Δομές Δεδομένων

Μέρος 7ο: Δυαδικά Δένδρα Αναζήτησης

Εξάμηνο Σπουδών: 6ο

Κωδικός Μαθήματος: 681

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

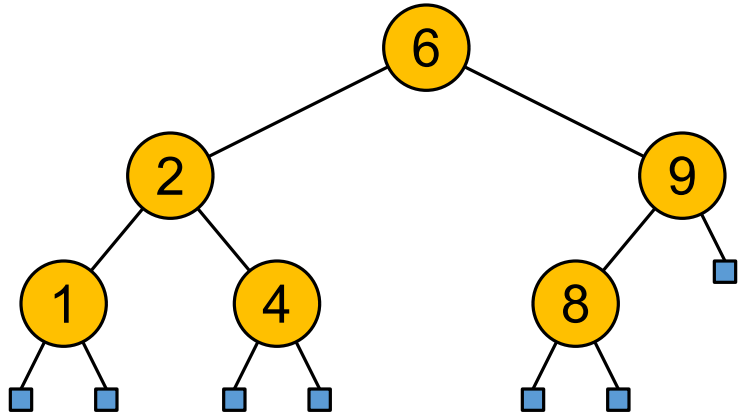
Δυαδικά Δένδρα Αναζήτησης

Binary Search Trees

- Ένα δυαδικό δένδρο αναζήτησης είναι ένα δυαδικό δένδρο το οποίο αποθηκεύει κλειδιά (ή καταχωρήσεις κλειδιών-τιμών) στους εσωτερικούς του κόμβους και ικανοποιεί την παρακάτω ιδιότητα:

  Έστω `u`, `v` και `w` τρεις κόμβοι, τέτοιοι ώστε ο `u` είναι στο αριστερό υποδένδρο του `v` και ο `w` είναι στο δεξί υποδένδρο του `v`. Τότε: `key(u) < key(v) < key(w)`

- Παράδειγμα:

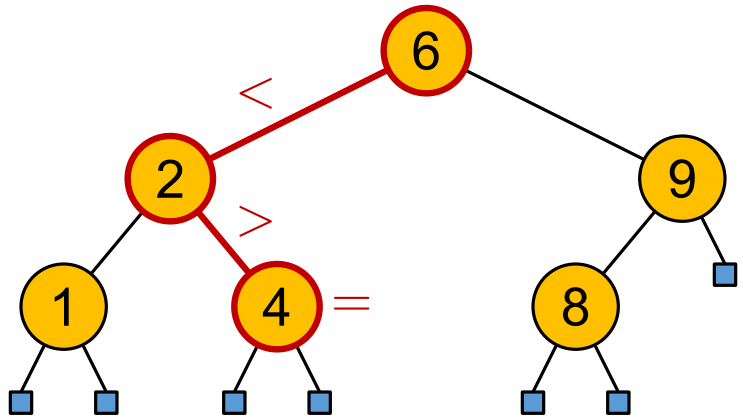


- Παρατηρήσεις:
  - Οι εξωτερικοί κόμβοι δεν αποθηκεύουν καταχωρήσεις
  - Μία inorder διαπέραση του δυαδικού δένδρου αναζήτησης επισκέπτεται τα κλειδιά σε αύξουσα διάταξη

Δυαδικά Δένδρα Αναζήτησης: Αναζήτηση στοιχείου

- Για την αναζήτηση του κλειδιού `k`, ιχνιλατούμε ένα μονοπάτι από τη ρίζα προς τα φύλλα
- Ο επόμενος κόμβος που επισκεπτόμαστε εξαρτάται από τη σύγκριση του `k` με το κλειδί του τρέχοντος κόμβου
- Εάν φτάσουμε σε ένα φύλλο, το κλειδί δεν βρέθηκε

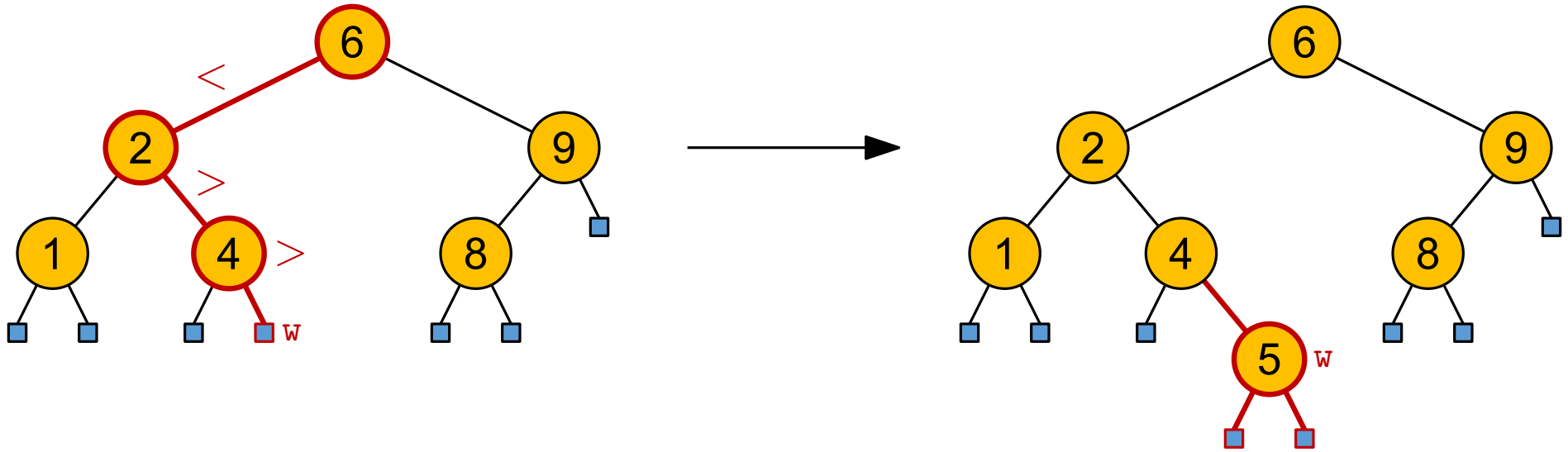
- Παράδειγμα: Αναζήτηση του κλειδιού 4




```
1 Algorithm TreeSearch(k, v)
2 |   if v.isExternal() then
3 |   |   return null;
4 |   if k < v.key() then
5 |   |   return TreeSearch(k, v.left());
6 |   else if k > v.key() then
7 |   |   return TreeSearch(k, v.right());
8 |   return v;
```

