# Διάλεξη 16: Εισαγωγή σε Δενδρικές Δομές Δεδομένων

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

*Εισαγωγή σε δενδρικές δομές δεδομένων,*

*Ορισμοί και πράξεις*

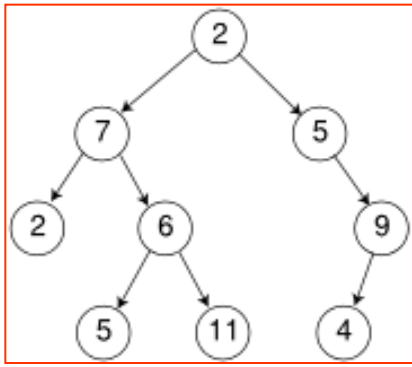
*Αναπαράσταση δενδρικών δομών δεδομένων στη μνήμη*

*Διάσχιση Δέντρων*

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

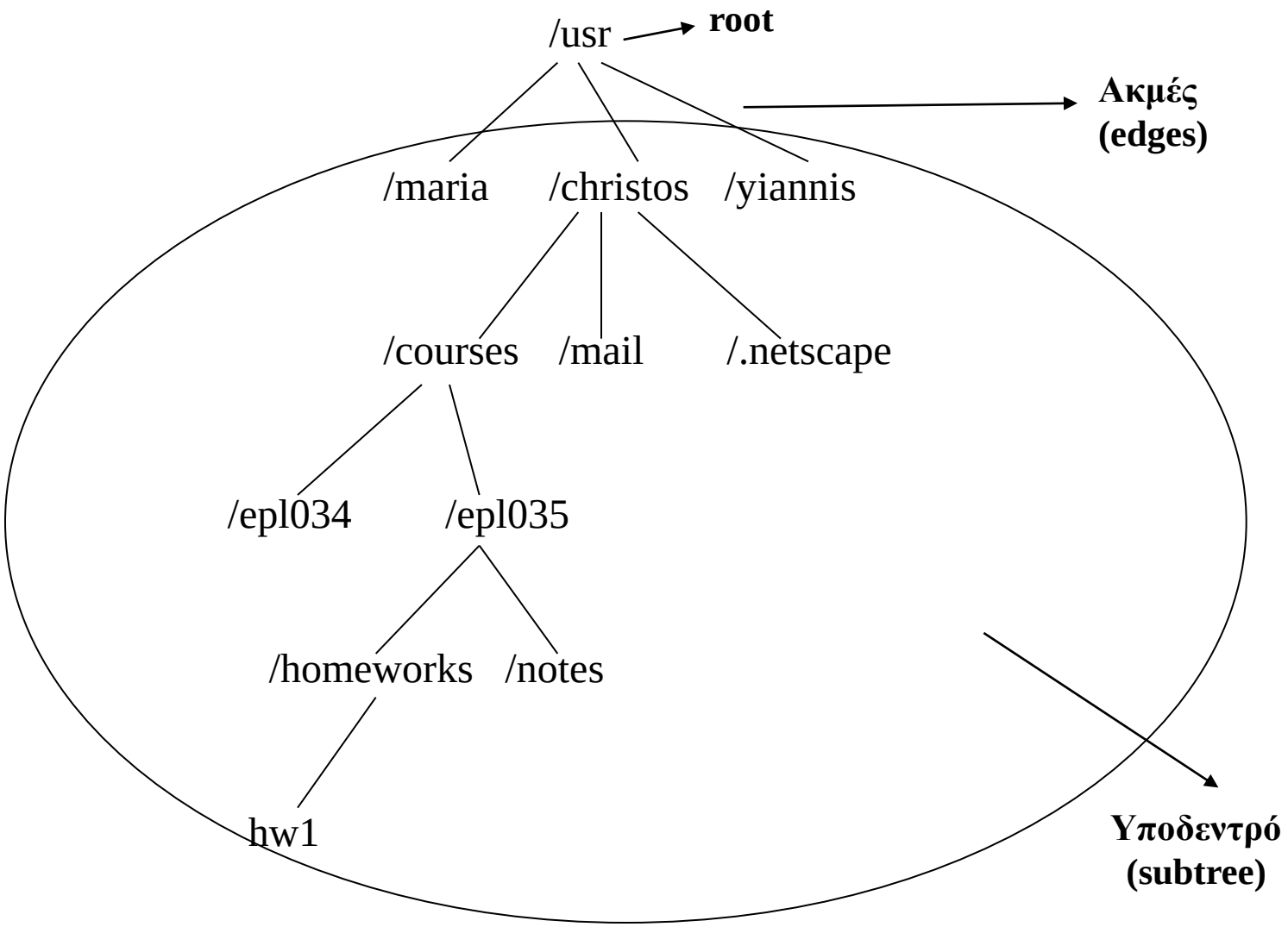
# Δένδρα

- Μέχρι τώρα μελετούσαμε *γραμμικές* δομές δεδομένων (linear data structures) π.χ. Stacks, queues, arrays, lists, κτλ
- Σε αυτή την ενότητα θα μελετήσουμε *δενδρικές* δομές δεδομένων (δηλαδή *μη-γραμμικές* δομές)
- ***Δένδρο*** είναι ένα σύνολο **κόμβων (κορυφών)** που συνδέονται από **ακμές (ή τόξα)** με τον ακόλουθο αναδρομικό ορισμό:
  1. Ένα δένδρο είναι είτε κενό ή
  2. αποτελείται από:
     - ***μια ρίζα* (root),** δηλαδή ένα κόμβο στον οποίο δεν καταλήγουν αλλά μόνο ξεκινούν ακμές, και
     - ***0 ή περισσότερα υποδένδρα*** $T_1, T_2, \ldots, T_k$, το καθένα ξεχωριστό από τα άλλα και από τη ρίζα. Υπάρχει μια ακμή από τη ρίζα στις ρίζες των $T_1, T_2, \ldots, T_k$.




- Από κάθε κόμβο ξεκινούν 0 ή περισσότερες **ακμές (edges)**.
- Σε κάθε κόμβο καταλήγει μία μόνο ακμή (αλλιώς θα είχαμε μια άλλη δομή – τον γράφο)
- Στη ρίζα δεν καταλήγει καμιά ακμή.

# Παράδειγμα 1: Οργάνωση Μονοπατιών σε ένα OS



# Παράδειγμα συνέχεια

Υποθέστε ότι χρησιμοποιούσαμε μια **γραμμική δομή** (π.χ. ένα πίνακα) για την αποθήκευση των διάφορων μονοπατιών

