

# Λεξικό, Union – Find

---

**Δημήτρης Φωτάκης**

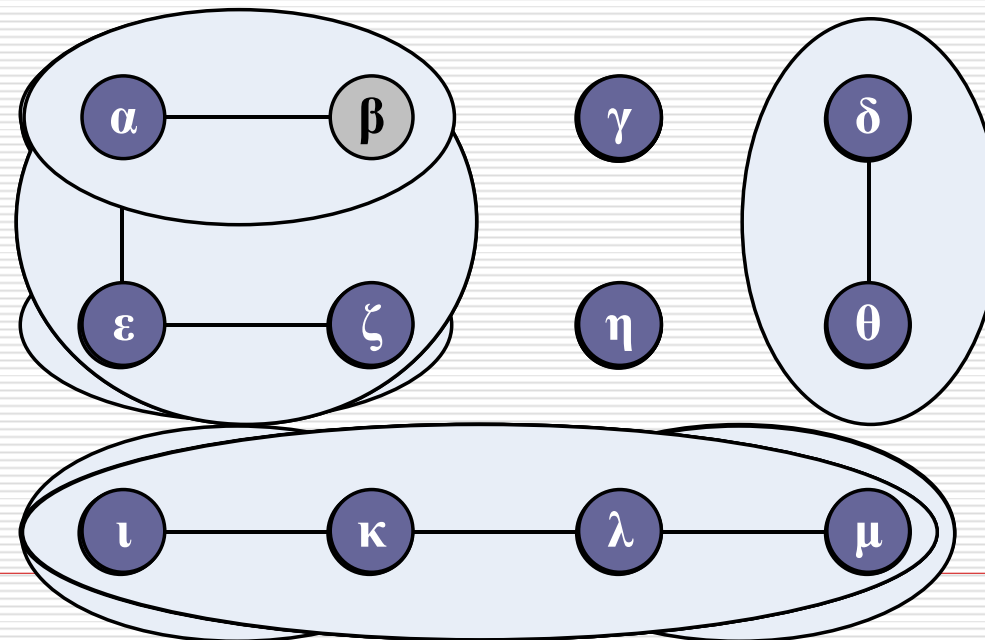
Σχολή Ηλεκτρολόγων Μηχανικών  
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο



# Διαχείριση Διαμερίσεων Συνόλου

- ❑ Στοιχεία σύμπαντος διαμερίζονται σε **κλάσεις ισοδυναμίας** που **μεταβάλλονται δυναμικά** με ένωση.
- ❑ Λειτουργίες:
  - Εύρεση **find(x)**: **αντιπρόσωπο** κλάσης όπου ανήκει x.
  - Ένωση **union(x, y)**: **ένωση** κλάσεων όπου ανήκουν x και y.



# Πρόβλημα Union – Find

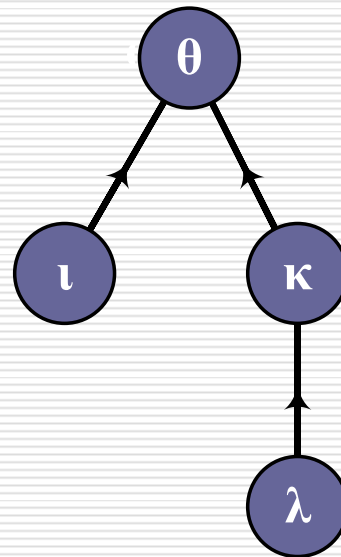
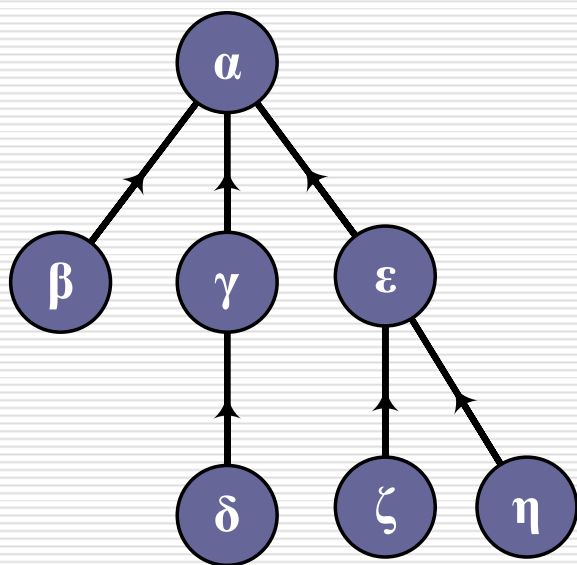
---

