Δομές Δεδομένων

Δένδρα Αναζήτησης

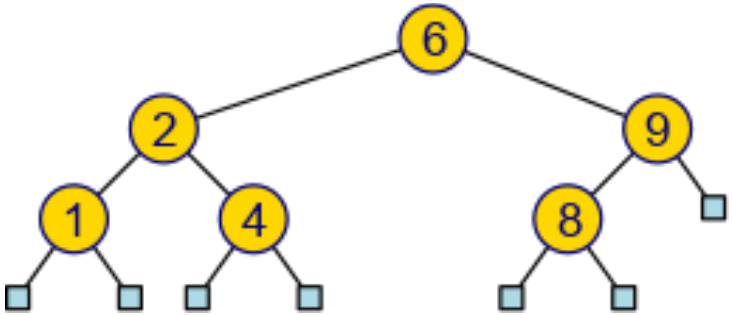
Α. Συμβώνης
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών
Τομέας Μαθηματικών

Δυαδικά Δένδρα Αναζήτησης
[Binary Search Trees]

Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

Δυαδικό Δένδρο Αναζήτησης [Binary Search Tree - BST]

- 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 ότι οι u, v και w είναι τρεις κόμβοι τέτοιοι ώστε ο u είναι στο αριστερό υποδένδρο του v και ο w είναι στο δεξί υποδένδρο του v. Τότε έχουμε ότι: $key(u) < key(v) < key(w)$
- Οι εξωτερικοί κόμβοι δεν αποθηκεύουν καταχωρήσεις
- Μία inorder διαπέραση ενός δυαδικού δένδρου αναζήτησης επισκέπτεται τα κλειδιά σε αύξουσα διάταξη



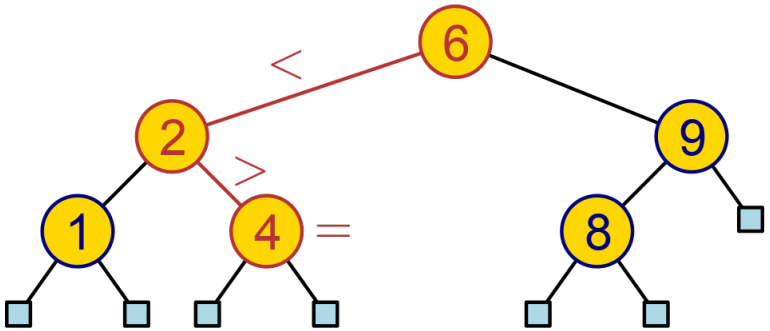
Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- 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

```
Algorithm TreeSearch(k,v):
    //t is the binary search tree
    if t.isExternal(v) then
        return null
    if k < key(v) then
        return TreeSearch(k,left(v))
    else if k == key(v) then
        return v
    else //k > key(v)
        return TreeSearch(k,right(v))
```

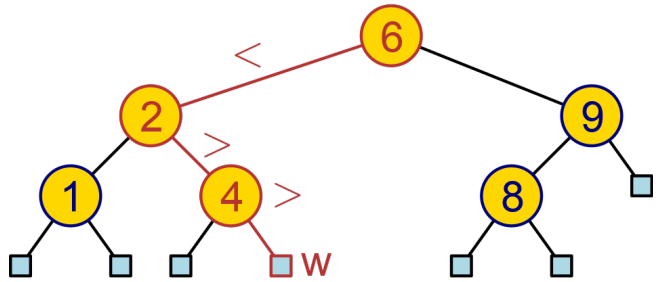
Παράδειγμα

- Αναζήτηση του κλειδιού **4**
  - Κλήση του αλγορίθμου `TreeSearch(4,root)`



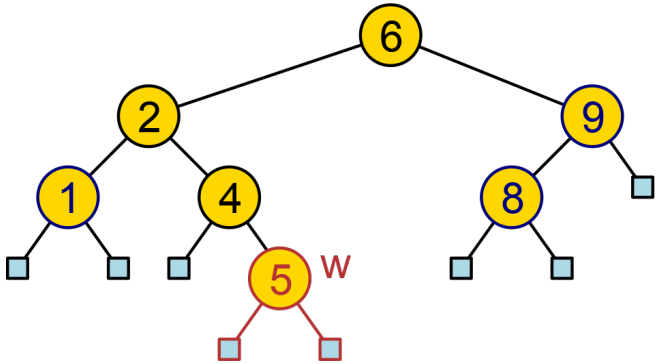
Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- Εισαγωγή στοιχείου
  - Για την εισαγωγή μιας εγγραφής `(k,d)` κάνουμε μια αναζήτηση για το κλειδί `k` (χρησιμοποιώντας τον αλγόριθμο `TreeSearch`)
  - Έστω ότι το `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**

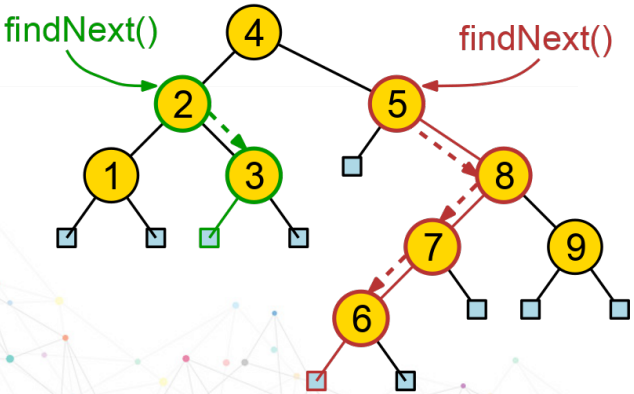


Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- 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

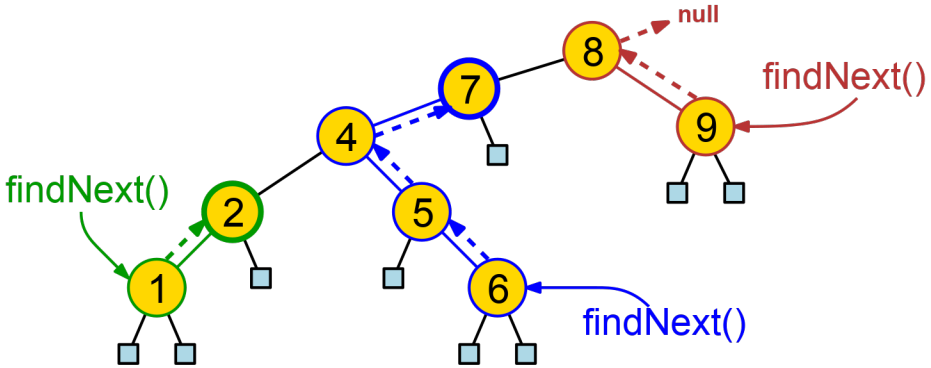
Περίπτωση-1: Το δεξί παιδί του κόμβου v δεν είναι φύλλο.

