

Ουρές Προτεραιότητας, Διαδικός Σωρός, Union-Find

**Δημήτρης Φωτάκης, Αριστείδης Παγουρτζής, Δώρα
Σούλιου, Παναγιώτης Γροντάς**

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο



18.10.22

Περιεχόμενα

- Δομές Δεδομένων
- Σωροί
 - Ορισμός – Ιδιότητα Σωρού
 - Αναπαράσταση με πίνακα
 - Αποκατάσταση ιδιότητας σωρού
 - Δημιουργία σωρού
- Heap Sort
- Ουρές Προτεραιότητας
- Δομές Union-Find (disjoint-set)
 - Αναπαράσταση με πίνακα
 - Join By Size
 - Path Compression

Δομές Δεδομένων

Δομές Δεδομένων

- (Αναπαράσταση,) **οργάνωση** και **διαχείριση** συνόλων αντικειμένων για αποδοτική **ενημέρωση** και **ανάκτηση** πληροφορίας.
 - Αποδοτική υλοποίηση αλγορίθμων και Βάσεων Δεδομένων.
- (Αποδοτική) **αναπαράσταση – οργάνωση** «σύνθετων» αντικειμένων με χρήση:
 - Βασικών τύπων δεδομένων (ints, floats, chars, strings, arrays).
 - Μηχανισμών που παρέχονται από γλώσσες προγραμματισμού (structs – records, objects).
- **Διαχείριση:** υλοποίηση στοιχειωδών λειτουργιών
 - Ταξινόμηση, αναζήτηση, min/max/median, first/last, ...
 - Εισαγωγή, διαγραφή, ενημέρωση.
- Λύσεις και τεχνικές για **αποδοτική διαχείριση** δεδομένων.
 - Ανάλυση για απαιτήσεις και καταλληλότητα.

Γενικευμένος Τύπος Δεδομένων

- **Abstract Data Type** (ADT): **σύνολο** (στιγμιότυπα) με **λειτουργίες** (μεθόδους) επί των στοιχείων του.
- **Δομή Δεδομένων**: **Υλοποίηση** ενός ADT
 - **Αναπαράσταση – οργάνωση** στιγμιοτύπων και **υλοποίηση λειτουργιών** με κατάλληλους αλγόριθμους.
 - **Διατύπωση**: ορισμός αναπαράστασης και περιγραφή υλοποίησης λειτουργιών (ψευδο-κώδικας).
 - **Ανάλυση**: προσδιορισμός απαιτήσεων σε χώρο αποθήκευσης και χρόνο εκτέλεσης για κάθε (βασική) λειτουργία.
- **Αναλογία Java: Interface vs Κλάση**

Ουρά Προτεραιότητας (Priority Queue)

- Ουρά όπου **σειρά διαγραφής** καθορίζεται από **προτεραιότητα (μεγαλύτερη – μικρότερη)**.
- Στοιχεία (**προτεραιότητα, πληροφορία**).
- Ακολουθία από λειτουργίες:
 - **insert(x)**: εισαγωγή x.
 - **deleteMax()**: διαγραφή και επιστροφή στοιχείου μέγιστης προτεραιότητας.
 - **max()**: επιστροφή στοιχείου μέγιστης προτεραιότητας (χωρίς διαγραφή).
 - **changePriority(k)**: αλλαγή προτεραιότητας θέσης k.
 - **isEmpty(), size()**: βοηθητικές λειτουργίες.

Εφαρμογές

- Άμεσες εφαρμογές:
 - Υλοποίηση ουρών αναμονής με προτεραιότητες.
 - Δρομολόγηση με προτεραιότητες.
 - Largest (Smallest) Processing Time First.

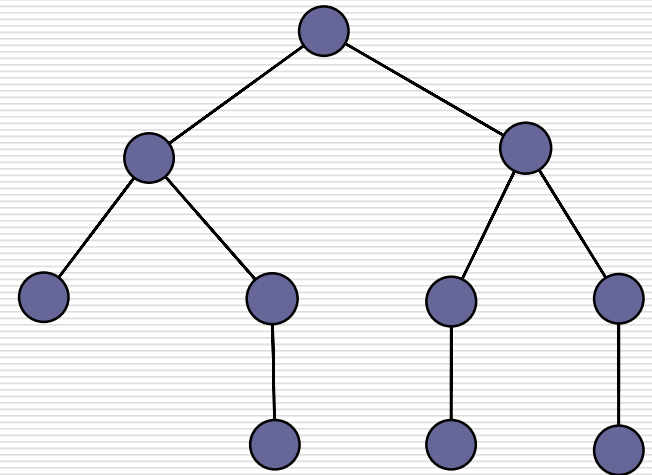
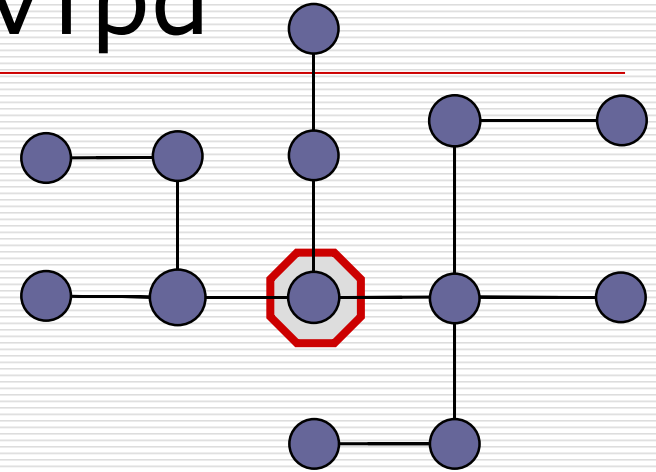
- Έμμεσες εφαρμογές:
 - Βασικό συστατικό **πολλών** ΔΔ και αλγορίθμων:
 - HeapSort (γενικά ταξινόμηση με επιλογή).
 - Αλγόριθμος Huffman.
 - Αλγόριθμοι Prim και Dijkstra.
 - ...