- Στοιχεία  $U = \{1, 2, \dots, n\}$  αρχικά σε  $n$  κλάσεις.
  - Κάθε κλάση προσδιορίζεται από στοιχείο – αντιπρόσωπο.
- $\text{find}(x)$ : αντιπρόσωπος κλάσης όπου ανήκει  $x$ .
  - Διατηρούμε μοναδικό αντιπρόσωπο για κάθε κλάση.
- $\text{union}(x, y)$ : αντικατάσταση (αντιπροσώπων) κλάσεων  $x$  και  $y$  με κλάση που προκύπτει από ένωση.
  - Ελέγχουμε αν  $x$  και  $y$  ανήκουν σε διαφορετική κλάση.
  - Νέος αντιπρόσωπος από τους αντιπροσώπους κλάσεων  $x, y$
  - Πάντα διαμέριση του  $U$  σε κλάσεις.
  - $\leq n - 1$  ενώσεις (μετά από  $n - 1$ , μία μόνο κλάση).
- Δομή δεδομένων που ελαχιστοποιεί συνολικό χρόνο για ακολουθία  $m$  ευρέσεων και  $n - 1$  ενώσεων.

# Αναπαράσταση Δέντρου & Δάσους

□ Πίνακας γονέων  $A[1...n]$  για δέντρο με ρίζα και  $n$  κόμβους:

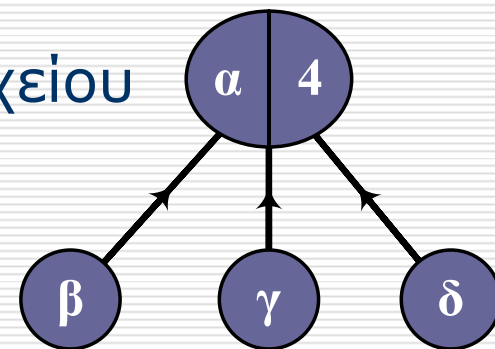
- $A[i] = j$  ανν  $j$  πατέρας του  $i$  στο δέντρο.
- $A[\text{ρίζας}] = \text{ρίζα}$  (ή  $-1$ ).
- Όμοια για δάσος όπου κάθε δέντρο έχει ρίζα.



α	α	α	γ	α	ε	ε	θ	θ	θ	κ
α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ

# Αναπαράσταση με Δέντρα

- Κλάση: **δέντρο** με **ρίζα** το στοιχείο-**αντιπρόσωπο**.
  - Όνομα ρίζας.
  - Μέγεθος κλάσης.
- Στοιχείο: **κόμβος** δέντρου με πεδία
  - Όνομα στοιχείου.
  - Όνομα γονέα: όνομα προηγούμενου στοιχείου στο μονοπάτι προς τη ρίζα-αντιπρόσωπο.
- Αναπαράσταση με πίνακα γονέων:
  - $A[x]$  : γονέας στοιχείου  $x$ .
  - Ρίζα – στοιχείο αντιπρόσωπος έχει  $A[x] = x$  και επιπλέον πεδίο **size**.