/usr/maria

/usr/christos

/usr/christos/courses

/usr/christos/courses/epl034

/usr/christos/courses/epl035

/usr/christos/courses/epl035/homeworks

/usr/christos/courses/epl035/homeworks/hw1

/usr /christos/courses/epl035/notes

/usr /christos/mail

/usr /christos/.netscape

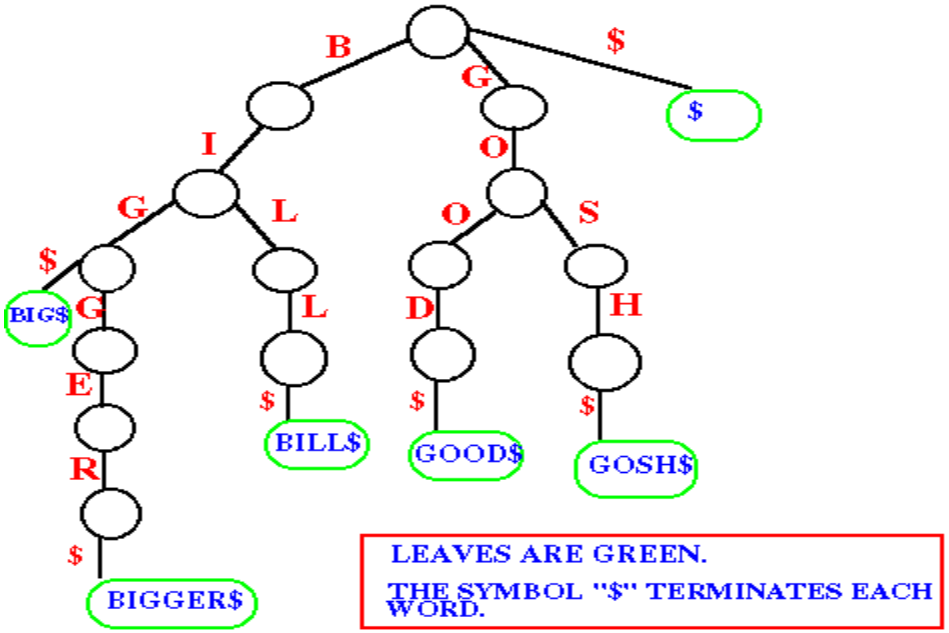
/usr /yiannis

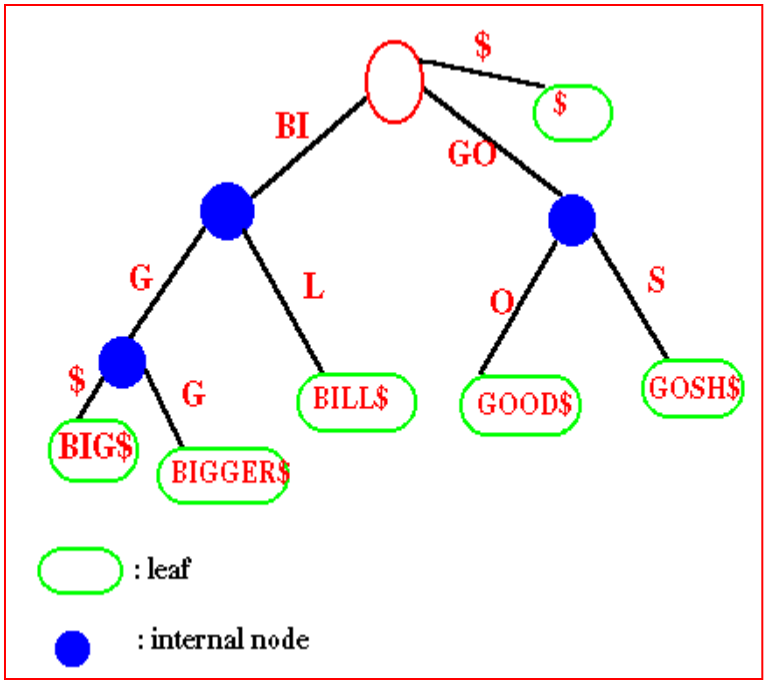
**Μειονεκτήματα**

- **Καταλαμβάνουμε πολύ χώρο ➔ Επαναληπτική αποθήκευση μονοπατιού**
- **Η αναζήτηση κάποιου στοιχείου χρειάζεται O(n) (n μέγεθος λίστας)**
- **Στα δένδρα τα δεδομένα είναι 'καλύτερα' οργανωμένα ώστε οι διαδικασίες αναζήτησης, εισαγωγής και εξαγωγής κόμβου να είναι πιο αποδοτικές (τάξεως O(logn)).**