- 1 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
- 1 βήμα προς τα πάνω-δεξιά (εάν αδύνατον, return null)

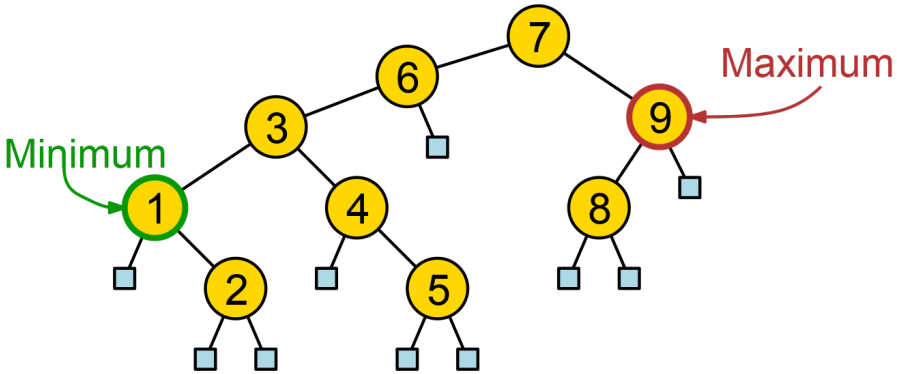


Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- Εύρεση προηγούμενου στοιχείου
  - Συμμετρικά με την εύρεση του επόμενου στοιχείου

- Εύρεση μικρότερου στοιχείου
  - Πήγαινε στη ρίζα
  - Συνεχώς, κάτω-αριστερά

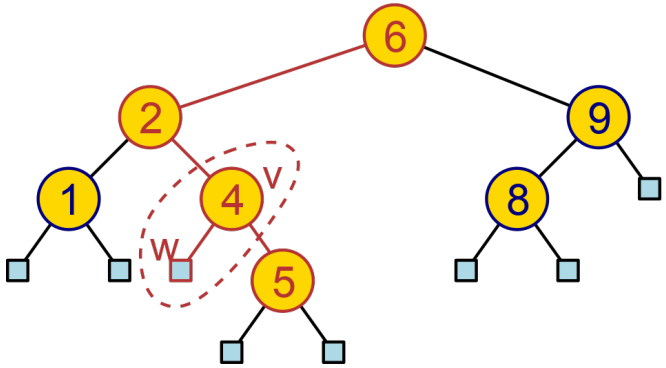
- Εύρεση μεγαλύτερου στοιχείου
  - Πήγαινε στη ρίζα
  - Συνεχώς, κάτω-δεξιά



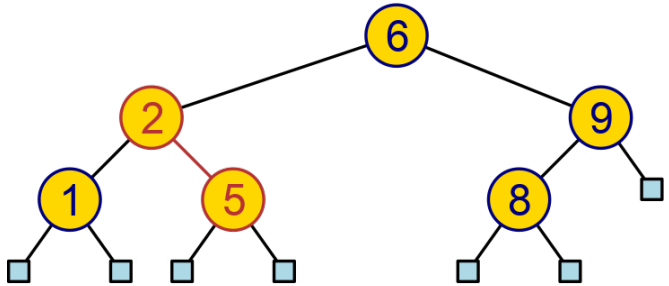
Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- 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 τον `v` και τον `w` από το δένδρο (συνδέοντας τον πατέρα του v με το άλλο παιδί του v)



Παράδειγμα
- Διαγραφή του στοιχείου **4**

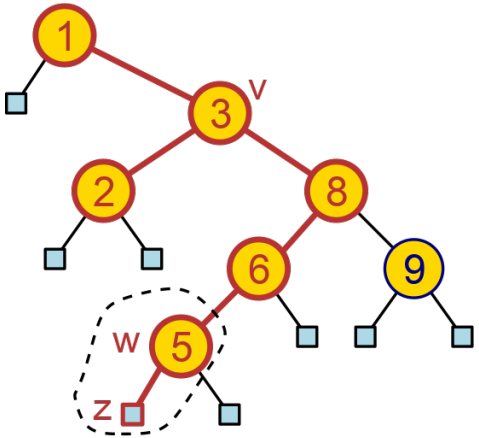


Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- Διαγραφή στοιχείου (*συνέχεια*)

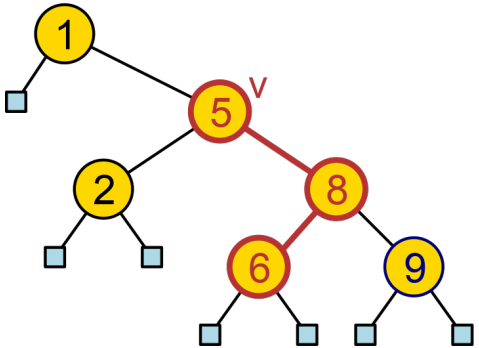
Περίπτωση-2: Τα παιδιά του κόμβου `v` είναι (και τα δύο) εσωτερικοί κόμβοι

- Βρίσκουμε τον εσωτερικό κόμβο `w` οποίος ακολουθεί τον `v` σε μια inorder διαπέραση του δένδρου
- Αντιγράφουμε το κλειδί του `w` στον κόμβο `v`
- Αφαιρούμε τον κόμβο `w` και το αριστερό παιδί του `z` (το οποίο είναι φύλλο)



