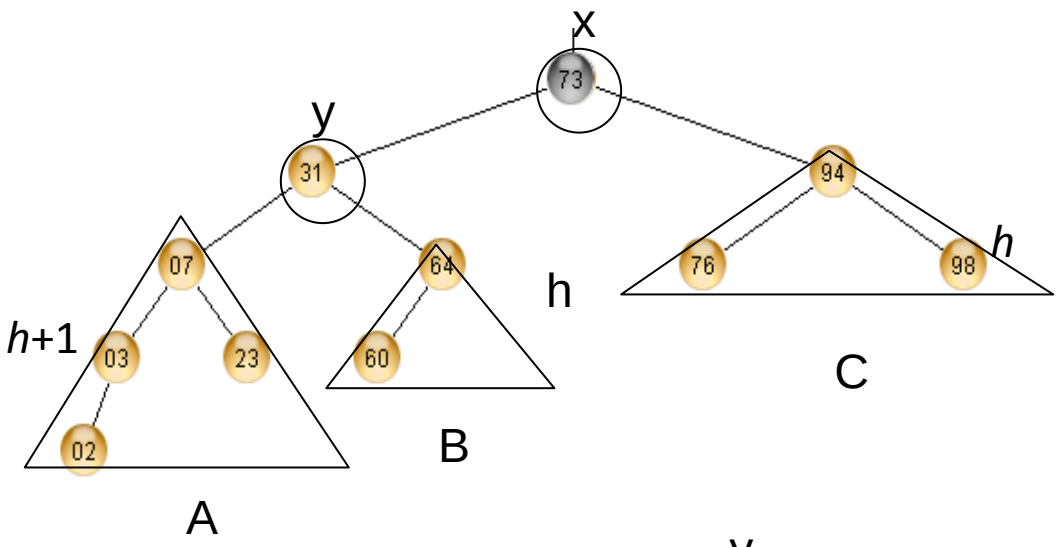
Εισαγωγή νέου στοιχείου - κόμβου



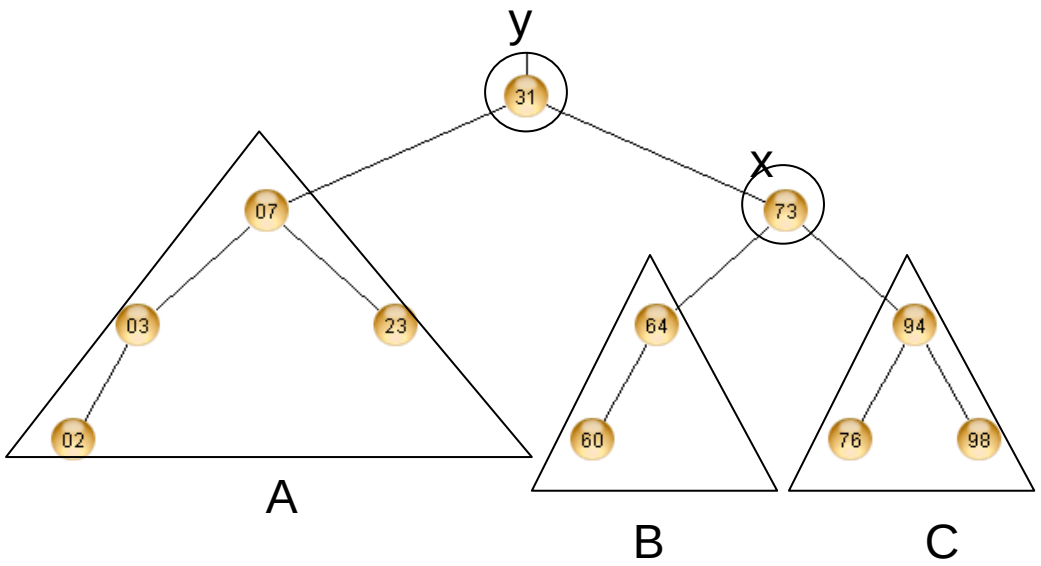
Μετά την εισαγωγή του κόμβου 2 το δένδρο που προκύπτει ΔΕΝ είναι AVL.