# Παράδειγμα 2: Suffix **Trie** (Re**Trie**val)

**Μπορούμε να κρατήσουμε ένα μεγάλο λεξικό στην μνήμη του υπολογιστή και να κάνουμε αναζητήσεις για λέξεις αποδοτικά**

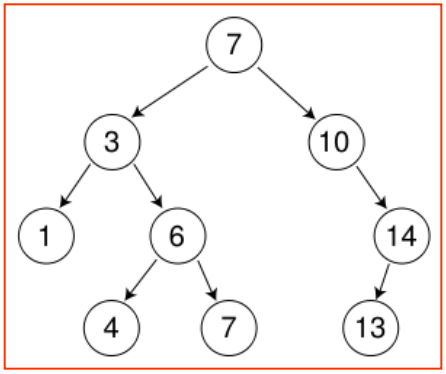



**PATRICIA Trie: Κάθε ακμή περιέχει 1 ή περισσότερους χαρακτήρες (συμπίεση)**

# Ορισμοί

- Όταν υπάρχει **ακμή** από ένα κόμβο **u** σε ένα κόμβο **v** τότε ο **u** είναι ο **πατέρας** ή ο προκάτοχος (**parent**) του v, και ο v είναι **παιδί** (**child**) του **u**.
- Η ορολογία αυτή γενικεύεται ώστε να μιλούμε για **προγόνους** (ancestors) και **απογόνους** (descendants) κόμβων.
- Κόμβοι που έχουν τον ίδιο προκάτοχο ονομάζονται **αδέλφια (siblings)**
- Κόμβοι που δεν έχουν παιδιά λέγονται **φύλλα** (leaves). Οι υπόλοιποι λέγονται **εσωτερικοί** (non-leaves - branches).
- **Βαθμός (degree) ενός κόμβου** είναι ο αριθμός των παιδιών του.
- **Βαθμός ενός δένδρου** είναι ο **μέγιστος** από τους βαθμούς των κόμβων του.
- **Δένδρα βαθμού n** λέγονται **n-αδικά**. Δένδρα που χρησιμοποιούνται συχνά είναι τα δυαδικά (binary), τετραδικά (quadtrees), οκταδικά (octtrees).




**Parent(6) => 3**

**Sibling(1) => 6**

**Leaves = {1,4,7,13}**

**Non-leaves={3,6,7,10,14}**

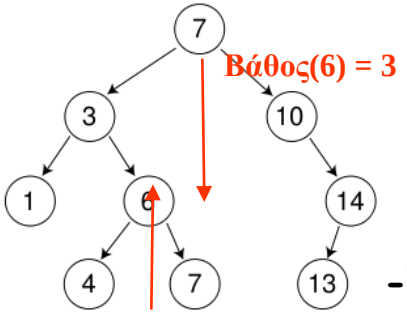
**Root={7}**

**Degree(Tree)=2**

**degree(14) = 1**

# Βάθος και ύψος ενός δένδρου

- **Βάθος ενός κόμβου (Node Depth)** (ή επίπεδο **Level**)
  => αριθμός προγόνων + 1.
  π.χ. το βάθος της ρίζας ενός δένδρου είναι 1.
- **Βάθος ενός δένδρου** ονομάζουμε τον μέγιστο από τα βάθη των κόμβων.
- **Μονοπάτι** ή **διαδρομή** (path) ενός δένδρου είναι μια ακολουθία κόμβων $v_1$, $v_{2,}$, …, $v_k$, όπου κάθε $v_i$ είναι πατέρας του $v_{i+1}$. Το **μήκος** του μονοπατιού $v_1$, $v_{2,}$, …, $v_k$ είναι $k$-1.
- **Ύψος ενός κόμβου** : το μήκος του μονοπατιού από το βαθύτερο κόμβο (στο ίδιο υπό-δένδρο) προς τον κόμβο.
  π.χ. το ύψος οποιουδήποτε φύλλου είναι 0.
- **Ύψος ενός δένδρου** : το ύψος της ρίζας του δένδρου.