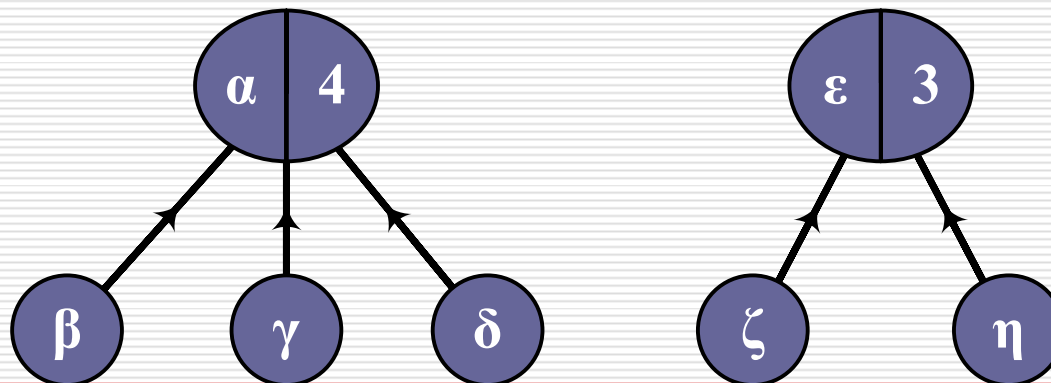
4			
α	α	α	α
α	β	γ	δ

# Αναπαράσταση με Δέντρα

- **find(x)**: ακολουθούμε δείκτες σε γονέα μέχρι τη ρίζα.

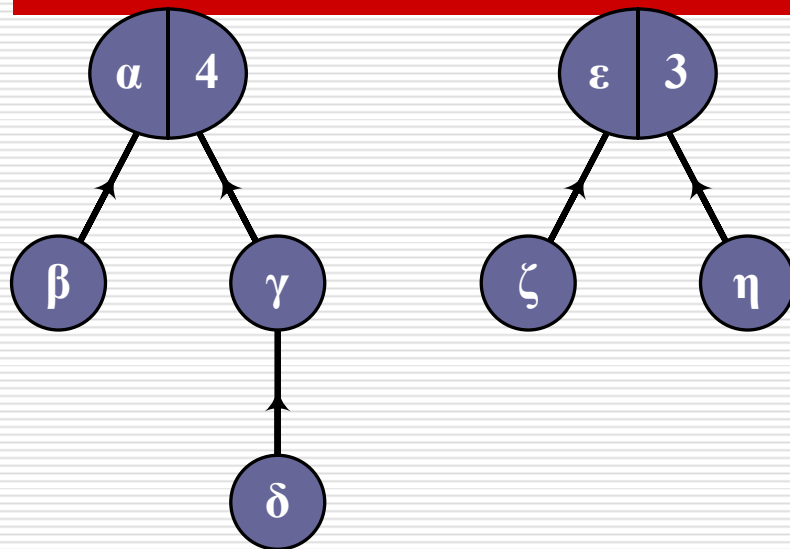
```
elem find(elem x) {  
    while (x != A[x])  
        x = A[x];  
    return x; }
```

- **union(x, y)**: x και y αντιπρόσωποι διαφορετικών συνόλων
  - Συνένωση δέντρων: ρίζα 1<sup>ου</sup> συν. γίνεται γονέας ρίζας 2<sup>ου</sup> συν.
  - Ενημέρωση μεγέθους

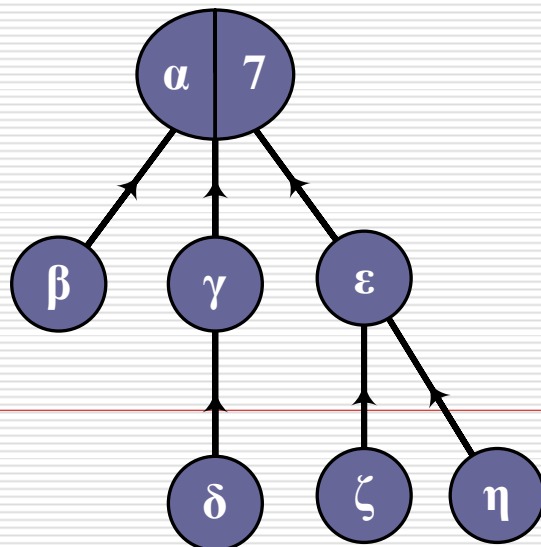


# Ένωση

---



```
unionTree(elem x, elem y) {  
    if (x == y) return;  
    A[y] = x;  
    A[x].size += A[y].size; }
```