Στοιχεία Ουράς Προτεραιότητας

- Ουρές Προτεραιότητας:
 - **Ολική διάταξη** στοιχείων με βάση προτεραιότητα.
 - Στοιχεία είναι **αριθμοί** (με συνήθη διάταξη) που δηλώνουν προτεραιότητα.
 - Εφαρμογή για στοιχεία **κάθε συνόλου** με σχέση **ολικής διάταξης** (αριθμοί, λέξεις, εισοδήματα, ...).
- Ουρά Προτεραιότητας με **γραμμική λίστα**:
 - Διαγραφή μέγιστου ή εισαγωγή απαιτεί **γραμμικό χρόνο**.
- Υλοποίηση ουράς προτεραιότητας με **σωρό (heap)**.
 - **Διαδικό δέντρο** με διάταξη σε κάθε μονοπάτι ρίζα – φύλλο.

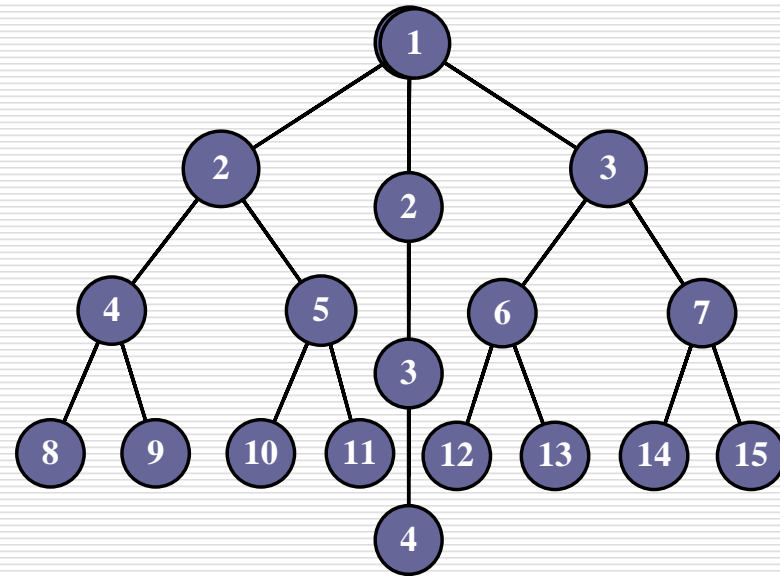
Ιεραρχικές Δομές: Δέντρα

- Γράφημα **ακυκλικό** και **συνεκτικό**.
- Δέντρο με n **κορυφές** έχει $m = n - 1$ **ακμές**.
- Δέντρο με **ρίζα** : **Ιεραρχία**
- **Ύψος** : μέγιστη απόσταση φύλλου από ρίζα.
- **Δυαδικό δέντρο** : έχει **ρίζα** και κάθε κορυφή ≤ 2 **παιδιά** :
 - **Αριστερό** και **δεξιό**.
- Κάθε **υποδέντρο** είναι δυαδικό δέντρο.



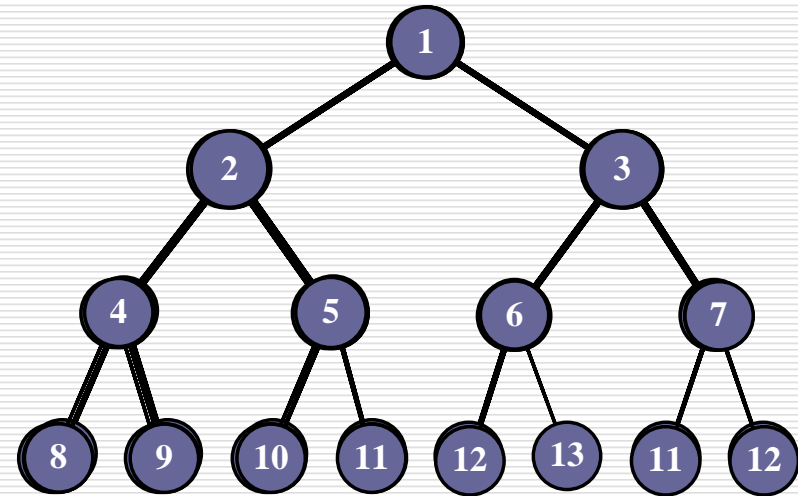
Δυαδικά Δέντρα

- #κορυφών για ύψος = h :
 $h+1 \leq \# \text{κορυφών} \leq 2^{h+1} - 1$
 - $h+1$ επίπεδα, ≥ 1 κορ. / επίπ.
 - $\leq 2^i$ κορυφές στο επίπεδο i .
 $1 + 2 + 4 + \dots + 2^h = 2^{h+1} - 1$
- Ύψος για #κορυφών = n :
 $\log_2(n+1) - 1 \leq \text{ύψος} \leq n - 1$
- **Γεμάτο** (full):
 - Κάθε κορυφή είτε φύλλο είτε 2 παιδιά.
- **Πλήρες** (complete) :
 - Όλα τα επίπεδα συμπληρωμένα (εκτός ίσως τελευταίο).
- **Τέλειο** (perfect) : $n = 2^{h+1} - 1$



Πλήρες

- Όλα τα επίπεδα **συμπληρωμένα εκτός ίσως από τελευταίο** που γεμίζει από αριστερά προς τα δεξιά.
- $n(h)$: #κορυφών για ύψος h :
 $2^h \leq n(h) \leq 2^{h+1} - 1$
 - $\text{τέλειο}(h) : 2^{h+1} - 1$
 - $\text{τέλειο}(h - 1) + 1 : (2^h - 1) + 1 = 2^h$.
- $h(n)$: ύψος για n κορυφές:
 $\log_2(n+1) - 1 \leq h(n) \leq \log_2 n$
- Ύψος : $h(n) = \lfloor \log_2 n \rfloor$
- #φύλλων = $\lceil n / 2 \rceil$



Αναπαράσταση

□ **Δείκτες** σε παιδιά, πατέρα (δυναμική).