Δένδρα AVL: Εισαγωγή - Απλή Περιστροφή

Περιστροφή:

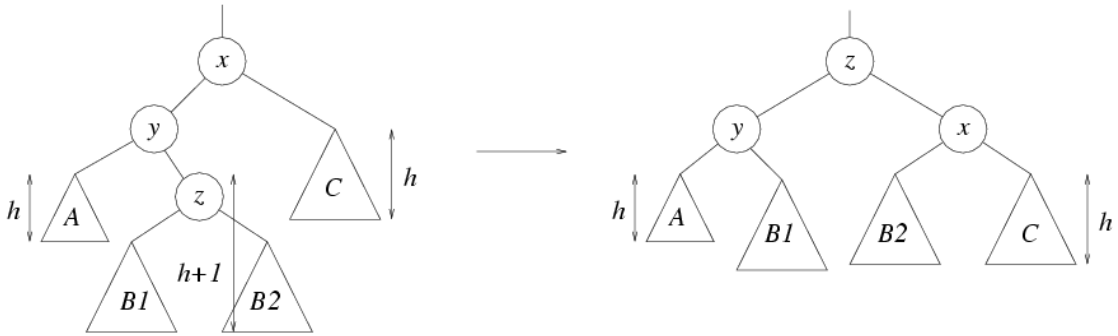


Μετά την Περιστροφή:

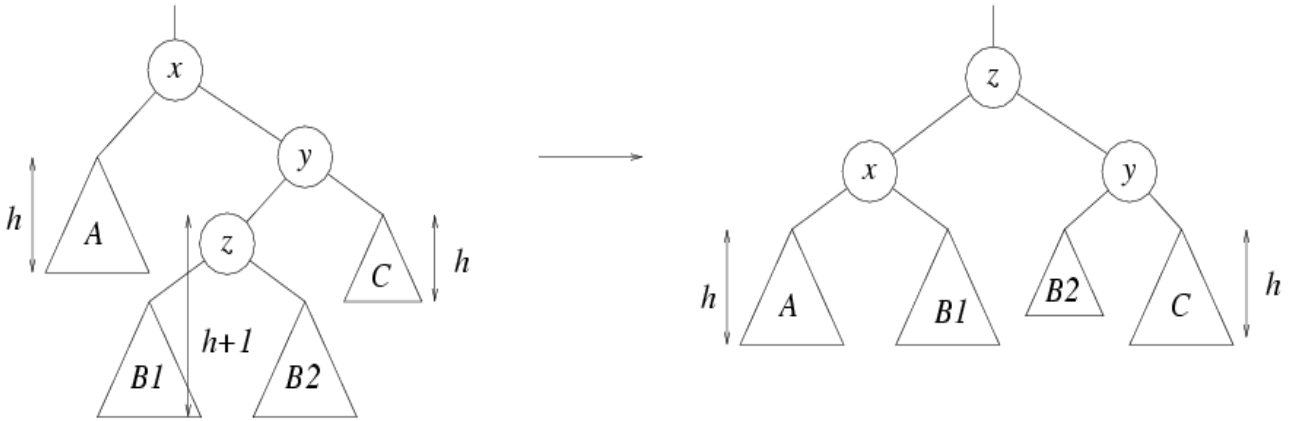


Δένδρα AVL: Διπλή Περιστροφή

Περίπτωση 1:

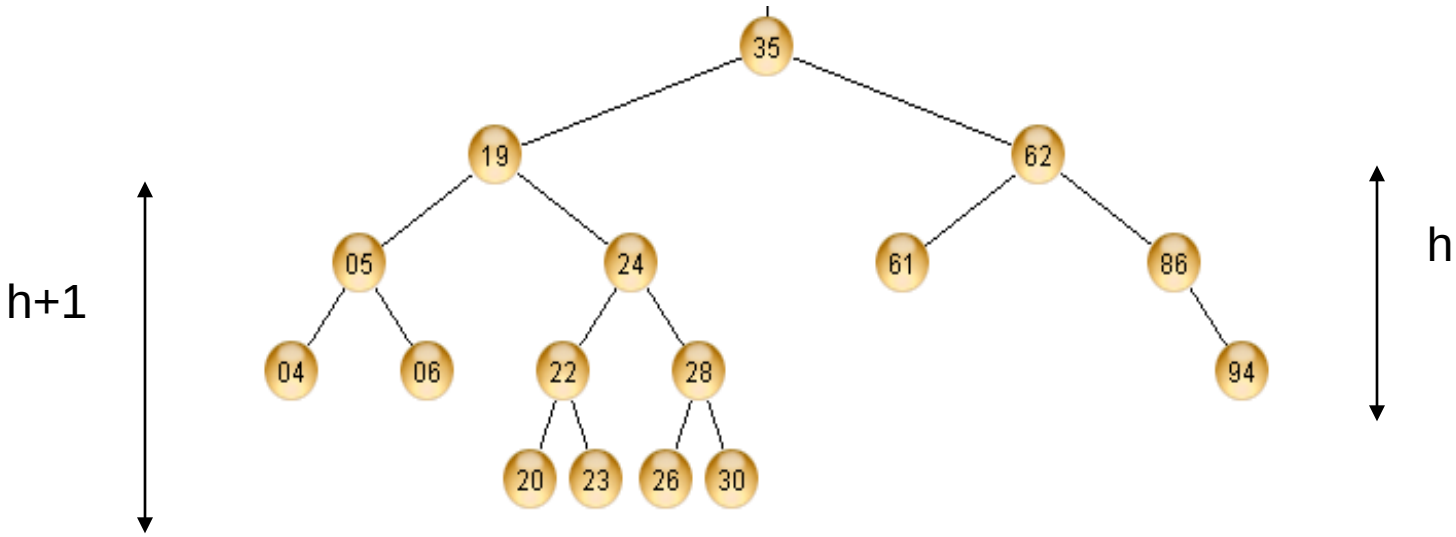


Περίπτωση 2:



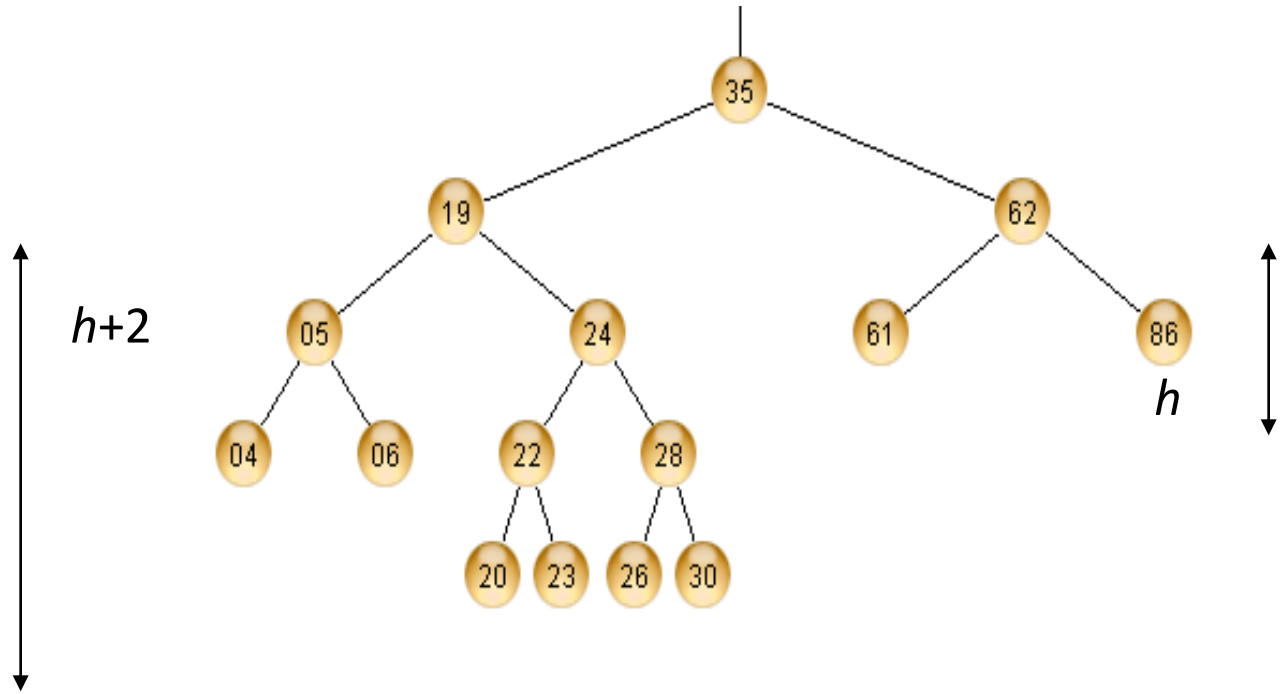
Δένδρα AVL: Διαγραφή - Διπλή Περιστροφή

Παράδειγμα: Διαγραφή του κόμβου 94.