# Απόδοση

---

- Χρόνος χ.π. για  $m$  finds και  $n$  unions:  $O(mn + n)$ 
  - Union :  $O(1)$  χρόνος.
  - Find :  $O(\text{ύψος δέντρου})$ 
    - Χειρότερη περίπτωση:  $\text{ύψος} = n - 1$ 
      - $\text{union}(n-1, n), \text{union}(n-2, n-1), \text{union}(n-3, n-2),$   
 $\text{union}(n-4, n-3), \dots, \text{union}(3, 2), \text{union}(1, 2).$
- Απλή δομή, εύκολη υλοποίηση, αλλά ακριβό find!

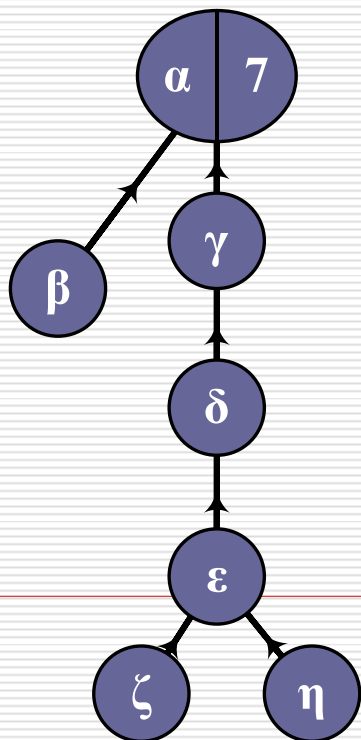


# Βεβαρυμένη Ένωση

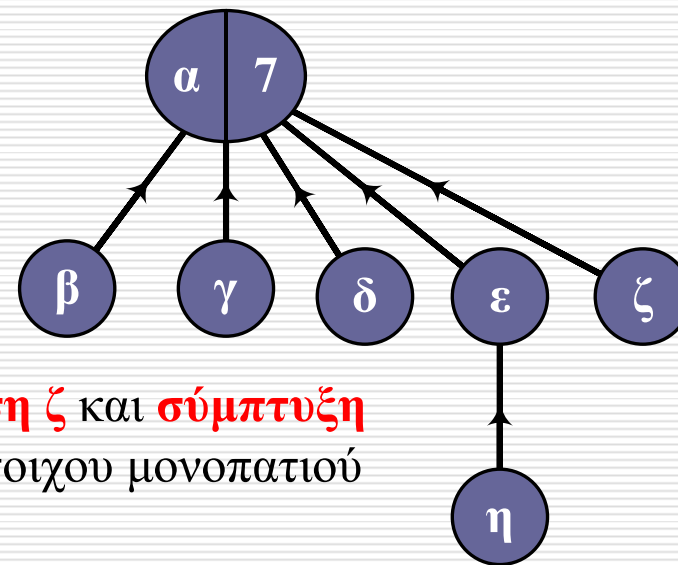
- «Δεύτερο» σύνολο αυτό με τα λιγότερα στοιχεία.
  - Λογαριθμικό ύψος δέντρου :  $O(\log n)$ .
  - Βεβαρυμένη ένωση: δέντρο ύψους  $h$  έχει  $\geq 2^h$  στοιχεία.
- Απόδειξη με επαγωγή:
  - Ισχύει για  $h = 0$  (δέντρο ενός στοιχείου).
  - Ένωση δέντρων  $x$  και  $y$  με ύψη  $h_x, h_y$ , και στοιχεία  $s_x \geq s_y$
  - Επαγωγικά, υποθέτουμε  $s \geq 2^h$  (για  $x$  και  $y$ )
    - Ύψος ένωσης =  $h_x$  : στοιχεία ένωσης  $\geq 2^{\text{ύψος}}$
    - Ύψος ένωσης =  $h_y + 1$  : στοιχεία ένωσης  $\geq 2 s_y \geq 2^{\text{ύψος}}$
- Χρόνος χ.π. για  $m$  finds και  $n$  unions:  $O(m \log n + n)$ 
  - Απλή υλοποίηση και αποδεκτή απόδοση.