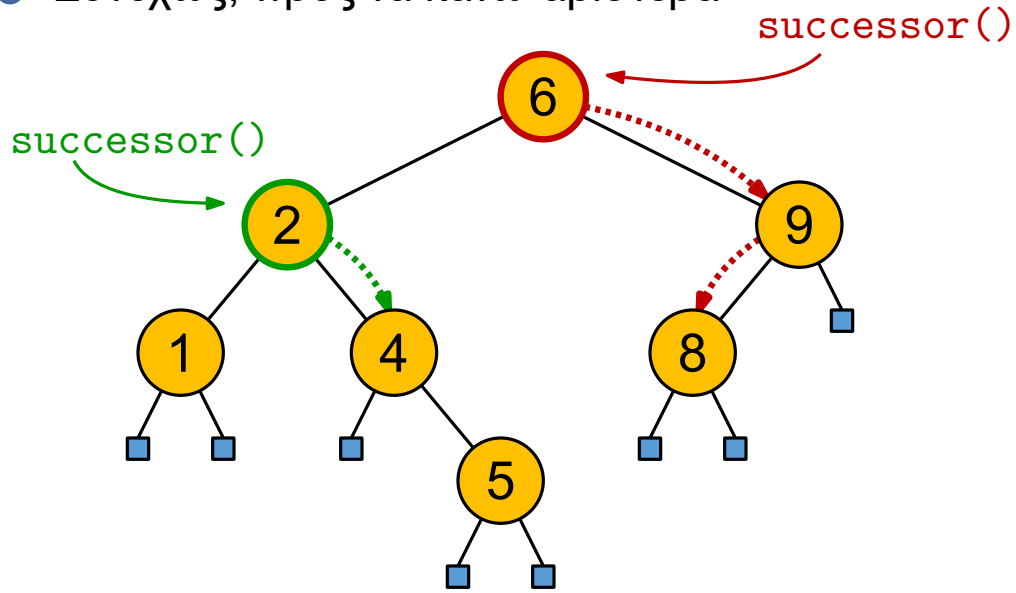
Δυαδικά Δένδρα Αναζήτησης: Εισαγωγή Στοιχείου

- Για την εισαγωγή μιας εγγραφής `(k, d)` κάνουμε μια αναζήτηση για το κλειδί `k` (χρησιμοποιώντας τον αλγόριθμο `TreeSearch`)
- Έστω ότι το στοιχείο `k` δεν υπάρχει ήδη στο δένδρο και ότι το `w` είναι το φύλλο που καταλήγει η (αποτυχημένη) αναζήτηση του `k`
- Εισάγουμε το `k` στον κόμβο `w` και επεκτείνουμε τον `w` σε εσωτερικό κόμβο
- Παράδειγμα: Εισαγωγή του στοιχείου 5

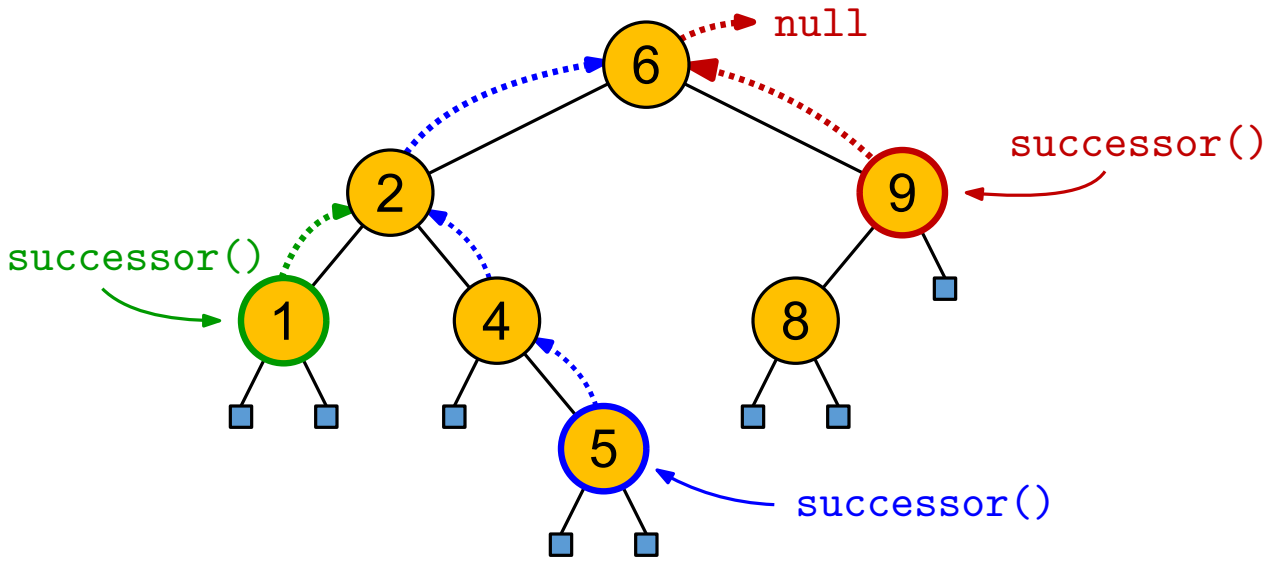


Δυαδικά Δένδρα Αναζήτησης: Εύρεση επόμενου στοιχείου

- Έστω ο κόμβος `v` ο οποίος περιέχει το κλειδί `k`. Θέλουμε να βρούμε τον κόμβο που περιέχει το αμέσως επόμενο σε μέγεθος κλειδί.
- Ο ζητούμενος κόμβος `w` είναι ο κόμβος που ακολουθεί τον `v` στην inorder διαπέραση
- **Περίπτωση-1:** Το δεξί παιδί του κόμβου `v` δεν είναι φύλλο.
  - Ένα βήμα προς τα κάτω-δεξιά
  - Συνεχώς, προς τα κάτω-αριστερά

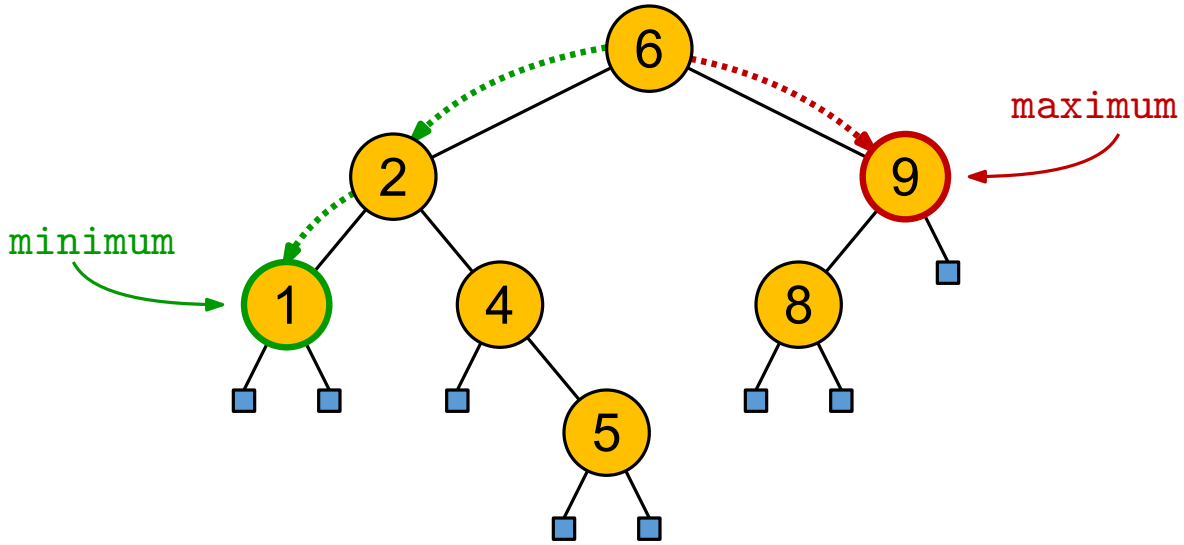


- **Περίπτωση-2:** Το δεξί παιδί του κόμβου `v` είναι φύλλο.
  - Συνεχώς, προς τα πάνω-αριστερά
  - Ένα βήμα προς τα πάνω-δεξιά (εάν αδύνατον, return null)