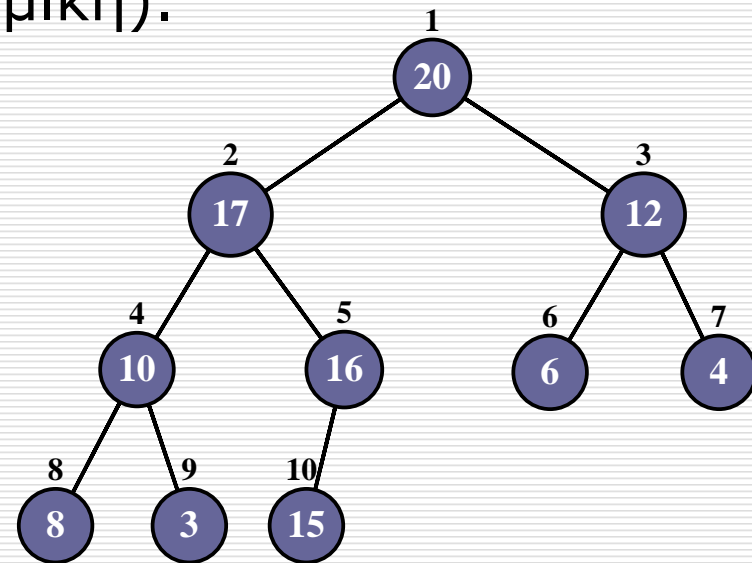
□ **Πλήρη** δυαδικά δέντρα :

■ **Πίνακας** (στατική).

■ Αρίθμηση αριστερά → δεξιά
και πάνω → κάτω.

■ **Ρίζα** : $\Pi[1]$

■ $\Pi[i]$: πατέρας $\Pi[i/2]$
αριστερό παιδί $\Pi[2i]$
δεξιό παιδί $\Pi[2i+1]$

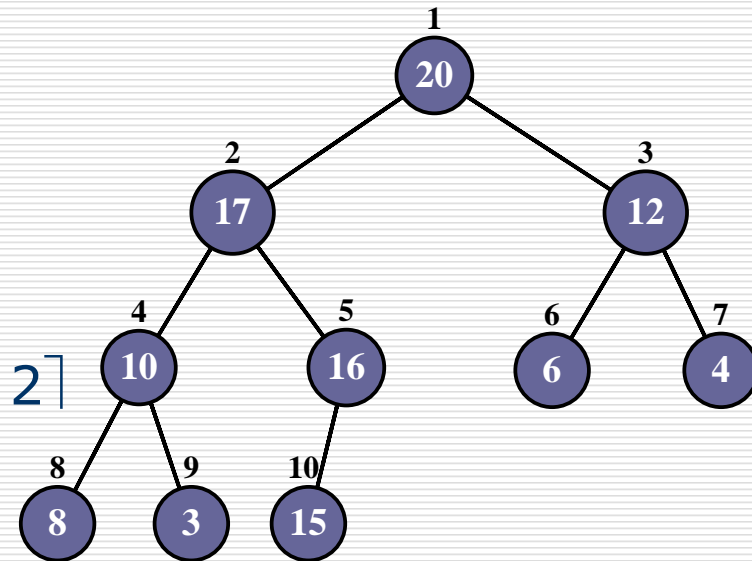


20	17	12	10	16	6	4	8	3	15
----	----	----	----	----	---	---	---	---	----

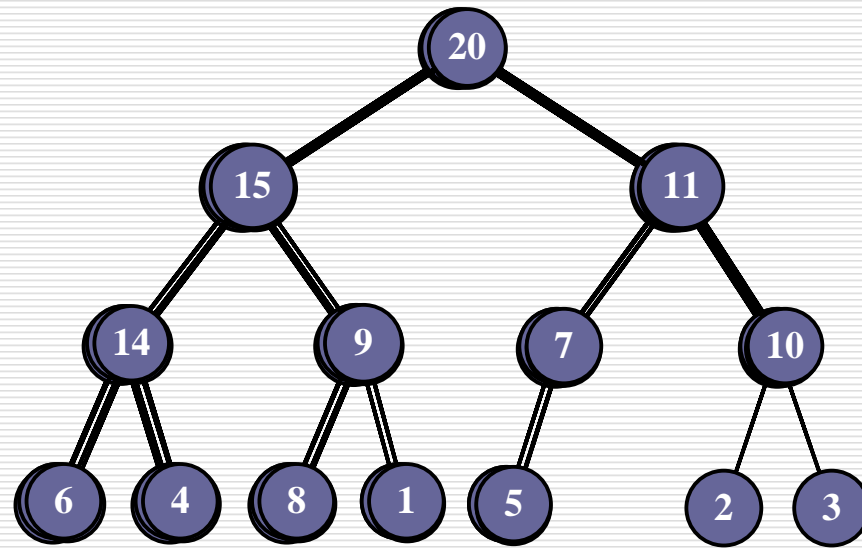
Δυαδικοί Σωροί

Σωρός (heap)