# Σύμπτυξη Μονοπατιών

- Find **ακριβό** γιατί στοιχεία **μακριά από ρίζα**.
- Σύμπτυξη μονοπατιού όταν find(x):
  - Όλοι οι **πρόγονοι του x** (και το x) γίνονται **παιδιά ρίζας**.
  - Δέντρο «**κονταίνει**» (όχι **επιβάρυνση** ασυμπτωτικού χρόνου).
  - Στο μέλλον, θα **βρίσκουμε** σύνολο των στοιχείων **γρήγορα**.



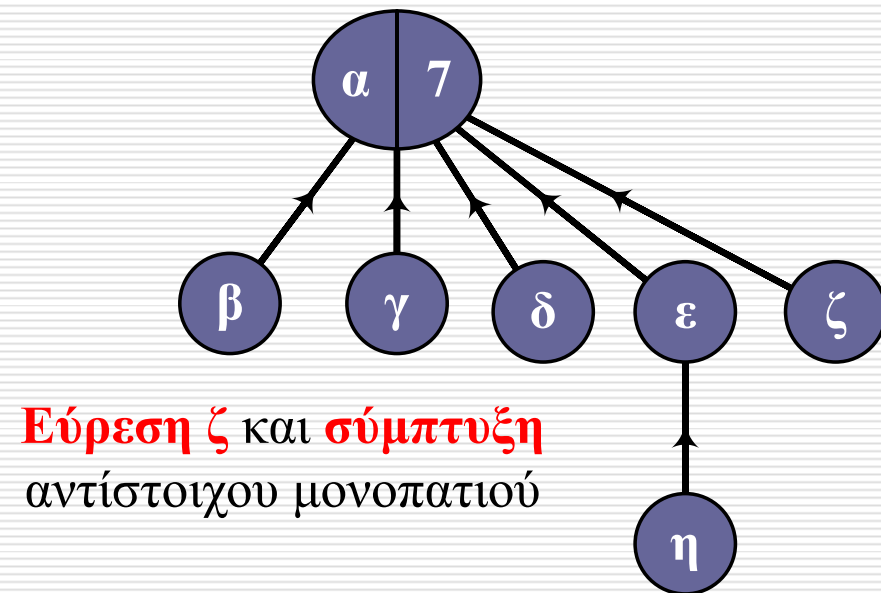
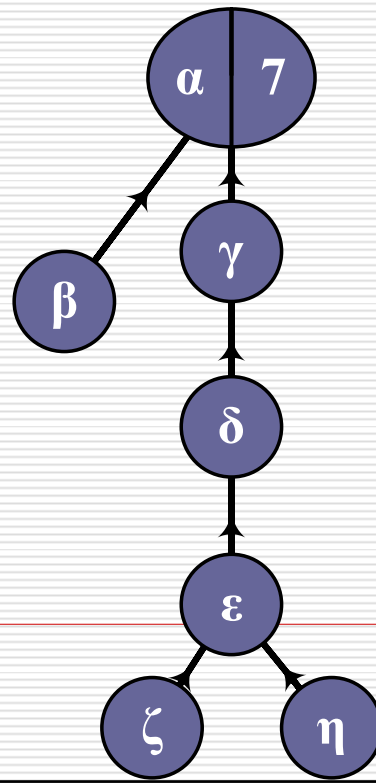
**Εύρεση ζ και σύμπτυξη**  
αντίστοιχου μονοπατιού



# Σύμπτυξη Μονοπατιών

```
elem findTreePathCompression(elem x) {  
    if (x != A[x])  
        A[x] = findTreePathCompression(A[x]);  
    return(A[x]);    }
```

- Ανεβαίνουμε μέχρι ρίζα.
- Επιστρέφοντας μέχρι  $x$ , όλοι οι δείκτες γονέων τίθενται να δείχνουν στη ρίζα.