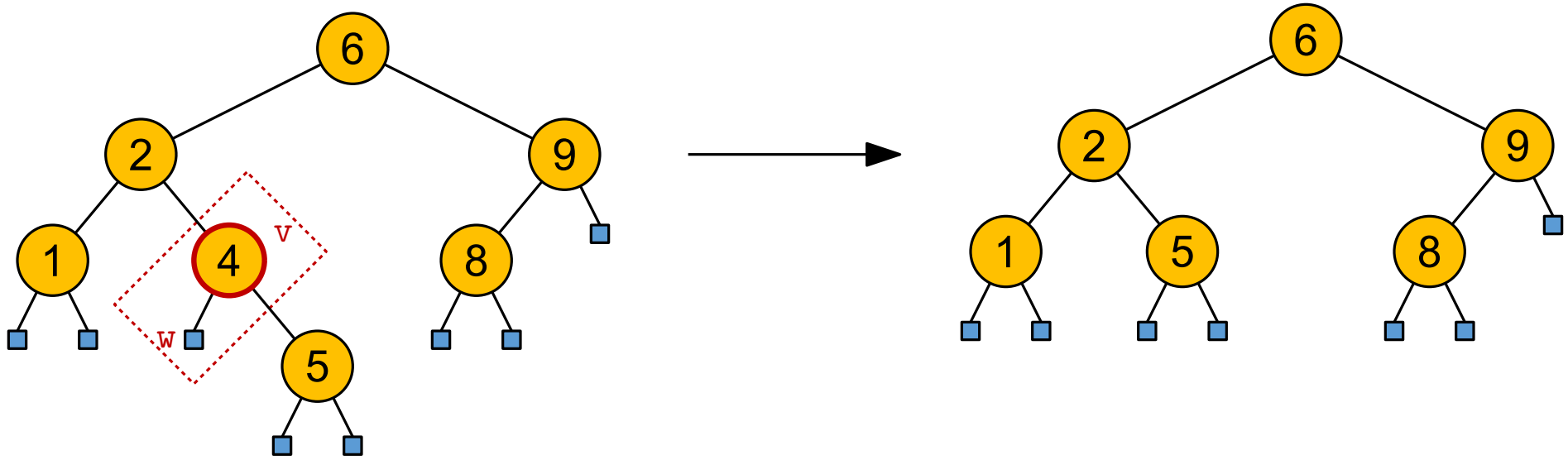
Δυαδικά Δένδρα Αναζήτησης: Εύρεση μέγιστου / ελάχιστου

- Εύρεση προηγούμενου στοιχείου: Συμμετρικά με την εύρεση του επόμενου στοιχείου
- Εύρεση μικρότερου στοιχείου:
  Πήγαινε στη ρίζα $\rightarrow$ συνεχώς, κάτω-αριστερά
- Εύρεση μέγιστου στοιχείου:
  Πήγαινε στη ρίζα $\rightarrow$ συνεχώς, κάτω-δεξιά
- Παράδειγμα:



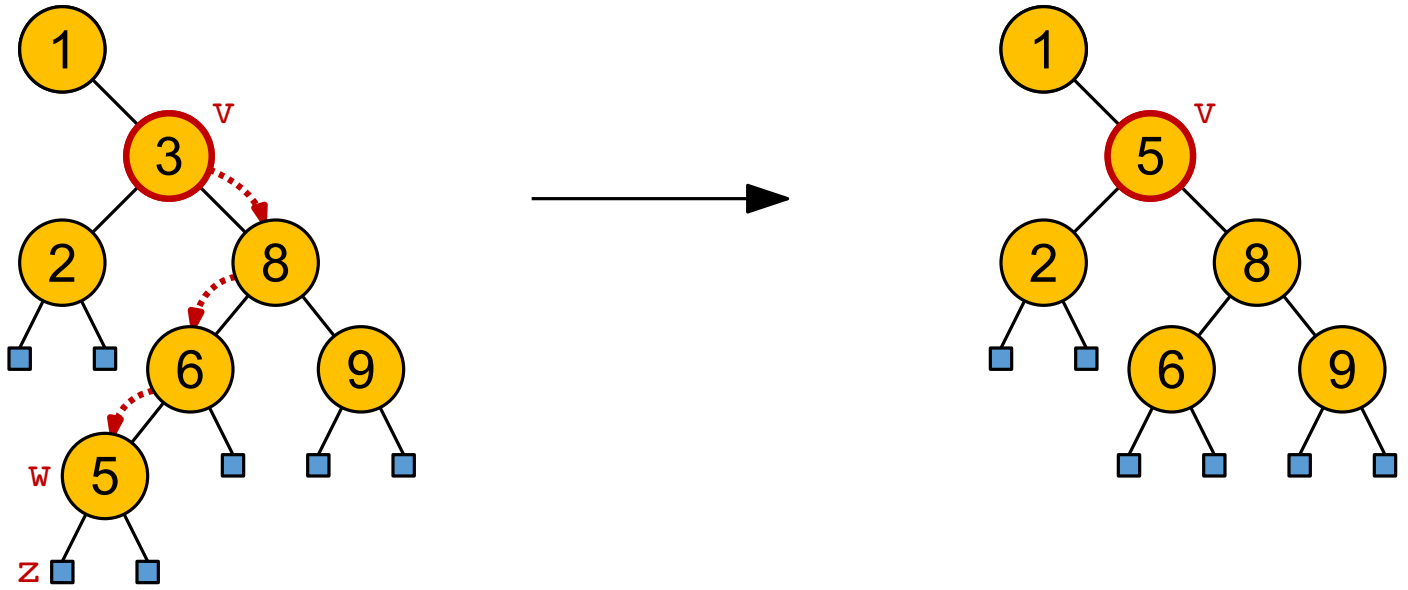
Δυαδικά Δένδρα Αναζήτησης: Διαγραφή στοιχείου

- Για τη διαγραφή μιας εγγραφής με κλειδί `k`, κάνουμε μια αναζήτηση για το κλειδί `k`.
- Έστω ότι η εγγραφή με κλειδί `k` είναι αποθηκευμένη στον κόμβο `v`
- **Περίπτωση-1:** : Ο κόμβος `v` έχει ως παιδί ένα φύλλο `w`
  - Διαγράφουμε τους `v` και `w` από το δένδρο
  - Συνδέουμε τον πατέρα του `v` με το άλλο παιδί του `v`
- Παράδειγμα: Διαγραφή του στοιχείου 4



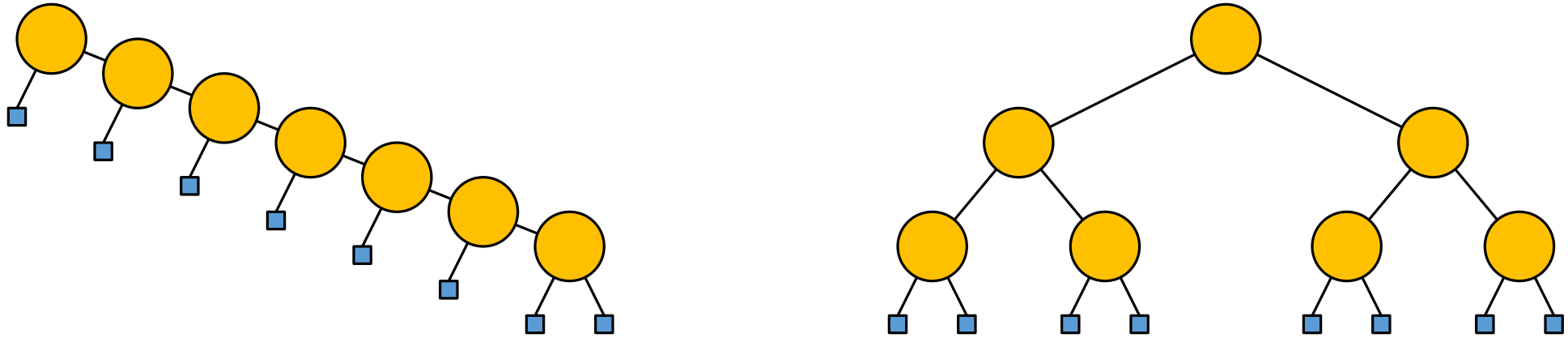
Δυαδικά Δένδρα Αναζήτησης: Διαγραφή στοιχείου