- Δέντρο **μέγιστου** (ελάχιστου):
Τιμές στις κορυφές και **τιμή κάθε κορυφής** \geq (\leq) **τιμές παιδιών της**.
- **Σωρός** : πλήρες δυαδικό δέντρο μέγιστου (ελάχιστου).
 - Ύψος $\Theta(\log n)$, #φύλλων = $\lceil n / 2 \rceil$
- Πίνακας **A[]** ιδιότη. **σωρού** :
 $\forall i \ A[i] \geq A[2i], A[2i+1]$.
- **Μέγιστο** : ρίζα
Ελάχιστο : κάποιο φύλλο



Σωροί και Μη-Σωροί



Αποκατάσταση Προς-τα-Κάτω

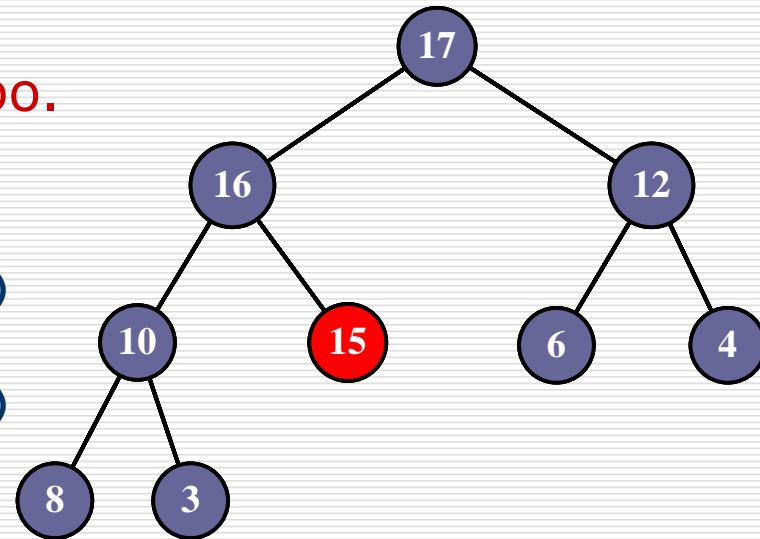
□ `combine(i)` :

Ενόσω όχι σωρός,

- $A[i] \leftrightarrow \max\{A[2i], A[2i+1]\}$

- συνεχίζω στο αντίστοιχο υποδέντρο.

```
combine(int i) {  
    l = 2*i; r = 2*i+1; mp = i;  
    if ((l <= hs) && (A[l] > A[mp]))  
        mp = l;  
    if ((r <= hs) && (A[r] > A[mp]))  
        mp = r;  
    if (mp != i) {  
        swap(A[i], A[mp]);  
        combine(mp); } }
```



□ Χρόνος για `deleteMax()` : $O(\text{ύψος}) = O(\log n)$

Σωρός σαν Ουρά Προτεραιότητας

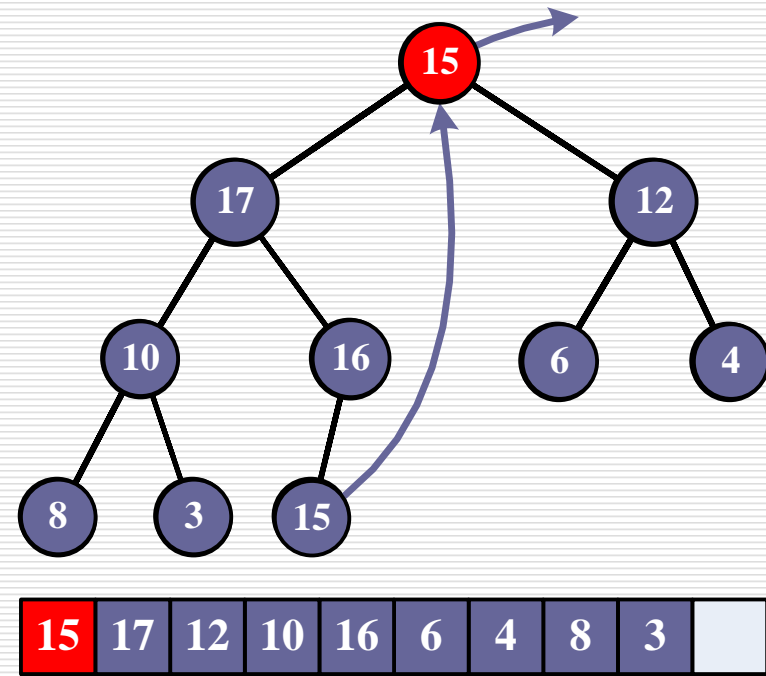
□ `int A[n], hs;`

□ `max() : O(1)`

```
int max() { return(A[1]); }
```

□ `deleteMax() :`

```
int deleteMax() {  
    if (isEmpty()) return(EMPTY);  
    max = A[1]; A[1] = A[hs--];  
    combine(1);  
    return(max); }
```

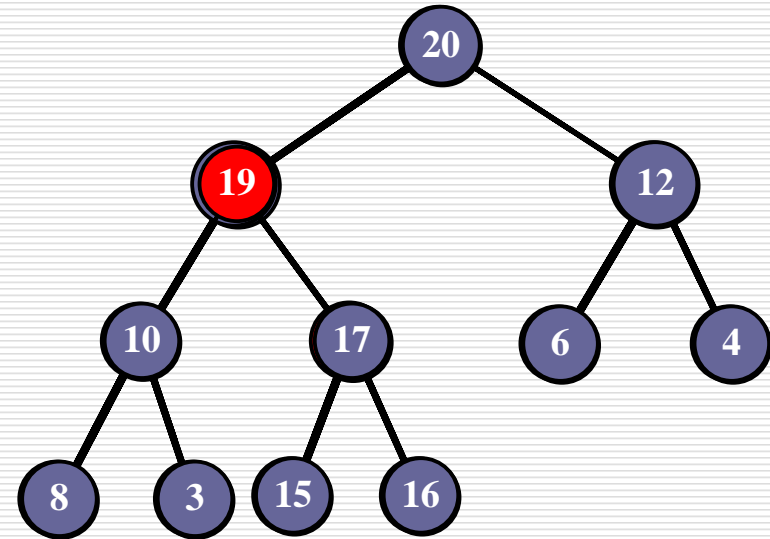


Γιατί να μην αντικαταστήσω απλά τη ρίζα με το μέγιστο παιδί της?