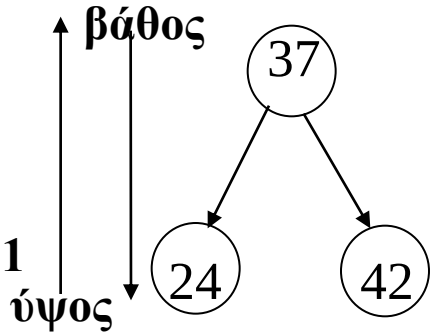


# Παράδειγμα


**Βάθος (Depth/Level) κόμβου:1**

**Ύψος (Height) κόμβου: 0**

**Βαθμός Δένδρου = 0**




**Βάθος (Δένδρου): 2**

**Ύψος (Δένδρου): 1**

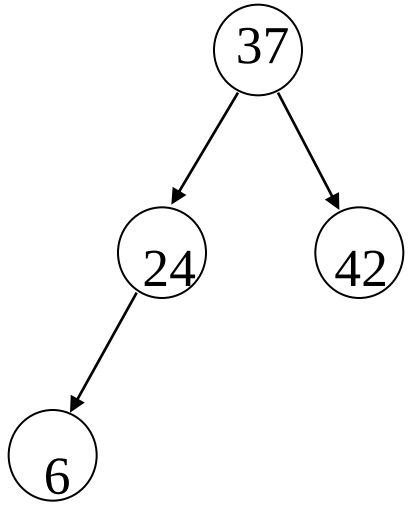
**Βάθος κόμβου 37: 1**

**Ύψος κόμβου 37: 1**

**Βάθος κόμβου 24: 2**

**Ύψος κόμβου 24: 0**

**Βαθμός Δένδρου = 2**




**Βάθος (Δένδρου) : 3**

**Ύψος (Δένδρου) : 2**

**Βάθος κόμβου 24: 2**

**Ύψος κόμβου 24: 1**

**Βάθος κόμβου 6: 3**

**Ύψος κόμβου 6: 0**

**Βαθμός Δένδρου = 2**

# Αναπαράσταση Δένδρων στη Μνήμη

- Μπορούμε να χρησιμοποιήσουμε **στατική** ή **δυναμική χορήγηση** μνήμης, αν και το δεύτερο είναι πιο φυσιολογικός τρόπος.

- Κάθε κόμβος αποτελείται από κάποιο αριθμό πεδίων:
  1. ***Τα πραγματικά δεδομένα του κόμβου*** (DATA και το κλειδί), και
  2. ***Δείκτες σε άλλους κόμβους*** του δένδρου (ένα ανά παιδί) .

```
typedef struct node {
    int val;
    struct node *left;
    struct node *right;
} NODE;
```

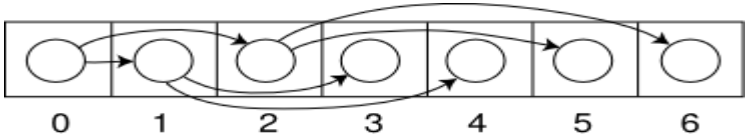
**N=2 παιδιά ανά κόμβο**

- Για δένδρο βαθμού n κάθε κόμβος έχει την πιο κάτω μορφή:

| Info | $p_1$ | $p_2$ | … | $p_n$ |
|---|---|---|---|---|

όπου ο $p_i$ είναι δείκτης στο i-οστό παιδί του κόμβου.

- Εναλλακτικά θα μπορούσαμε να αναπαραστήσουμε ένα δένδρο μέσα σε ένα **στατικά ορισμένο πίνακα**. Π.χ. όπου οι δείκτες είναι απλά αριθμοί (θέσεις του πίνακα).




- Και στις δυο περιπτώσεις θέλουμε 4 bytes ανά δείκτη (σε x86) ή integer θέσης πίνακα. Αν έχουμε ένα 5-αδικό δέντρο τότε κάθε κόμβος χρειάζεται επιπλέον 20 bytes.
- Αν ένας κόμβος δεν έχει παιδία τότε αυτοί οι δείκτες θα μένουν αναξιοποίητοι, επομένως θα έχουμε σπάταλη χώρου.

# Αναπαράσταση Δένδρων στη Μνήμη

- **Πρόβλημα**: Μείωση της σπατάλης χώρου από **αχρησιμοποίητους** δείκτες.