- Για τη διαγραφή μιας εγγραφής με κλειδί `k`, κάνουμε μια αναζήτηση για το κλειδί `k`.
- Έστω ότι η εγγραφή με κλειδί `k` είναι αποθηκευμένη στον κόμβο `v`
- **Περίπτωση-2:** : Τα παιδιά του κόμβου `v` είναι (και τα δύο) εσωτερικοί κόμβοι
  - Βρίσκουμε τον εσωτερικό κόμβο `w` οποίος ακολουθεί τον `v` σε μια inorder διαπέραση του δένδρου
  - Αντιγράφουμε το κλειδί του `w` στον κόμβο `v` και αφαιρούμε τον κόμβο `w` και το αριστερό παιδί (φύλλο) `z` αυτού
- Παράδειγμα: Διαγραφή του στοιχείου 3



Δυαδικά Δένδρα Αναζήτησης: Απόδοση

- Υποθέτουμε ένα δυαδικό δένδρο αναζήτησης με n εγγραφές και ύψος h
- Ο χώρος που χρησιμοποιείται είναι $O(n)$
- Η εισαγωγή, διαγραφή, εύρεση επόμενου/προηγούμενου, ελαχίστου και μεγίστου ολοκληρώνονται σε $O(h)$ χρόνο
- Το ύψος h είναι $O(n)$ στη χειρότερη περίπτωση και $O(\log n)$ στη καλύτερη περίπτωση



- Σημείωση: Όταν τα στοιχεία εισάγονται σε τυχαία σειρά, το αναμενόμενο ύψος h είναι $O(\log n)$

ΑΤΔ: Ουρά Προτεραιότητας

	Unsorted list	Sorted list	Heap	Binary Search Tree	
				worst case	on average
`empty()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`size()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`insert(e)`	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
`removeMin()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
`min()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

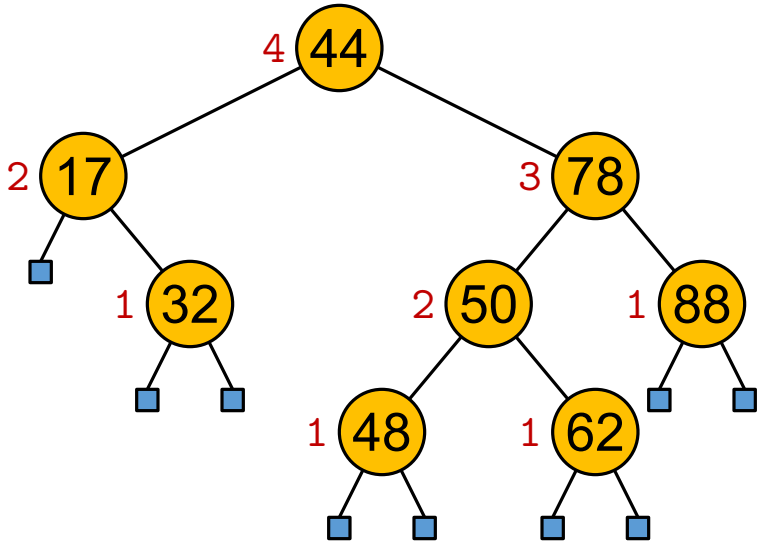
AVL Δένδρα

AVL Trees

- Ένα AVL δένδρο είναι ένα δυαδικό δένδρο αναζήτησης, τέτοιο ώστε για κάθε εσωτερικό κόμβο του v ισχύει ότι τα ύψη των υπο-δένδρων του v διαφέρουν το πολύ κατά 1.

- Τα AVL δένδρα είναι “ζυγισμένα”

- Παράδειγμα AVL δέντρου (τα ύψη εμφανίζονται δίπλα στους κόμβους)



Το ύψος AVL δένδρου

- Θεώρημα: Το ύψος ενός AVL δένδρου το οποίο έχει αποθηκευμένα n κλειδιά είναι $O(\log n)$.

  Απόδειξη:

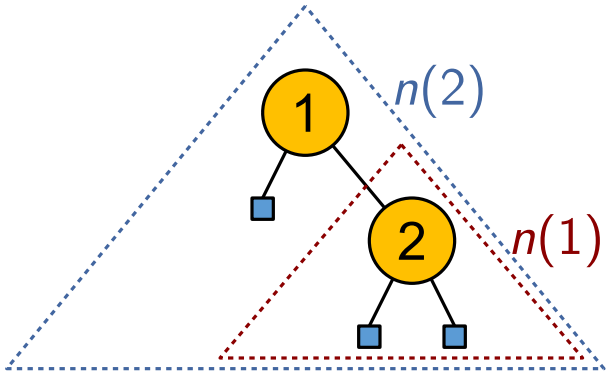
  $n(h) \leftarrow$ ο ελάχιστος αριθμός εσωτερικών κόμβων ενός AVL δένδρο ύψους h.

  Ισχυρισμός: $n(h) > 2^{h/2-1}$

  - Εύκολα βλέπουμε ότι $n(1) = 1$ και $n(2) = 2$
  - Για $h > 2$, ισχύει*: $n(h) = 1 + n(h-1) + n(h-2)$

    Προφανώς $n(h-1) > n(h-2)$, οπότε: $n(h) > 2 \cdot n(h-2)$.

$$\left.\begin{array}{l} n(h) > 2^1 \cdot n(h-2) \\ n(h) > 2^2 \cdot n(h-4) \\ \ldots \\ n(h) > 2^i \cdot n(h-2i) \end{array}\right\} \quad n(h) > 2^{h/2-1} \quad \Rightarrow \quad \boxed{h < 2 \cdot \log n(h) + 2}$$