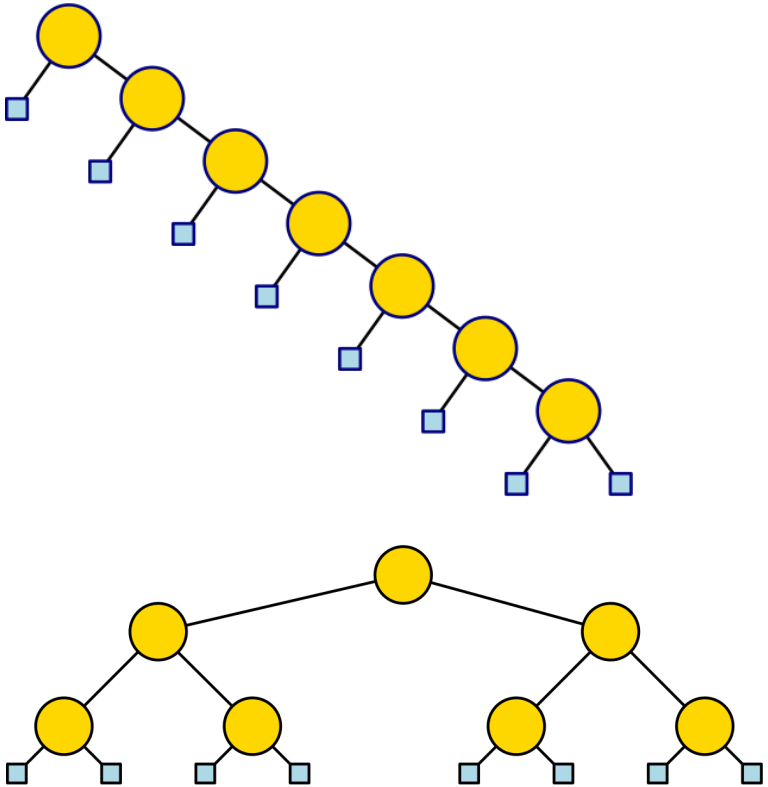
Παράδειγμα

- Αφαίρεση του στοιχείου **3**



Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- Απόδοση
  - Υποθέτουμε ένα δυαδικό δένδρο αναζήτησης με $\boldsymbol{n}$ εγγραφές και ύψος $\boldsymbol{h}$
    - Ο χώρος που χρησιμοποιείται είναι $\boldsymbol{O(n)}$
    - Η εισαγωγή, διαγραφή, εύρεση επόμενου/προηγούμενου, ελαχίστου και μεγίστου ολοκληρώνονται σε $\boldsymbol{O(h)}$ χρόνο
  - Το ύψος $\boldsymbol{h}$ του δένδρου είναι $\boldsymbol{O(n)}$ στη χειρότερη περίπτωση και $\boldsymbol{O(\log n)}$ στη καλύτερη περίπτωση
  - Όταν τα στοιχεία εισάγονται σε τυχαία σειρά, το αναμενόμενο ύψος $\boldsymbol{h}$ του δένδρου είναι $\boldsymbol{O(\log n)}$



Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

- Υλοποίηση SortedMap με Binary Search Tree

ΑΤΔ SortedMap	Binary Search Tree	
	Worst Case	Average Case
`put(k,v)`	$O(n)$	$O(\log n)$
`get(k)`	$O(n)$	$O(\log n)$
`remove(k)`	$O(n)$	$O(\log n)$
`size()`	$O(1)$	$O(1)$
`isEmpty()`	$O(1)$	$O(1)$
`firstEntry(),lastEntry()`	$O(n)$	$O(\log n)$
`ceilingEntry(k), floorEntry(k), lowerEntry(k), higherEntry(k)`	$O(n)$	$O(\log n)$
`subMap(k1,k2)` [s το πλήθος των στοιχείων μεταξύ k1 και k2]	$O(n+s)$	$O(n+\min\{n, s\log n+\log n\}) = O(n)$
`entrySet(), keySet(), values()`	$O(n)$	$O(n)$

Δυαδικά Δένδρα Αναζήτησης [Binary Search Trees]

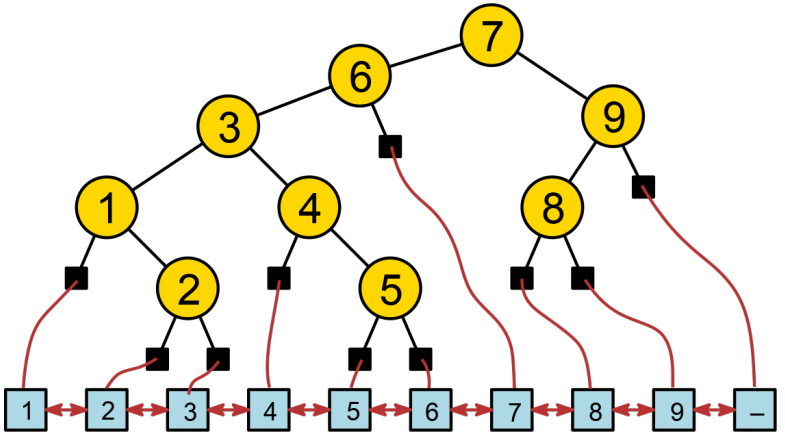
- Υλοποίηση AdaptablePriorityQueue με Binary Search Tree

	Υλοποίηση				
Adaptable Priority Queue	Μη-ταξινομημένη λίστα (PositionalList)	Ταξινομημένη λίστα (PositionalList)	**Σωρός (Heap)**	**Binary Search Tree**	
				worst case	**Average case**
`insert()`	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$	$\mathbf{O(\log n)}$
`removeMin()`	$O(n)$	$O(1)$	$O(\log n)$	$O(n)$	$\mathbf{O(\log n)}$
`min()`	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$\mathbf{O(1)}$
`remove()`	$O(1)$	$O(1)$	$O(\log n)$	$O(n)$	$\mathbf{O(\log n)}$
`replaceKey()` `[decreaseKey()]`	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$	$\mathbf{O(\log n)}$
`replaceValue()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$\mathbf{O(1)}$
`size()` `isEmpty(), isFull`	$O(1)$				

Φύλλο-προσανατολισμένα δυαδικά δένδρα αναζήτησης
[Leaf-oriented BSTs]

Φύλλο-προσανατολισμένα Δυαδικά Δένδρα Αναζήτησης

- Φύλλο-προσανατολισμένα δυαδικά δένδρα αναζήτησης [leaf-oriented BSTs]
  - Αποθηκεύουμε στους εσωτερικούς κόμβους μόνο κλειδιά
  - Αποθηκεύουμε κάθε εγγραφή στο «προηγούμενο φύλλο» του κόμβου που περιέχει το κλειδί.
  - Τα φύλλα είναι συνδεδεμένα σε αλυσίδα
  - Δοσμένης της θέσης [Position] μίας εγγραφής, μπορούμε να προσδιορίσουμε την επόμενη/προηγούμενη εγγραφή στη διάταξη σε $O(1)$ χρόνο.