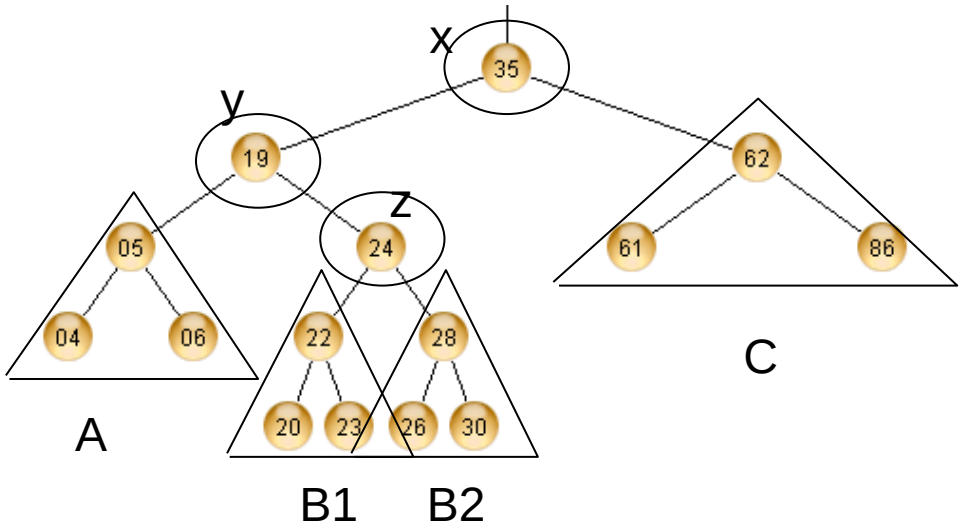
Δένδρα AVL: Διαγραφή - Διπλή Περιστροφή

Δεν είναι πλέον δένδρο AVL

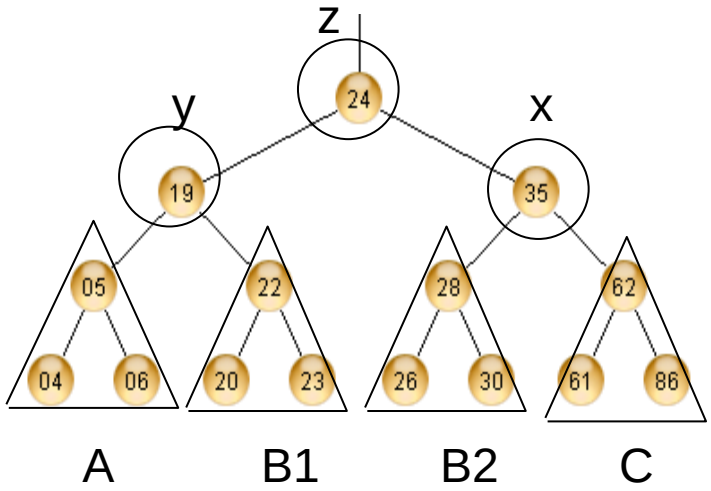


Δένδρα AVL: Διαγραφή - Διπλή Περιστροφή

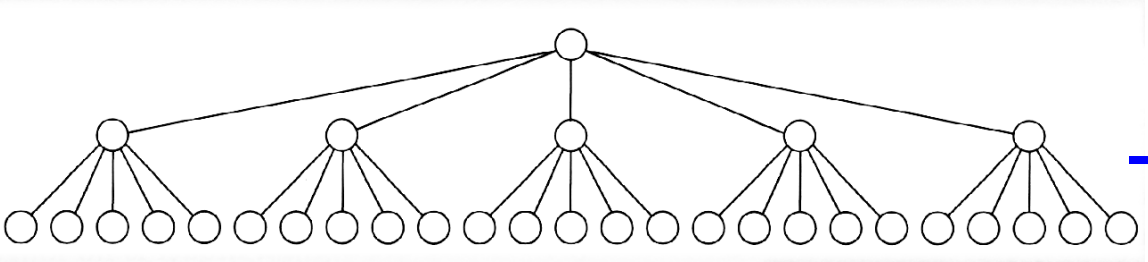
Περιστροφή:



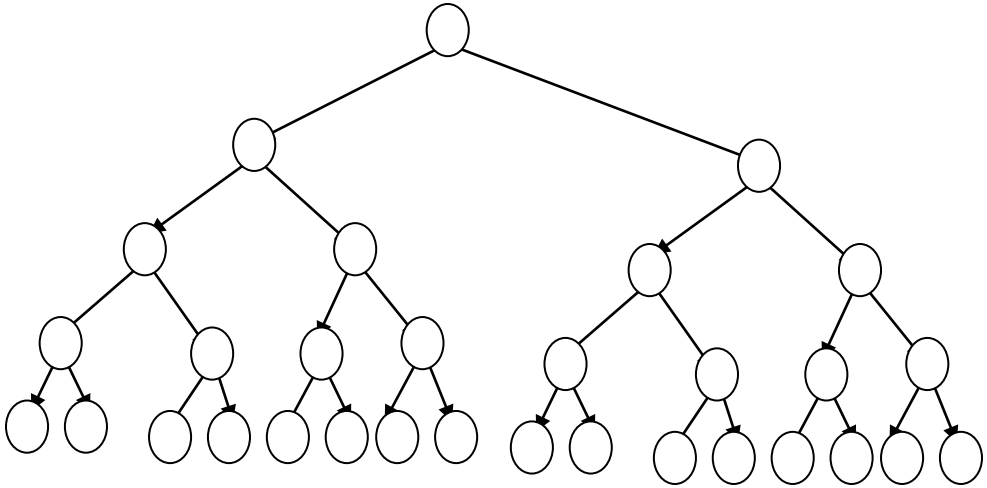
Μετά την Περιστροφή:



Δένδρα Πολλών Δρόμων



Δένδρο 5 δρόμων
31 κόμβοι
3 επίπεδα



Δυαδικό δένδρο
31 κόμβοι
5 επίπεδα

Βιβλιογραφία

- *Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++*, S. Sahni, Εκδόσεις Τζιόλα, 2004.
- *Δομές Δεδομένων*, Π. Μποζάνης, Εκδόσεις Τζιόλα 2006.
- Δομές Δεδομένων, Έννοιες, Τεχνικές και Αλγόριθμοι, Γ.Φ. Γεωργακόπουλος., Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2002.
- *Αλγόριθμοι σε C, Μέρη 1-4 (Θεμελιώδεις Έννοιες, Δομές Δεδομένων, Ταξινόμηση, Αναζήτηση),* Robert Sedgewick, Τρίτη Αμερικάνικη Έκδοση, Εκδόσεις Κλειδάριθμός 2005.
- *Εισαγωγή στους αλγορίθμους Τόμος Ι*, T.H. Cormen, C.E. Leiserson, R.L. Rivest, Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2006.