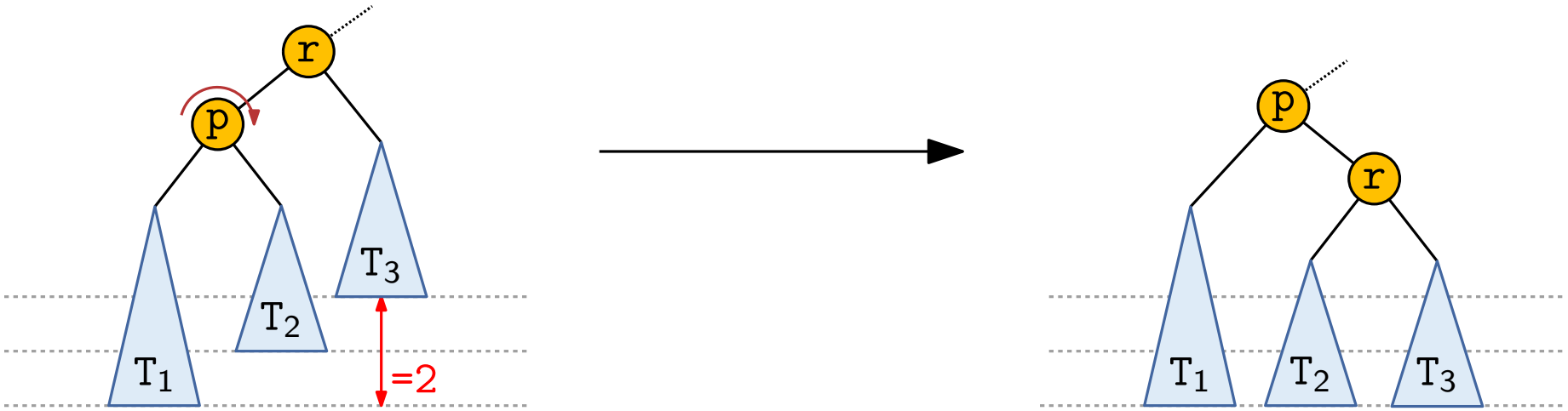
* Ένα AVL δέντρο ύψους h περιέχει τον κόμβο-ρίζα, ένα AVL υποδέντρο ύψους $h-1$ και ένα άλλο ύψους $h-2$

AVL Δένδρα: Αναδιάταξη μέσω περιστρεφών

AVL Trees

- Περιστροφή: Τοπικός μετασχηματισμός γύρω από έναν κόμβο χωρίς να αλλάζει η inorder διαπέραση των στοιχείων.
  Rotation

- Απλές περιστροφές: Δεξιά περιστροφή γύρω από τον κόμβο `p`

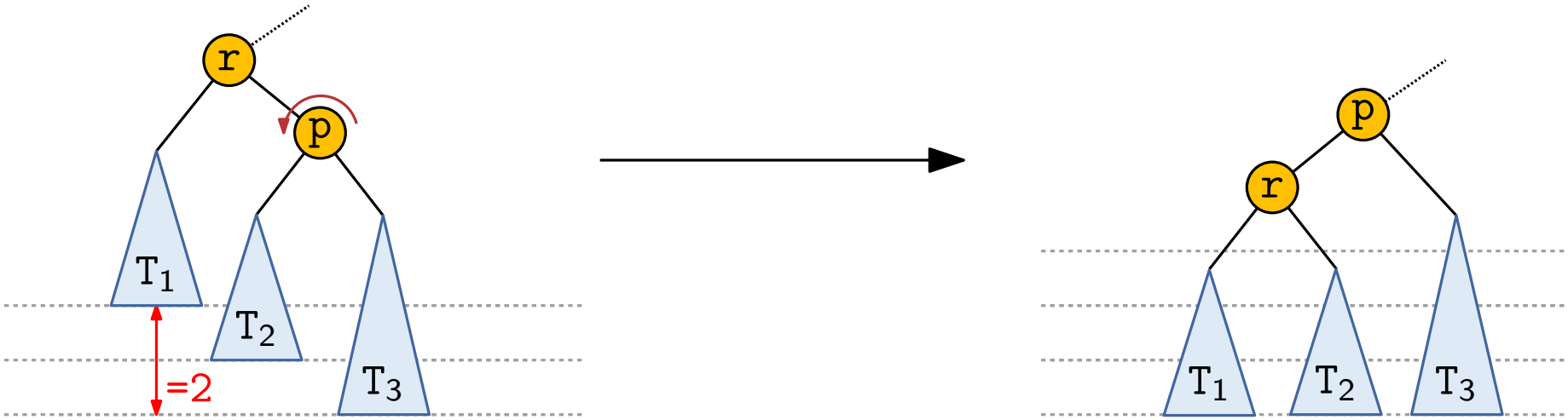


AVL Δένδρα: Αναδιάταξη μέσω περιστρεφών

AVL Trees

- Περιστροφή: Τοπικός μετασχηματισμός γύρω από έναν κόμβό χωρίς να αλλάζει η inorder διαπέραση των στοιχείων.
  Rotation

- Απλές περιστροφές: Αριστερή περιστροφή γύρω από τον κόμβο `p`

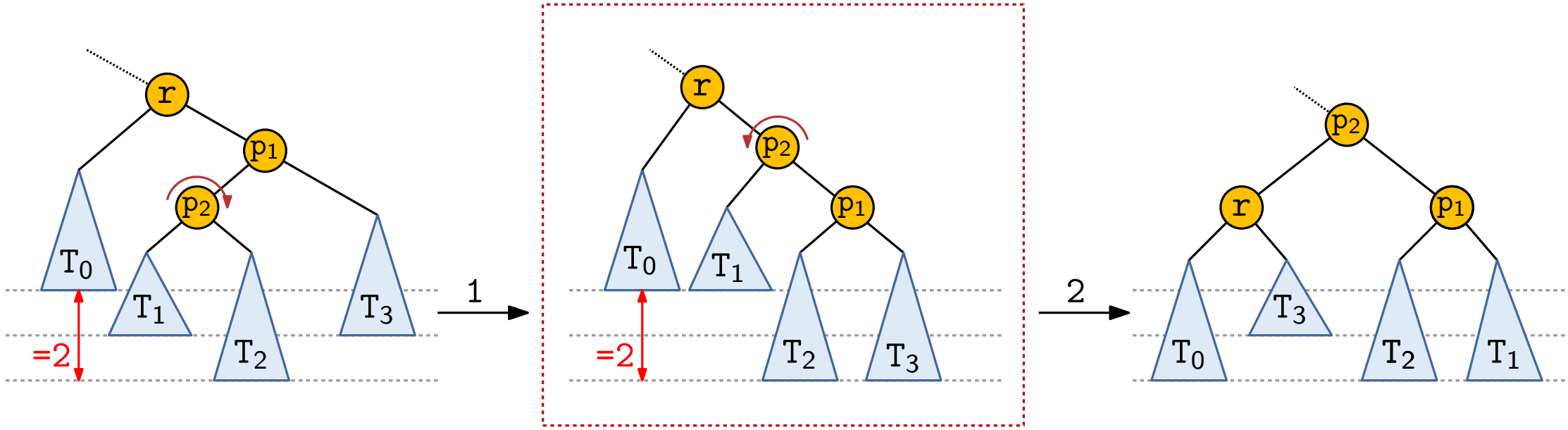


AVL Δένδρα: Αναδιάταξη μέσω περιστρεφών

AVL Trees

- **Περιστροφή:** Τοπικός μετασχηματισμός γύρω από έναν κόμβό χωρίς να αλλάζει η inorder διαπέραση των στοιχείων.
  Rotation

- **Διπλές περιστροφές:** Διπλή περιστροφή γύρω από τους κόμβους p_1 και p_2
  [πρώτα “δεξιά”, μετά “αριστερά”]