Φύλλο-προσανατολισμένα Δυαδικά Δένδρα Αναζήτησης

- Υλοποίηση SortedMap με Leaf-oriented BST

ΑΤΔ SortedMap	Binary Search Tree	
	Worst Case	Average Case
`put(k,v)`	$O(n)$	$O(\log n)$
`get(k)`	$O(n)$	$O(\log n)$
`remove(k)`	$O(n)$	$O(\log n)$
`size()`	$O(1)$	$O(1)$
`isEmpty()`	$O(1)$	$O(1)$
`firstEntry(),lastEntry()`	$O(n)$	$\mathbf{O(1)}$
`ceilingEntry(k), floorEntry(k), lowerEntry(k), higherEntry(k)`	$O(n)$	$O(\log n)$
`subMap(k1,k2)` [s το πλήθος των στοιχείων μεταξύ k1 και k2]	$O(n+s)$	$\mathbf{O(s+\log n)}$
`entrySet(), keySet(), values()`	$O(n)$	$O(n)$

Φύλλο-προσανατολισμένα Δυαδικά Δένδρα Αναζήτησης

- Υλοποίηση AdaptablePriorityQueue με Leaf-oriented BST

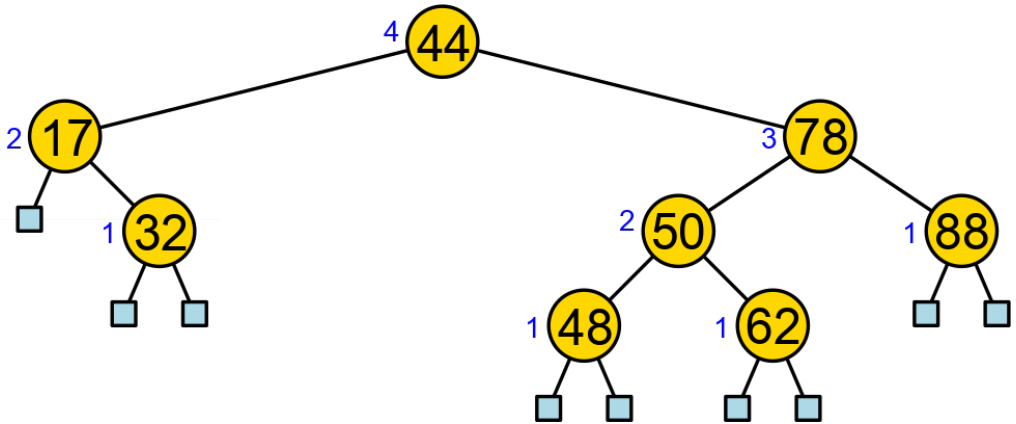
	Υλοποίηση				
Adaptable Priority Queue	Μη-ταξινομημένη λίστα (PositionalList)	Ταξινομημένη λίστα (PositionalList)	**Σωρός (Heap)**	**Leaf-oriented Binary Search Tree**	
				worst case	**Average case**
`insert()`	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$
`removeMin()`	$O(n)$	$O(1)$	$O(\log n)$	$O(n)$	$O(1)$
`min()`	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`remove()`	$O(1)$	$O(1)$	$O(\log n)$	$O(1)$	$O(1)$
`replaceKey()` `[decreaseKey()]`	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$
`replaceValue()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`size()` `isEmpty(), isFull`	$O(1)$				

AVL Δένδρα
[AVL Trees]

AVL Δένδρα [AVL Trees]

AVL δένδρο

- 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"



AVL Δένδρα [AVL Trees]

- Ύψος AVL δένδρου

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

Απόδειξη: (*με επαγωγή*)

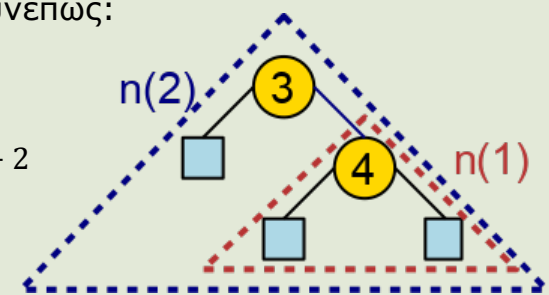
Θέτουμε ως $\mathrm{n}(h)$: τον ελάχιστο αριθμό από εσωτερικούς κόμβους ενός AVL δένδρο ύψους h.

- Εύκολα βλέπουμε ότι $n(1) = 1$ και $n(2) = 2$
- Για $h > 2$, ένα AVL δένδρο ύψους h περιέχει τον κόμβο ρίζα, ένα AVL υποδένδρο ύψους $h-1$ και ένα υποδένδρου ύψους $h-2$.

  Ισχύει λοιπόν ότι $\mathrm{n}(h) = 1 + \mathrm{n}(h-1) + \mathrm{n}(h-2)$

  Γνωρίζοντας ότι $\mathrm{n}(h-1) > \mathrm{n}(h-2)$, έχουμε ότι: $\mathrm{n}(h) > 2 \cdot \mathrm{n}(h-2)$. Συνεπώς:

$$\left.\begin{aligned} &\mathrm{n}(h) > 2 \cdot \mathrm{n}(h-2) \\ &\mathrm{n}(h) > 4 \cdot \mathrm{n}(h-4) \\ &\mathrm{n}(h) > 8 \cdot \mathrm{n}(h-6) \\ &\dots \\ &\mathrm{n}(h) > 2^i \cdot \mathrm{n}(h-2i) \end{aligned}\right\} \Rightarrow \mathrm{n}(h) > 2^{h/2-1} \Rightarrow h < 2 \cdot \log \mathrm{n}(h) + 2$$