Εισαγωγή: Αποκατάσταση Προς-τα-Πάνω

- `insert(k)` :
 - Εισαγωγή στο **τέλος**.
 - Ενώσω όχι σωρός, $A[i] \leftrightarrow A[i/2]$

```
insert(int k) {  
    A[++hs] = k;  
    i = hs; p = i / 2;  
    while ((i > 1) && (A[p] < A[i]))  
    {  
        swap(A[p], A[i]);  
        i = p; p = i / 2; } }  
}
```

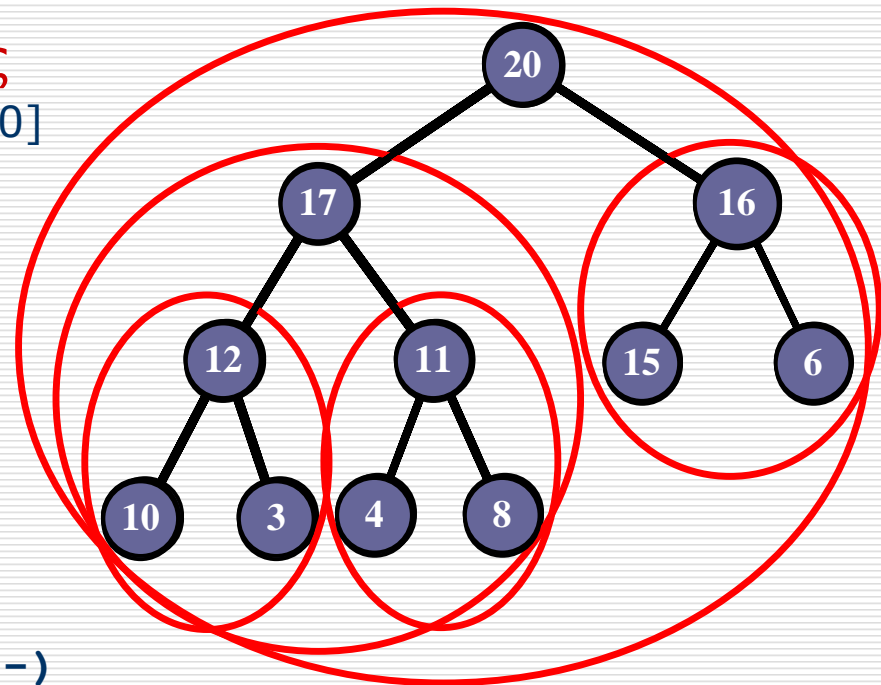


- Χρόνος για `insert()` : $O(\text{ύψος}) = O(\log n)$
- **Αύξηση προτεραιότητας** : **εισαγωγή** (αποκατ. προς-τα-πάνω).
Μείωση προτεραιότητας : **διαγραφή** (αποκατ. προς-τα-κάτω).

Δημιουργία Σωρού

- $A[n]$ → σωρός με n εισαγωγές
[3, 4, 6, 10, 8, 15, 16, 17, 12, 11, 20]
- Χρόνος $O(n \log n)$.
- **Ιεραρχικά** (bottom-up):
Υποδέντρα-σωροί **ενώνονται**
σε δέντρο-σωρό.

```
constructHeap(int n) {  
    hs = n;  
    for (i = n / 2; i > 0; i--)  
        combine(i);  
}
```



Χρόνος Δημιουργίας

```
for (i = n / 2; i > 0; i--)  
    combine(i);
```

- Χρόνος $\text{combine}(i) = O(\text{ύψος } i)$.

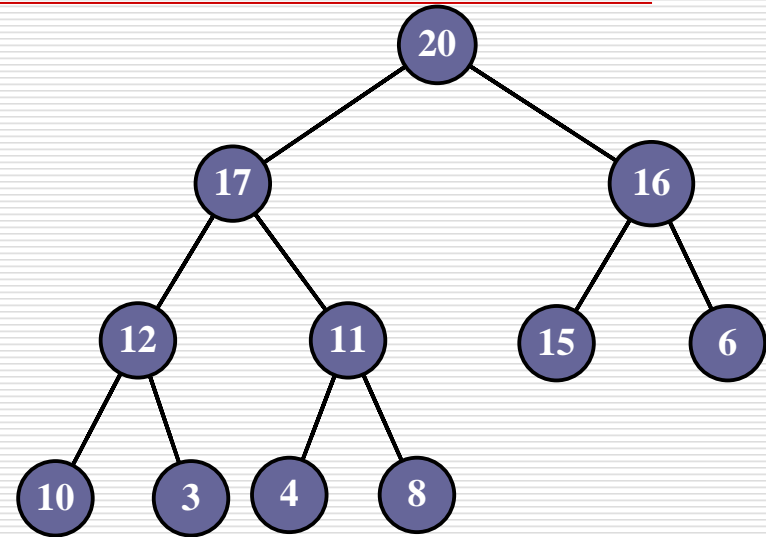
$n / 4$ στοιχεία χρόνος $1 \times c$

$n / 8$ στοιχεία χρόνος $2 \times c$

.....
 $n / 2^k$ στοιχεία χρόνος $(k-1) \times c, k \leq \log_2 n$

- $\sum_{k=2}^{\log n} \frac{n \cdot k \cdot c}{2^k} = O\left(n \cdot \sum_{k=2}^{\log n} \frac{k}{2^k}\right) = O(n)$, γιατί $\sum_{k=0}^{\infty} \frac{k}{2^k} = 2$

- Χρόνος $\text{constructHeap}() = \Theta(n)$.