# Απόδοση

---

- Δέντρα, βεβαρυμένη ένωση, και σύμπτυξη μονοπατιών.
- Χρόνος χ.π. για  $m \geq n$  finds και  $n$  unions:  $O(m \alpha(n, m))$ 
  - $\alpha(n, m)$ : αντίστροφη συνάρτηση Ackermann.
  - Μεγαλώνει εξαιρετικά αργά!
  - Στην πράξη, μπορεί να θεωρηθεί σταθερά.
- Απλή δομή, εύκολη υλοποίηση, και ουσιαστικά γραμμικός χρόνος!

# Πρόβλημα (ADT) Λεξικού

---

- Δυναμικά μεταβαλλόμενη συλλογή αντικειμένων που αναγνωρίζονται με «κλειδί» (π.χ. κατάλογοι, πίνακες ΒΔ).
- **Λεξικό** : συλλογή αντικειμένων με μοναδικό «κλειδί».
  - «Κλειδί»: αριθμός ή τύπος δεδομένων με ολική διάταξη.
  - Γενίκευση και για μη-μοναδικά κλειδιά.
- ADT λεξικού υποστηρίζει ακολουθίες λειτουργιών:
  - Αναζήτηση στοιχείου με κλειδί  $x$ 
    - `member(x)`: ελέγχει ύπαρξη στοιχείου με κλειδί  $x$
    - `search(x)`: επιστρέφει δείκτη σε θέσεις  $x$
  - Εισαγωγή στοιχείου με κλειδί  $x$
  - Διαγραφή στοιχείου με κλειδί  $x$

# Λειτουργίες Λεξικού

---

- Λεξικό υποστηρίζει λειτουργίες:
  - Αναζήτηση/εισαγωγή/διαγραφή στοιχείου με κλειδί  $x$
  - Εκτύπωση στοιχείων σε αύξουσα / φθίνουσα σειρά
  - Προηγούμενο και επόμενο στοιχείο.
  - Μέγιστο και ελάχιστο στοιχείο.
  - $k$ -οστό μικρότερο στοιχείο
  - Βοηθητικές λειτουργίες ...

# Υλοποιήσεις Λεξικού

---

- Μη-ταξινομημένη **διασυνδεδεμένη λίστα**:
  - Εισαγωγή:  $O(1)$
  - Αναζήτηση / τυχαία διαγραφή:  $O(n)$
  - Κατάλληλη όταν **συχνές εισαγωγές**, σπάνιες αναζητήσεις / διαγραφές μεμονωμένες ή στο τέλος (π.χ. log file).
- Ταξινομημένος **πίνακας**:
  - (Δυαδική) αναζήτηση:  $O(\log n)$
  - **Στατική συλλογή** : «εισαγωγή»  $O(\log n)$  / στοιχείο  
Χρόνος **ταξινόμησης** :  $O(n \log n)$
  - Δυναμική συλλογή : εισαγωγή / διαγραφή  $O(n)$
  - Κατάλληλη όταν **συχνές αναζητήσεις** και δεδομένα μεταβάλλονται σπάνια (π.χ. αγγλο-ελληνικό λεξικό).

# Υλοποιήσεις Λεξικού

---

- Ζυγισμένο (Δυαδικό) Δέντρο Αναζήτησης:
  - Αναζήτηση / εισαγωγή / διαγραφή:  $O(\log n)$
  - Μέγιστο / ελάχιστο / προηγούμενο / επόμενο /  $k$ -οστό:  $O(\log n)$
  - **Range queries** σε γραμμικό χρόνο.
  - Πλήρως δυναμική – επιπλέον χώρος για δείκτες!
- Πίνακας Κατακερματισμού (hashing):
  - Αναζήτηση / διαγραφή :  $O(1)$
  - Εισαγωγή :  $O(1)$  expected amortized,  $O(\log n)$  (ακόμη και  $O(1)$ ) whp.,  $O(n)$  χ.π.
  - **Δεν υποστηρίζει** αποδοτικά **άλλες λειτουργίες**.
  - Δυναμική – επιπλέον χώρος στον πίνακα (util  $\approx 50\%$ )