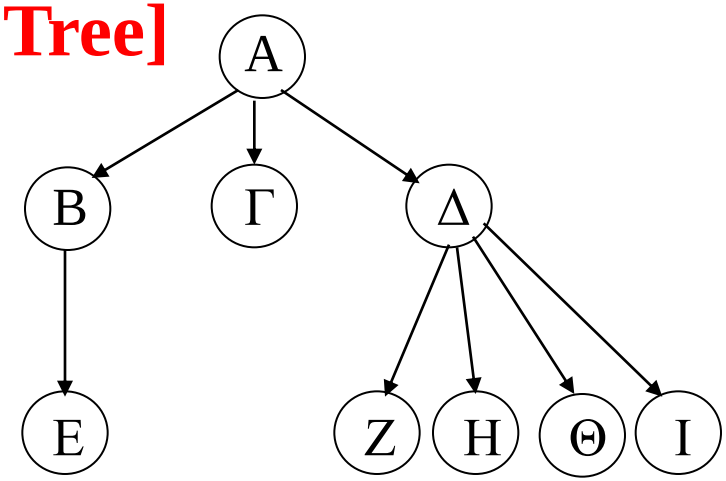
- **Λύση**: Μπορούμε να μετατρέψουμε ένα δέντρο **n-αδικό** (κάθε κόμβος με n παιδιά) σε ένα **2-αδικό** (κάθε κόμβος με 2 παιδιά) χρησιμοποιώντας την πολιτική «**Πρώτο Παιδί / Επόμενο Αδέρφι**» **(First Child/Next Sibling)**

- Δηλαδή κάθε κόμβος u περιέχει
  1. τα δεδομένα του κόμβου,
  2. δείκτη στο αριστερότερο (πρώτο) παιδί του u,
  3. δείκτη στον επί δεξιά αδελφό του u, αν υπάρχει.