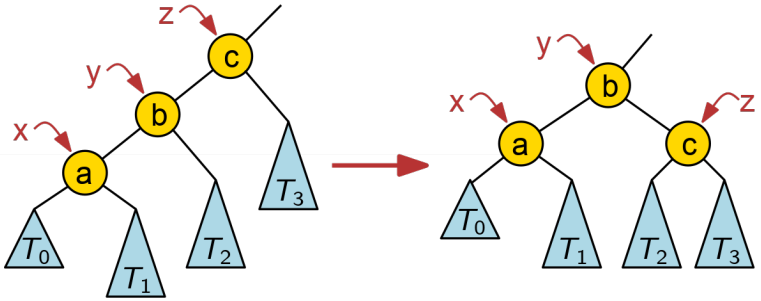


- Άρα το ύψος ενός AVL δένδρο είναι $O(\log n)$

AVL Δένδρα [AVL Trees]

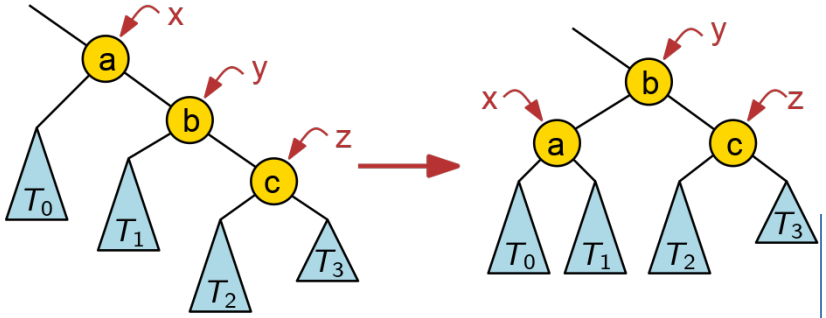
- Αναδιάταξη μέσω περιστρεφών
  - Περιστροφή [Rotation]: Τοπικός μετασχηματισμός γύρω από έναν κόμβό, χωρίς να αλλάζει η inorder διαπέραση των στοιχείων.

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



Το `y` δείχνει στο «μεσαίο» σε μέγεθος στοιχείο εκ των `a, b, c`, όπου `a≤b≤c`

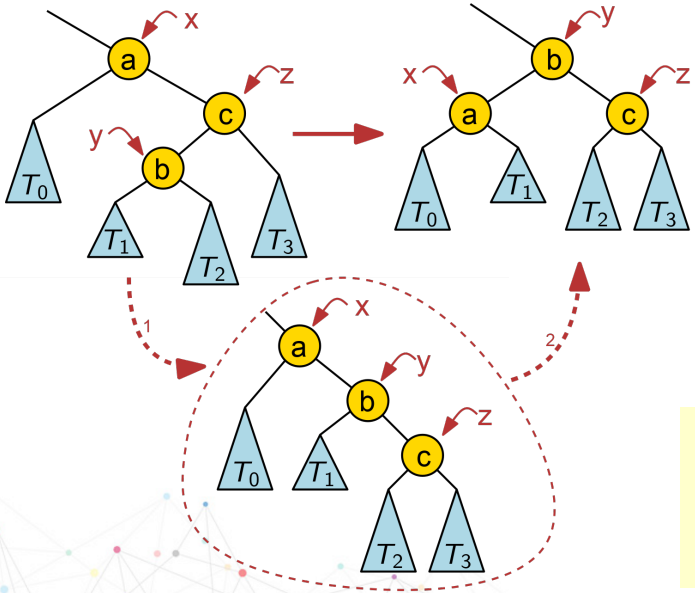
  - Αριστερή περιστροφή γύρω από τον κόμβο `y`



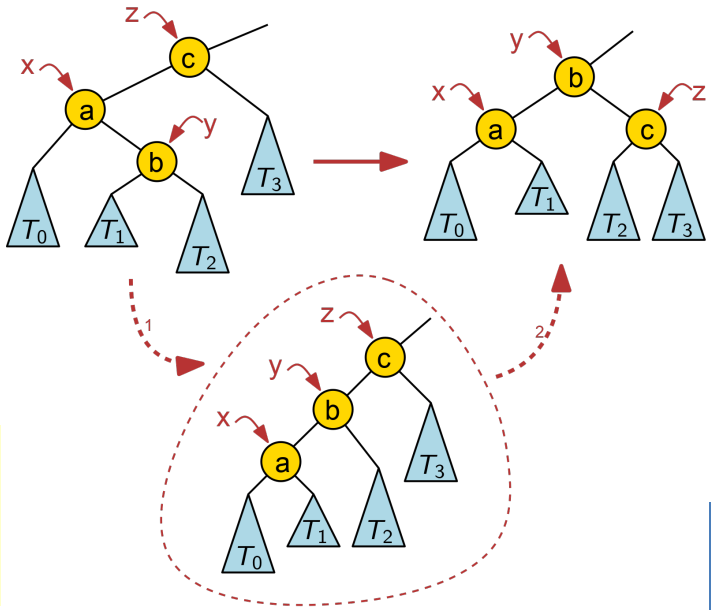
AVL Δένδρα [AVL Trees]

- Αναδιάταξη μέσω περιστρεφών (*συνέχεια*)
  - Διπλές περιστροφές

- Διπλή αριστερή περιστροφή γύρω από τον κόμβο `y` [πρώτα «δεξιά», μετά «αριστερά»]



- Διπλή δεξιά περιστροφή γύρω από τον κόμβο `y` [πρώτα «αριστερά», μετά «δεξιά»]



Το `y` δείχνει στο «μεσαίο» σε μέγεθος στοιχείο εκ των `a, b, c`, όπου `a≤b≤c`

AVL Δένδρα [AVL Trees]

- Εισαγωγή στοιχείου
  - Η εισαγωγή είναι ίδια όπως και στα δυαδικά δένδρα αναζήτησης
  - Πάντα ολοκληρώνεται με μία «επέκταση» ενός εξωτερικού κόμβου