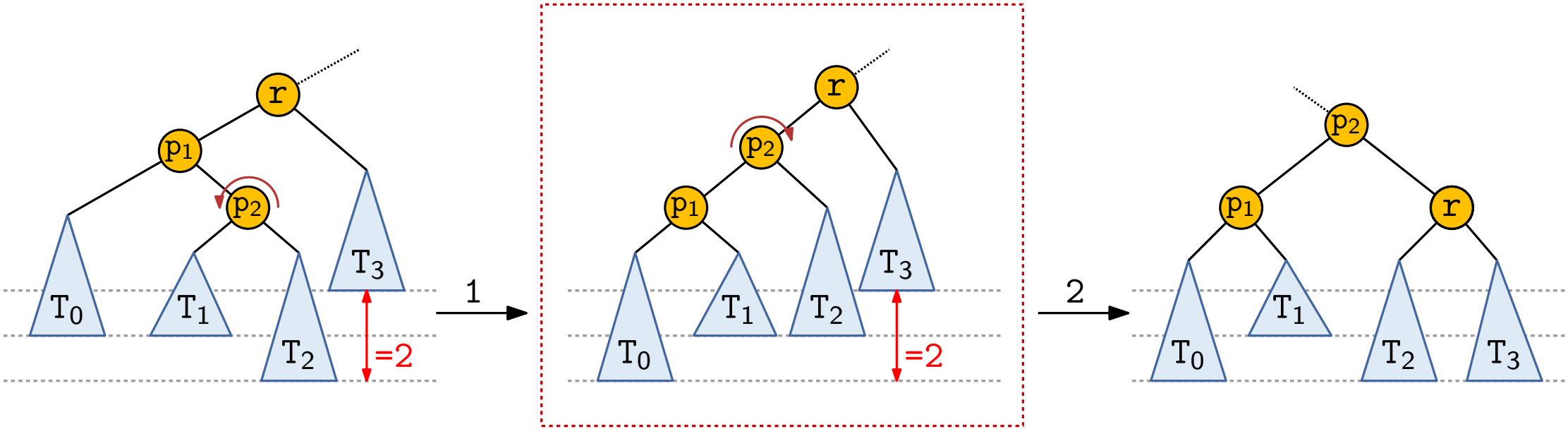


AVL Δένδρα: Αναδιάταξη μέσω περιστρεφών

AVL Trees

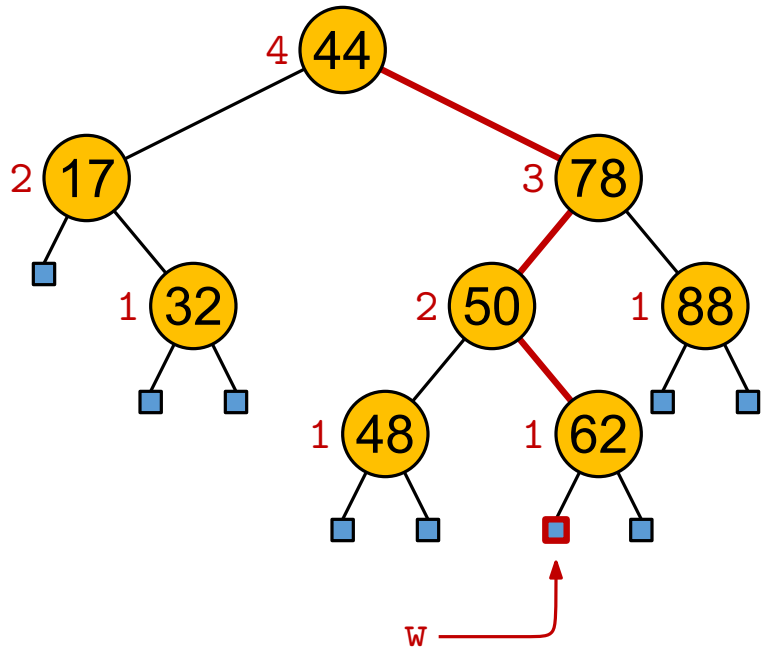
- Περιστροφή: Τοπικός μετασχηματισμός γύρω από έναν κόμβό χωρίς να αλλάζει η inorder διαπέραση των στοιχείων.
  Rotation

- Διπλές περιστροφές: Διπλή περιστροφή γύρω από τους κόμβους p_1 και p_2
  [πρώτα “αριστερά”, μετά “δεξιά”]



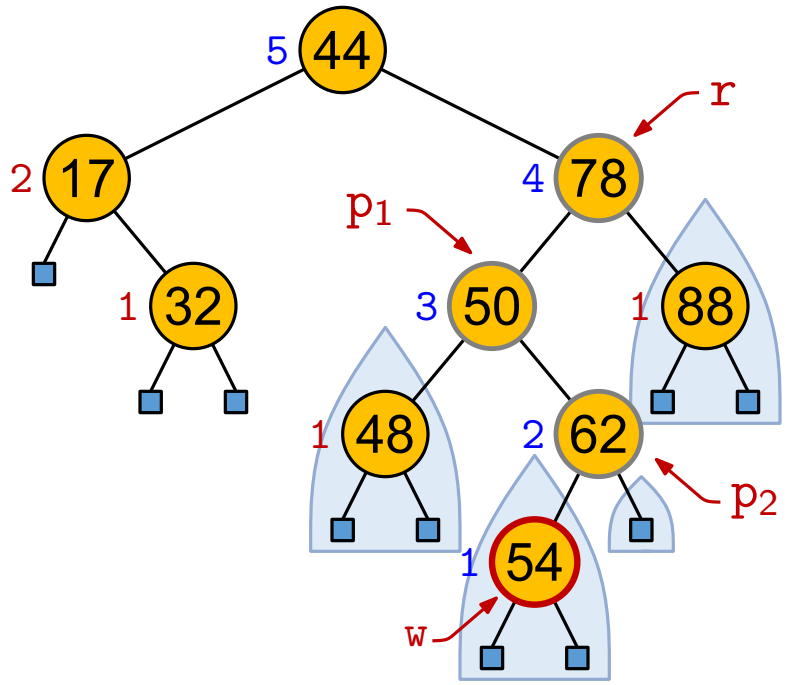
AVL Δένδρα: Εισαγωγή Στοιχείου

- Η εισαγωγή είναι ίδια όπως και στα δυαδικά δένδρα αναζήτησης.
- Πάντα ολοκληρώνεται με μία “επέκταση” ενός εξωτερικού κόμβου.
- Παράδειγμα: Εισαγωγή του στοιχείου 54



AVL Δένδρα: Εισαγωγή Στοιχείου

- Η εισαγωγή είναι ίδια όπως και στα δυαδικά δένδρα αναζήτησης.
- Πάντα ολοκληρώνεται με μία “επέκταση” ενός εξωτερικού κόμβου.
- Παράδειγμα: Εισαγωγή του στοιχείου 54