- Αυτή η πολιτική μας επιτρέπει να μειώσουμε τον αριθμό των δεικτών

# Παράδειγμα αναπαράστασης δένδρου

**[Normal Tree]**




**Δένδρο Βαθμού n (n=4)**

**Συνολικοί δείκτες:**

**9 nodes *4 pointers = 36**

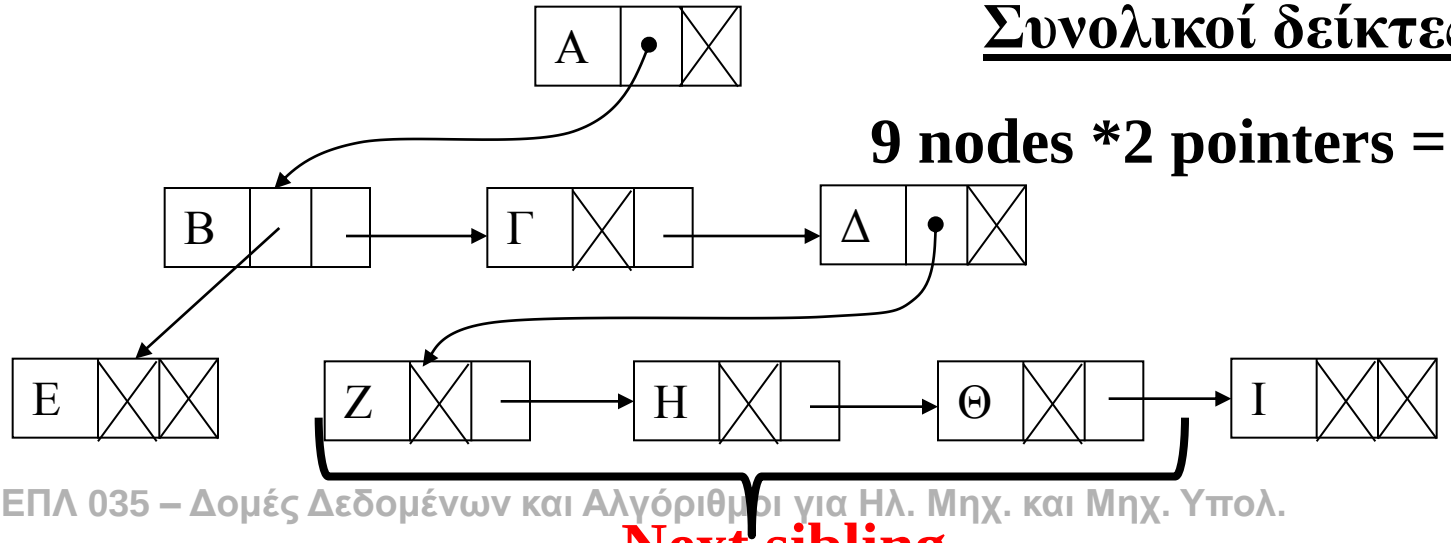
---

**[First Child/Next Sibling]**

**Δένδρο Βαθμού n (n=2)**

**Συνολικοί δείκτες:**

**9 nodes *2 pointers = 18**



# Διάσχιση Δένδρων

- Αν θέλουμε να **επισκεφθούμε όλους τους κόμβους** ενός δένδρου, μπορούμε να χρησιμοποιήσουμε ένα από τους πιο κάτω τρόπους, οι οποίοι **διαφέρουν στη σειρά με την οποία εξετάζουν τους κόμβους.**

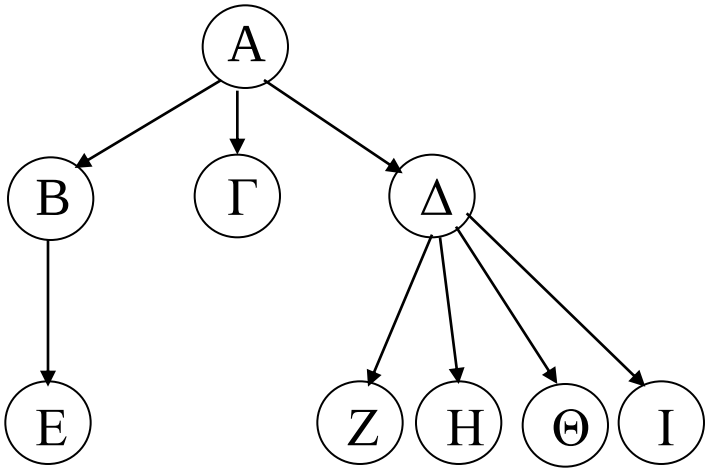
1. ***Προθεματική Διάσχιση*: (Preorder Traversal)** επισκεπτόμαστε (εκτυπώνουμε) πρώτα τη ρίζα και ύστερα τα παιδιά της. Αναδρομικά η πράξη ορίζεται ως εξής:

```
Print_Preorder(treenode u)
    Print u;
    foreach child v of u
        Print_Preorder(v)
```

2. ***Μεταθεματική Διάσχιση*: (Postorder Traversal)** επισκεπτόμαστε (εκτυπώνουμε) πρώτα τα παιδιά και ύστερα τη ρίζα του δένδρου. Αναδρομικά η πράξη ορίζεται ως εξής:

```