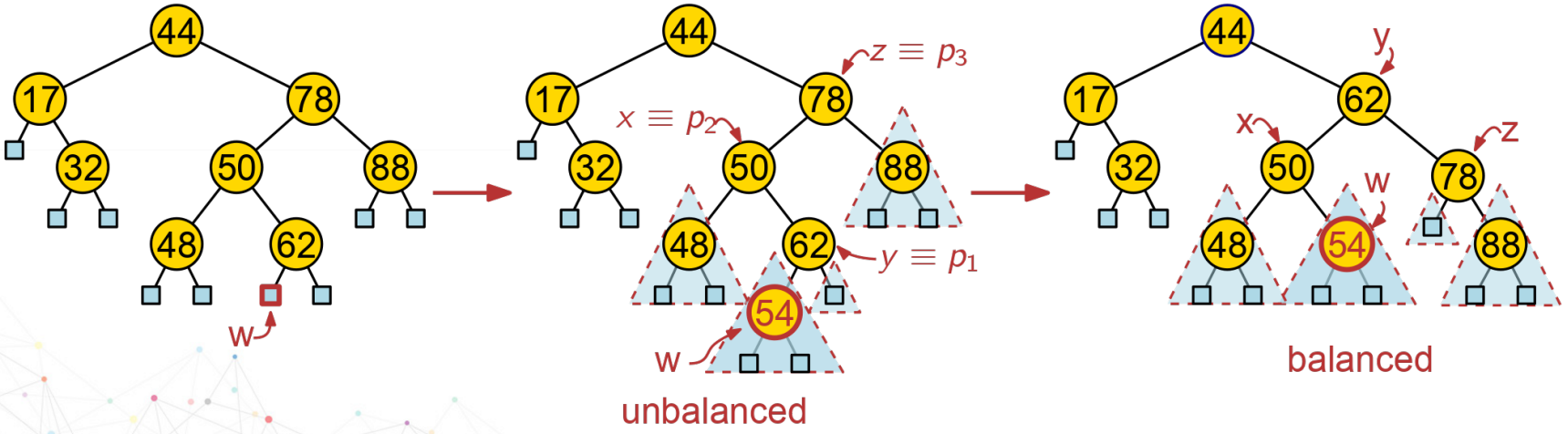
p_3: Ο πιο κοντινός μη-ισοζυγισμένος πρόγονος του w

p_2: Το παιδί του p_3 το οποίο είναι πρόγονος του w.

p_1: Το παιδί του p_2 το οποίο είναι πρόγονος του w

Παράδειγμα:

- Εισαγωγή στοιχείου **54**



AVL Δένδρα [AVL Trees]

- Διαγραφή στοιχείου
  - Όπως στα δυαδικά δένδρα αναζήτησης,
    - ο κόμβος που θα διαγραφεί μετατρέπεται σε άδειο εξωτερικό κόμβο.
    - Στον πατέρας του, `w`, μπορεί να προκληθεί ανισορροπία.

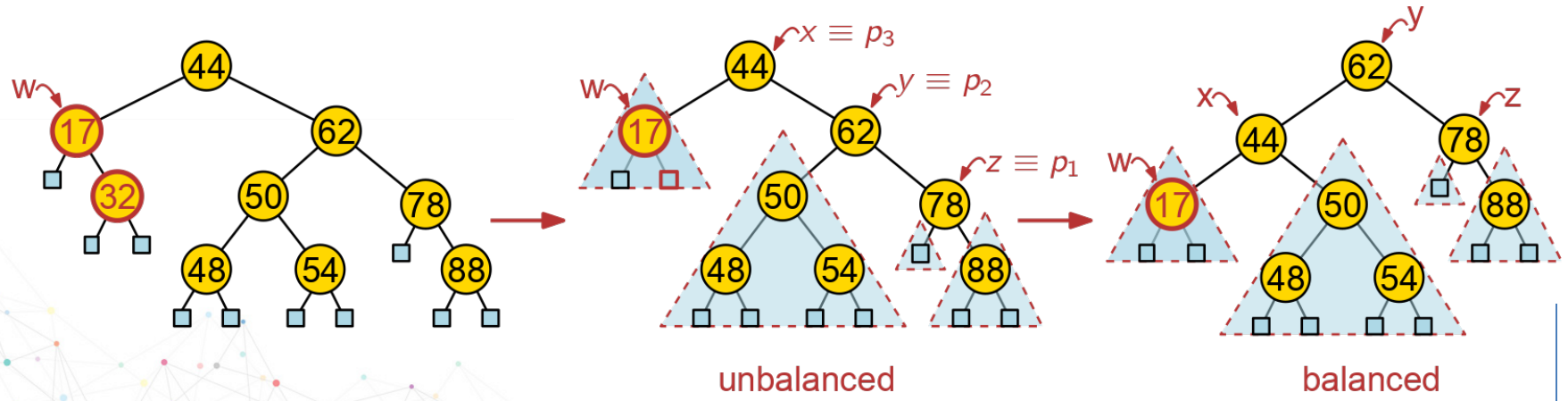
p_3: Ο πιο κοντινός μη-ισοζυγισμένος πρόγονος του `w`

p_2: Το παιδί του p_3 με το μεγαλύτερο ύψος υποδένδρου

p_1: Το παιδί του p_2 με το μεγαλύτερο ύψος υποδένδρου

Παράδειγμα:

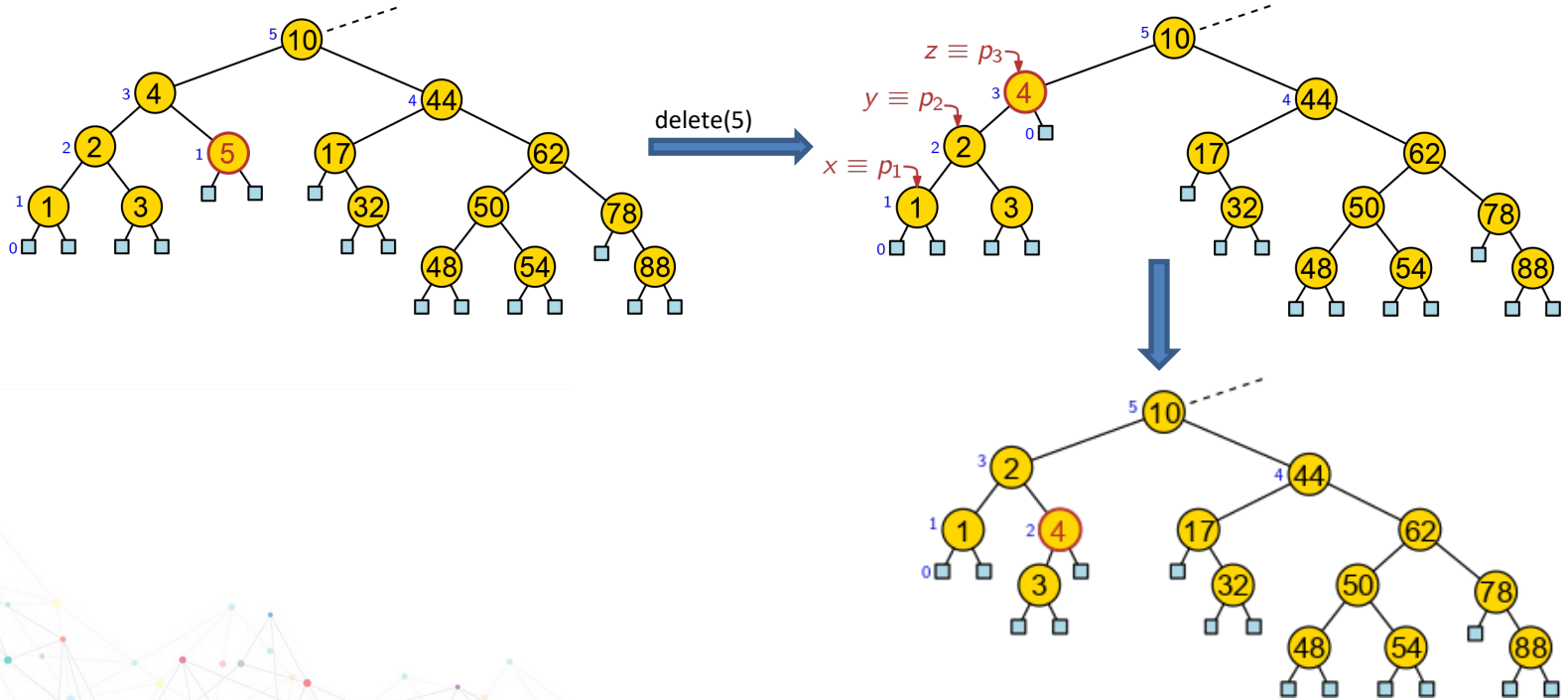
- Αφαίρεση στοιχείου **32**



AVL Δένδρα [AVL Trees]

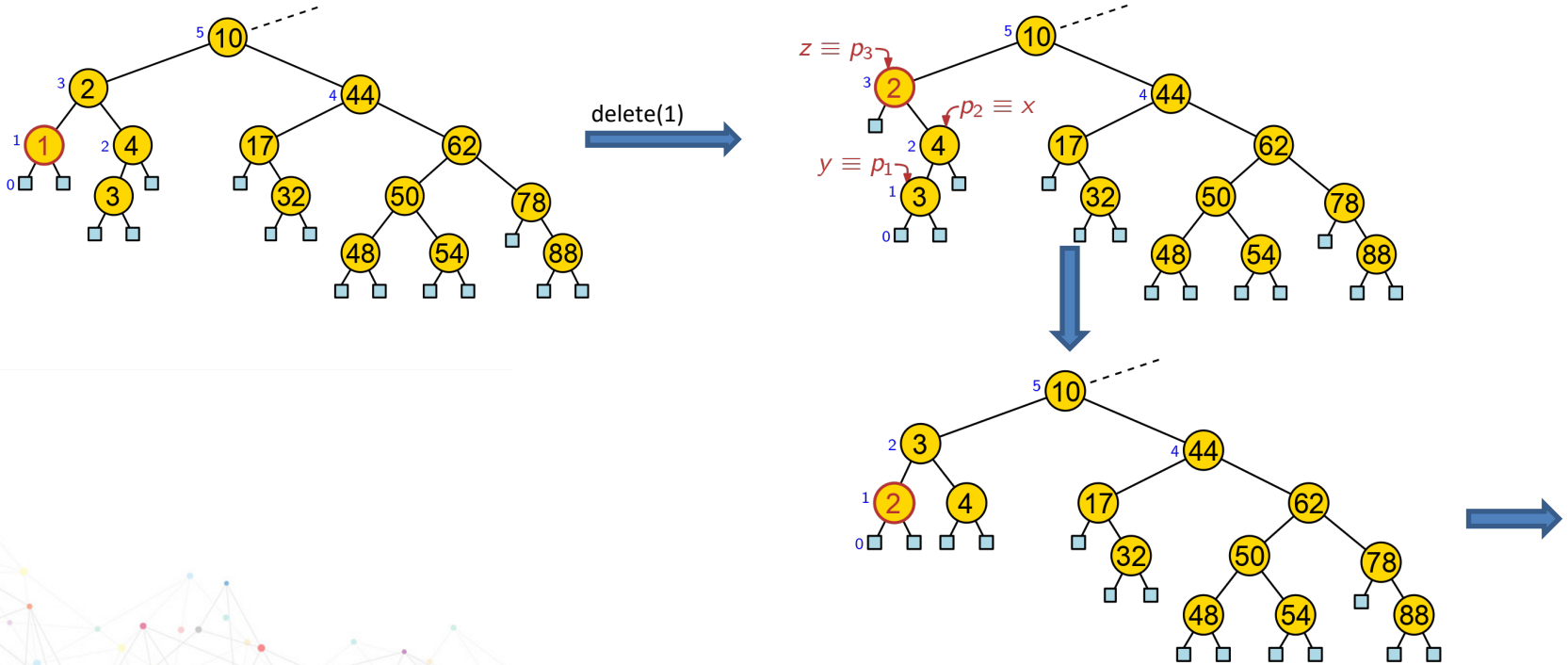
- Διαγραφή στοιχείου (συνέχεια)

Παράδειγμα: Αφαίρεση στοιχείων **5, 1**



AVL Δένδρα [AVL Trees]