r: Ο πιο κοντινός μη-ισοζυγισμένος πρόγονος του w

p_1: Το παιδί του r το οποίο είναι πρόγονος του w

p_1: Το παιδί του p_1 το οποίο είναι πρόγονος του w

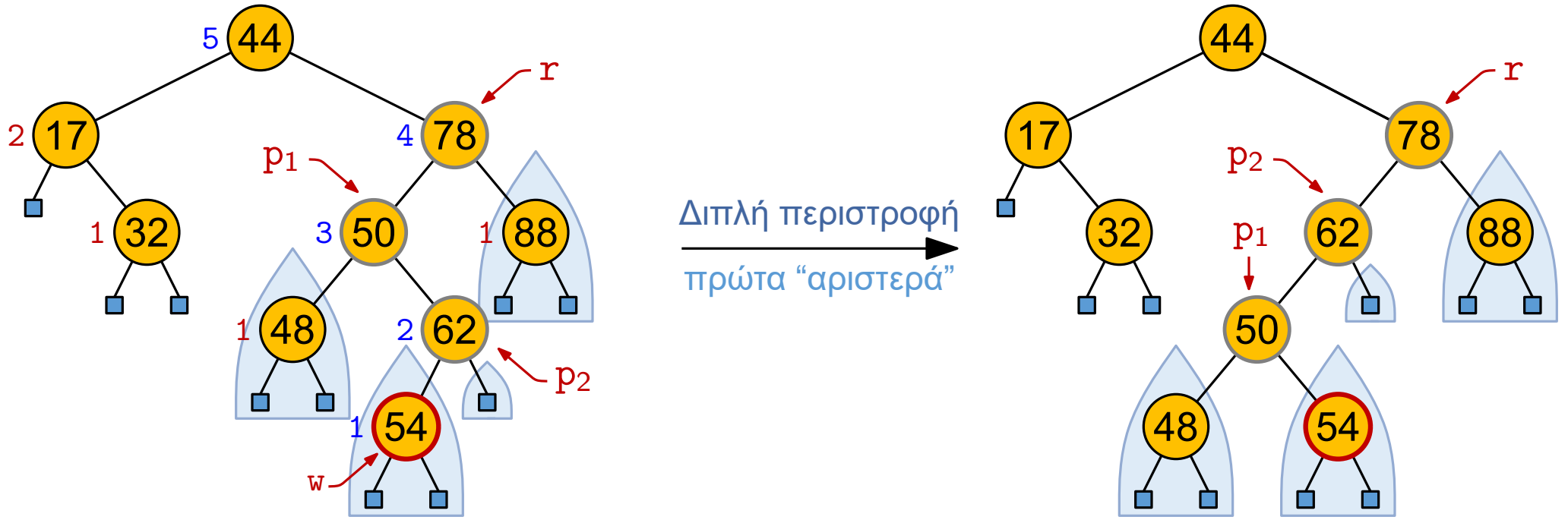
AVL Δένδρα: Εισαγωγή Στοιχείου

- Η εισαγωγή είναι ίδια όπως και στα δυαδικά δένδρα αναζήτησης.
- Πάντα ολοκληρώνεται με μία “επέκταση” ενός εξωτερικού κόμβου.
- Παράδειγμα: Εισαγωγή του στοιχείου 54

`r`: Ο πιο κοντινός μη-ισοζυγισμένος πρόγονος του `w`

p_1: Το παιδί του `r` το οποίο είναι πρόγονος του `w`

p_1: Το παιδί του p_1 το οποίο είναι πρόγονος του `w`



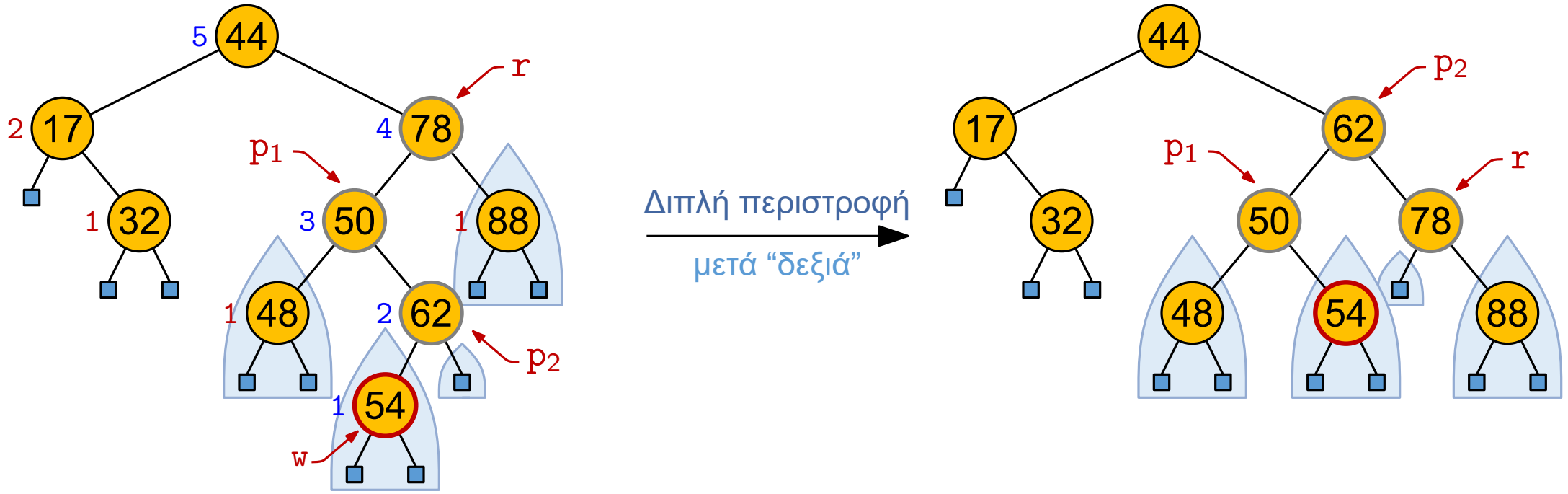
AVL Δένδρα: Εισαγωγή Στοιχείου

- Η εισαγωγή είναι ίδια όπως και στα δυαδικά δένδρα αναζήτησης.
- Πάντα ολοκληρώνεται με μία “επέκταση” ενός εξωτερικού κόμβου.
- Παράδειγμα: Εισαγωγή του στοιχείου 54

`r`: Ο πιο κοντινός μη-ισοζυγισμένος πρόγονος του `w`

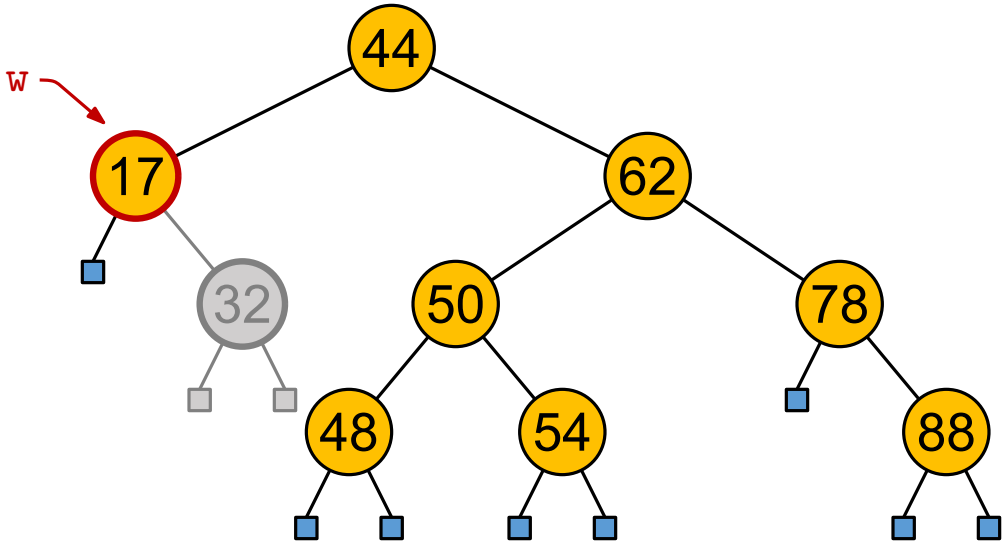
p_1: Το παιδί του `r` το οποίο είναι πρόγονος του `w`