Print_Postorder(treenode u)
    foreach child v of u
        Print_Postorder(v)
    Print u;
```

# Παράδειγμα Διάσχισης δένδρου




```
Print_Preorder(treenode u)
    Print u;
        foreach child v of u
            Print_Preorder(v)
```

```
Print_Postorder(treenode u)
        foreach child v of u
            Print_Postorder(v)
        Print u;
```

**Προθεματική Διάσχιση (Preorder) :** **Α** Β Ε Γ Δ Ζ Η Θ Ι

**Μεταθεματική Διάσχιση (Postorder) :** Ε Β Γ Ζ Η Θ Ι Δ **Α**

# Χρήσιμες Πράξεις του ΑΤΔ Δένδρο

| | |
|---|---|
| `Parent(u)` | επέστρεψε τον πατέρα του u |
| `Children(u)` | επέστρεψε τα παιδιά του u |
| `FirstChild(u)` | επέστρεψε το πρώτο παιδί του u |
| `RightSibling(u)` | επέστρεψε τον κόμβο στα δεξιά του u |
| `LeftSibling(u)` | επέστρεψε τον κόμβο στα αριστερά του u |
| `IsLeaf(u)` | αν το u είναι φύλλο τότε δώσε true, διαφορετικά δώσε false |
| `IsRoot(u)` | αν το u είναι η ρίζα του δένδρου δώσε true, διαφορετικά δώσε false |
| `Depth(u)` | δώσε το βάθος του u στο δένδρο. |

Η υλοποίηση αυτών των συναρτήσεων θα γίνει στο εργαστήριο