Απόδοση Σωρού

- Χώρος : $\Theta(1)$ (in-place)
- Χρόνοι :
 - createHeap : $\Theta(n)$
 - insert, deleteMax : $O(\log n)$
 - max, size, isEmpty : $\Theta(1)$
- Εξαιρετικά **εύκολη υλοποίηση!**
- Συμπέρασμα:
 - Γρήγορη και ευρύτατα χρησιμοποιούμενη ουρά προτεραιότητας.

Heap-Sort

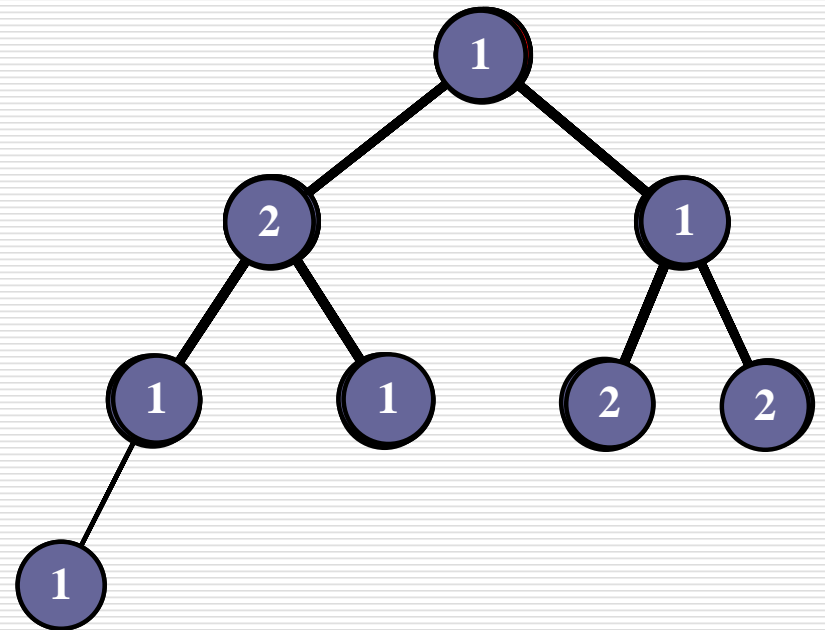
- **Αρχικοποίηση** : δημιουργία **σωρού** με n στοιχεία.
 - `constructHeap()` : χρόνος $\Theta(n)$.
- **Εξαγωγή μέγιστου** και τοποθέτηση **στο τέλος** ($n - 1$ φορές).
 - `deleteMax()` : χρόνος $\Theta(\log n)$.
- **Χρόνος** : $\Theta(n) + n \Theta(\log n) = \Theta(n \log n)$.

```
hs = n;
constructHeap(n);
for (i = n; i > 1; i--) {
    swap(A[1], A[i]); hs--;
    combine(1); }
```

- Χρονική Πολυπλοκότητα Ταξινόμησης: $O(n \log n)$.

Heap-Sort : Παράδειγμα

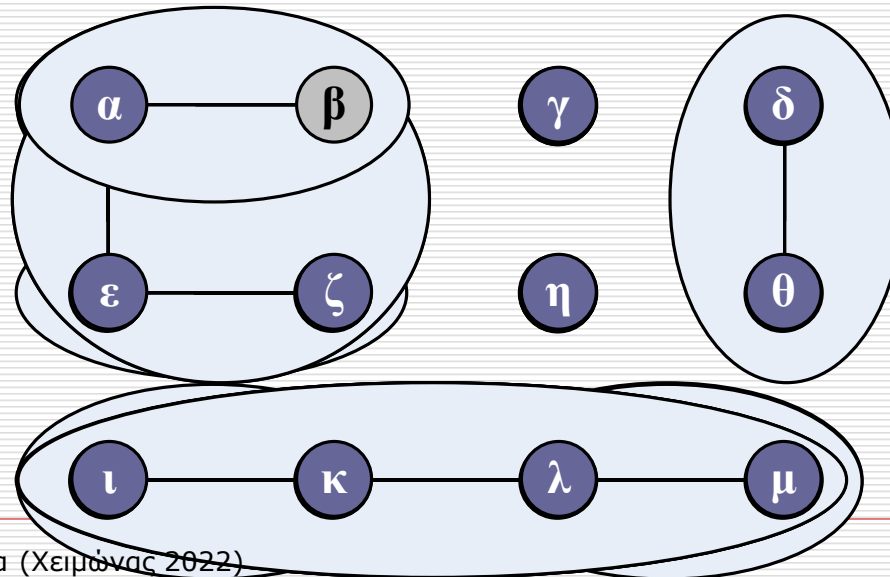
```
constructHeap(n);  
for (i = n; i > 1; i--) {  
    swap(A[1], A[i]); hs--;  
    combine(1); }  
}
```



Union – Find

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.