p_1: Το παιδί του p_1 το οποίο είναι πρόγονος του `w`



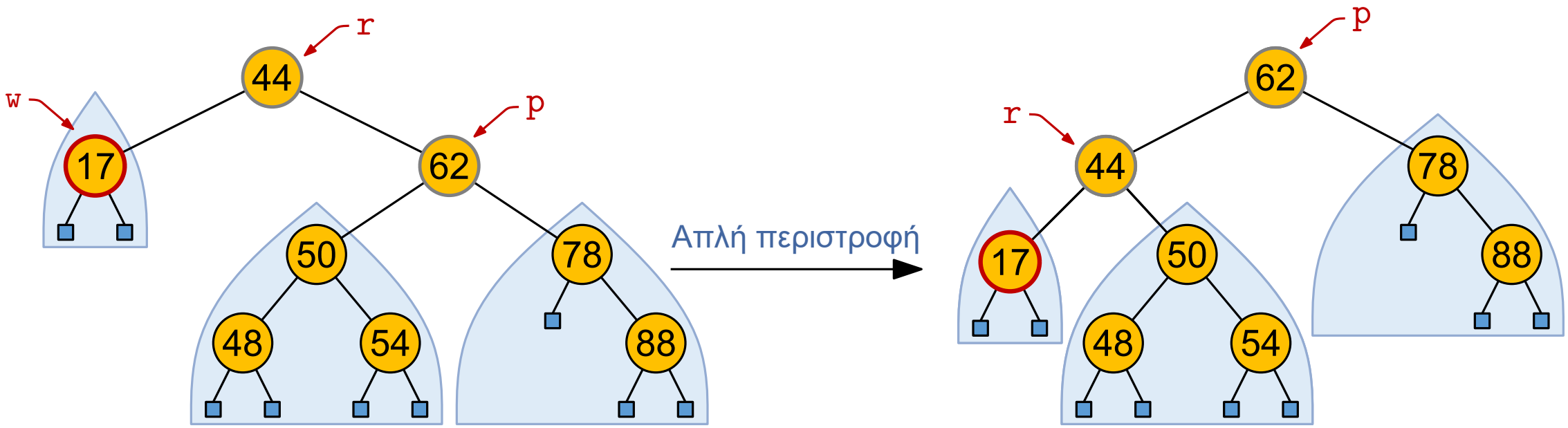
AVL Δένδρα: Διαγραφή Στοιχείου

- Η διαγραφή γίνεται όπως και στα δυαδικά δένδρα αναζήτησης.
- Στον πατέρα, `w`, του κόμβου που διαγράφεται μπορεί να προκληθεί ανισορροπία.
- Παράδειγμα: Διαγραφή του στοιχείου 32



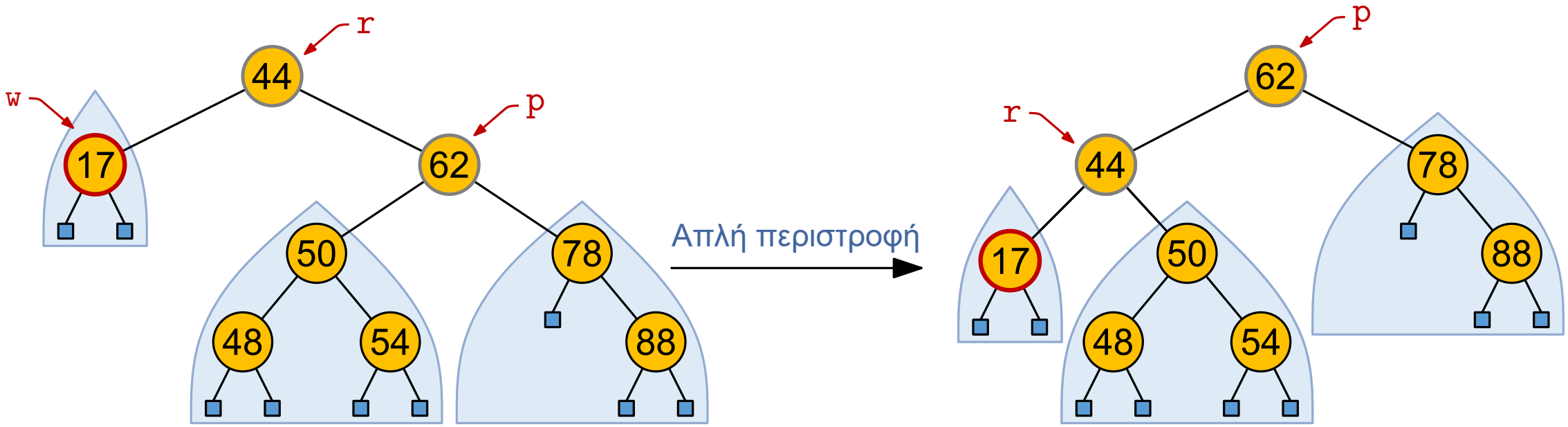
AVL Δένδρα: Διαγραφή Στοιχείου

- Η διαγραφή γίνεται όπως και στα δυαδικά δένδρα αναζήτησης.
- Στον πατέρα, `w`, του κόμβου που διαγράφεται μπορεί να προκληθεί ανισορροπία.
- Παράδειγμα: Διαγραφή του στοιχείου 32



AVL Δένδρα: Διαγραφή Στοιχείου

- Η διαγραφή γίνεται όπως και στα δυαδικά δένδρα αναζήτησης.
- Στον πατέρα, `w`, του κόμβου που διαγράφεται μπορεί να προκληθεί ανισορροπία.
- Παράδειγμα: Διαγραφή του στοιχείου 32



- Σημείωση: Η περιστροφή μπορεί να προκαλέσει ανισορροπία σε άλλο κόμβο ψηλότερα στο δέντρο $\Rightarrow$ Συνεχίζουμε τον έλεγχο της ισορροπίας μέχρι να φτάσουμε στη ρίζα του δένδρου

AVL Δένδρα: Απόδοση

- Ένα AVL δένδρο στο οποίο είναι αποθηκευμένα n στοιχεία χρησιμοποιεί $O(n)$ χώρο
- Μία περιστροφή χρειάζεται $O(1)$ χρόνο
- Η αναζήτηση στοιχείου χρειάζεται $O(\log n)$ χρόνο
  - Δεν είναι απαραίτητη καμία αναδιάταξη
  - Το ύψος του δένδρου είναι $O(\log n)$
- Η εισαγωγή στοιχείου χρειάζεται $O(\log n)$ χρόνο
  - Η αρχική αναζήτηση χρειάζεται $O(\log n)$ χρόνο
  - Απαιτείται μόνο μία αναδιάταξη
- Η διαγραφή στοιχείου χρειάζεται $O(\log n)$ χρόνο
  - Η αρχική αναζήτηση χρειάζεται $O(\log n)$ χρόνο
  - Το πολύ $O(\log n)$ αναδιατάξεις

ΑΤΔ: Ουρά Προτεραιότητας

	Unsorted list	Sorted list	Heap	Binary Search Tree		AVL Tree
				worst case	on average	
`empty()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`size()`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`insert(e)`	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
`removeMin()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
`min()`	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Επιπλέον υλικό

- Ενότητες 3.1, 4.1, 4.2:
  Michael T. Goodrich, Roberto Tamassia, Αλγόριθμοι σχεδίαση και εφαρμογές
  ISBN: 9789605126971, εκδόσεις Γκιούρδα, 2016.