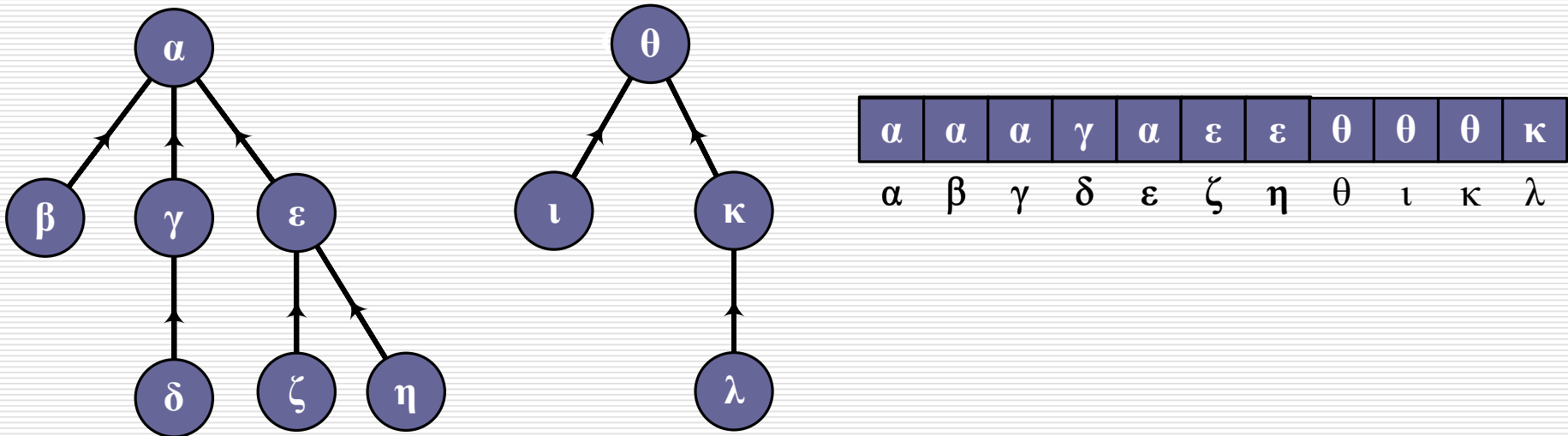


Πρόβλημα Union – Find

- 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.

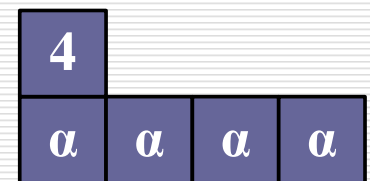
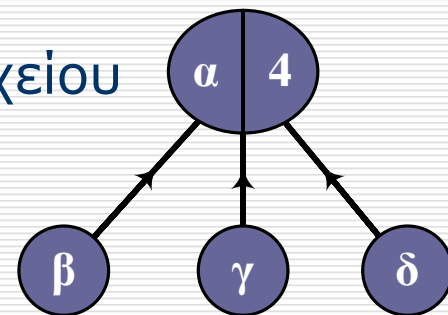
Αναπαράσταση Δέντρου & Δάσους

- Πίνακας γονέων $A[1..n]$ για δέντρο με ρίζα και n κόμβους:
 - $A[i] = j$ αν j πατέρας του i στο δέντρο.
 - $A[\text{ρίζας}] = \text{ρίζα}$ (ή -1).
 - Όμοια για δάσος όπου κάθε δέντρο έχει ρίζα.



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**.

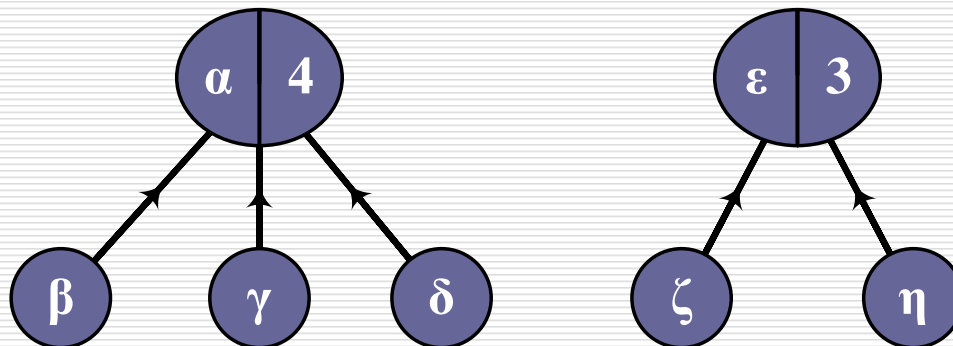


Αναπαράσταση με Δέντρα

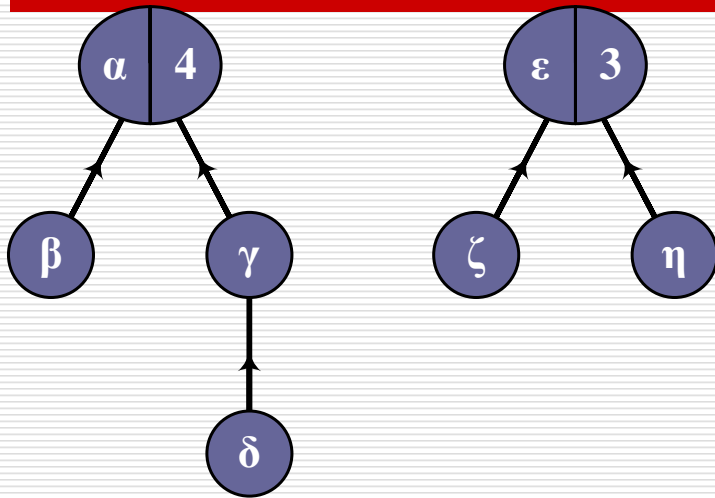
- `find(x)`: ακολουθούμε δείκτες σε γονέα μέχρι τη ρίζα.

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

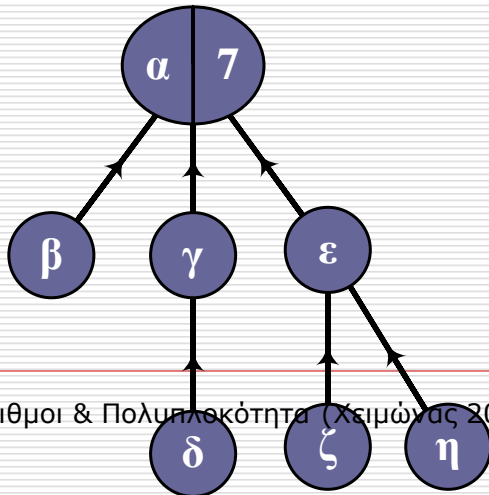
- `union(x, y)`: x και y αντιπρόσωποι διαφορετικών συνόλων
 - Συνένωση δέντρων: ρίζα 1^{ου} συν. γίνεται γονέας ρίζας 2^{ου} συν.
 - Ενημέρωση μεγέθους



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: $\text{ύψος} = n - 1$
 - union($n-1, n$), union($n-2, n-1$), union($n-3, n-2$), union($n-4, n-3$), ..., union($3, 2$), union($1, 2$).
- Απλή δομή, εύκολη υλοποίηση, αλλά ακριβό find!

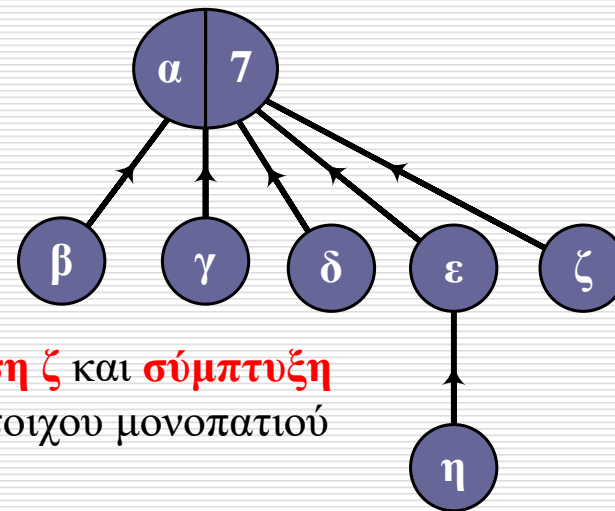
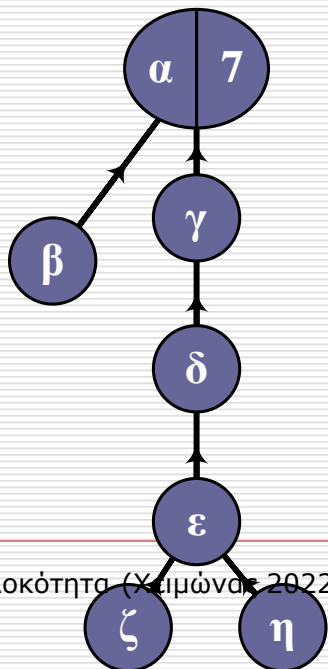
Βεβαρυμένη Ένωση

(Union By Rank / Union By Size)

- «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** όταν στοιχεία **μακριά από ρίζα**.
- Σύμπτυξη μονοπατιού όταν $\text{find}(x)$:
 - 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**.

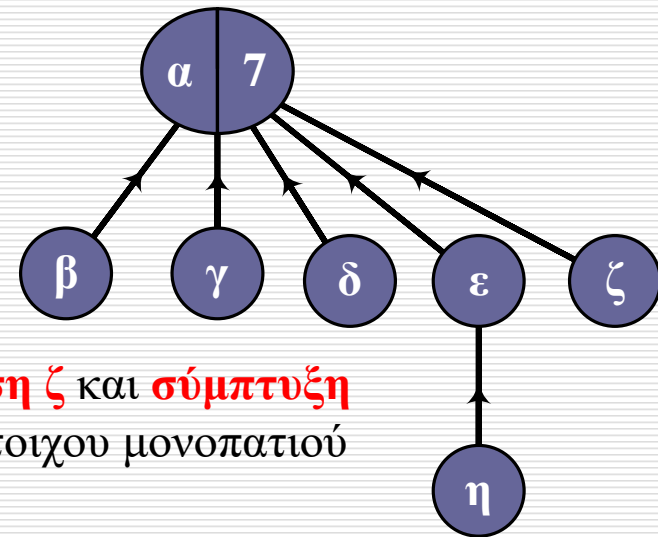
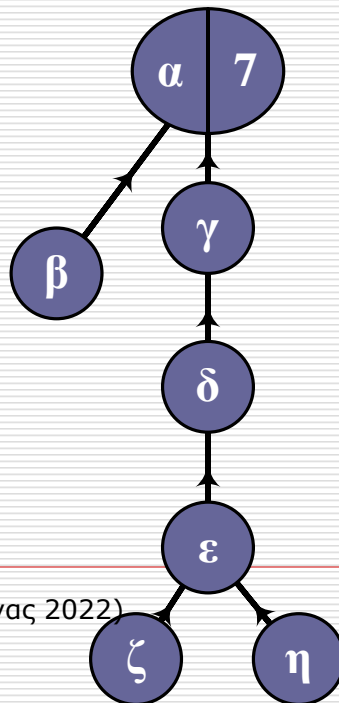


Εύρεση ζ και σύμπτυξη
αντίστοιχου μονοπατιού

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.



Εύρεση ζ και σύμπτυξη
αντίστοιχου μονοπατιού

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!