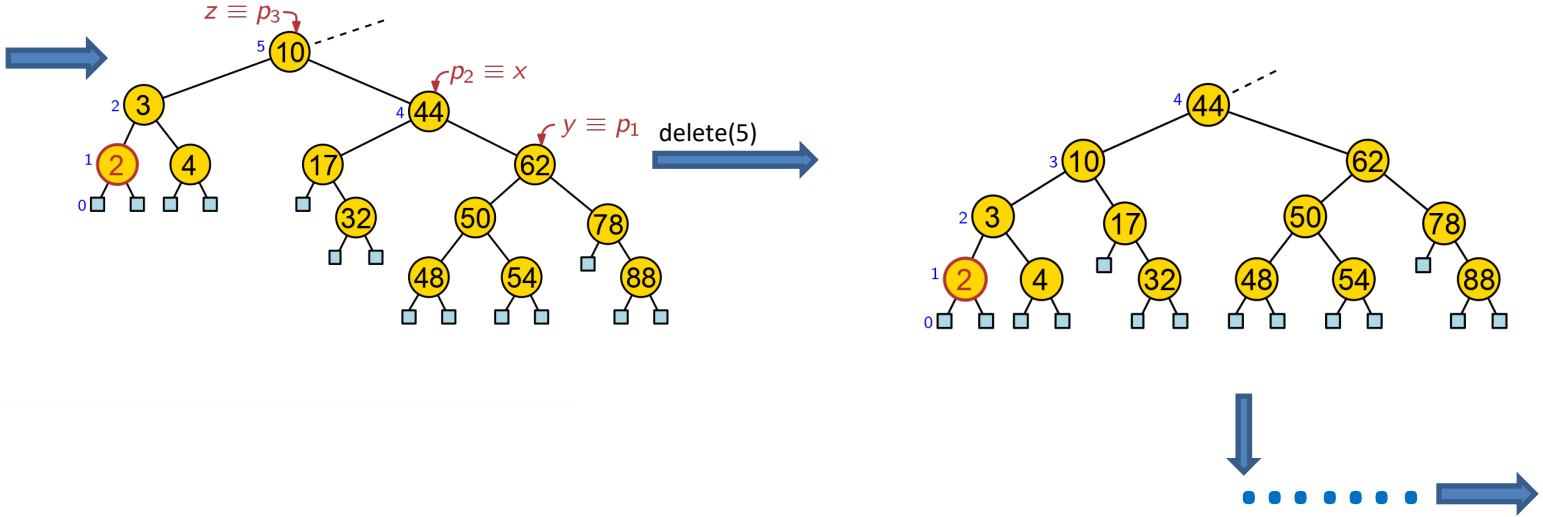
Παράδειγμα (συνέχεια): Αφαίρεση στοιχείων **5, 1**



AVL Δένδρα [AVL Trees]

- Διαγραφή στοιχείου (συνέχεια)

Παράδειγμα (συνέχεια): Αφαίρεση στοιχείων **5, 1**



AVL Δένδρα [AVL Trees]

- 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* $\boldsymbol{O}(\log \boldsymbol{n})$
    - Η εισαγωγή στοιχείου χρειάζεται $O(\log n)$ χρόνο
      - *Η αρχική αναζήτηση χρειάζεται* $O(\log n)$
      - *Απαιτείται μόνο μία αναδιάταξη*
    - Η διαγραφή στοιχείου χρειάζεται $O(\log n)$ χρόνο
      - *Η αρχική αναζήτηση χρειάζεται* $O(\log n)$
      - *Το πολύ* $O(\log n)$ *αναδιατάξεις*

AVL Δένδρα [AVL Trees]

- Υλοποίηση SortedMap με AVL Tree

ΑΤΔ SortedMap	AVL Tree	Leaf-oriented AVL Tree
`put(k,v)`	$O(\log n)$	$O(\log n)$
`get(k)`	$O(\log n)$	$O(\log n)$
`remove(k)`	$O(\log n)$	$O(\log n)$
`size()`	$O(1)$	$O(1)$
`isEmpty()`	$O(1)$	$O(1)$
`firstEntry(),lastEntry()`	$O(\log n)$	$\mathbf{O(1)}$
`ceilingEntry(k), floorEntry(k), lowerEntry(k), higherEntry(k)`	$O(\log n)$	$O(\log n)$
`subMap(k1,k2)` [s το πλήθος των στοιχείων μεταξύ k1 και k2]	$O(s \log n)$	$\mathbf{O(s + \log n)}$
`entrySet(), keySet(), values()`	$O(n)$	$O(n)$

AVL Δένδρα [AVL Trees]

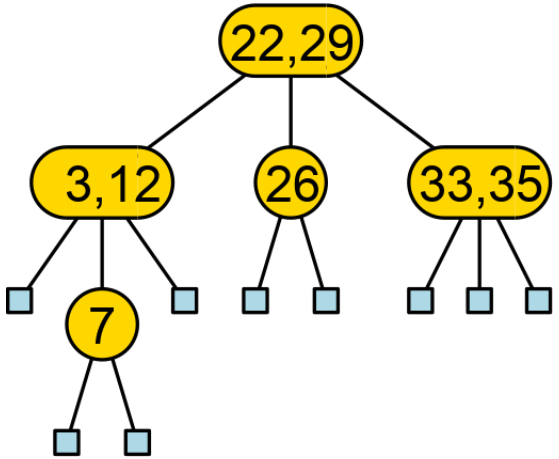
- Υλοποίηση AdaptablePriorityQueue με AVL Tree

	Υλοποίηση			
Adaptable Priority Queue	Σωρός (Heap)	Binary Search Tree (Average case)	AVL Tree	Leaf-oriented AVL Tree
`insert()`	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`removeMin()`	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`min()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`remove()`	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`replaceKey()` `[decreaseKey()]`	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
`replaceValue()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`size()`	$O(1)$			
`isEmpty(), isFull`				

B-Δένδρα
[B-Trees]

B-Δένδρα [B-Trees]

- k-αδικά δένδρα αναζήτησης (ή πολλαπλής αναζήτησης)
  - Δένδρα αναζήτησης με το πολύ $k \geq 1$ παιδιά σε κάθε κόμβο
  - Σε κάθε κόμβο, τα κλειδιά είναι ταξινομημένα
  - Η αναζήτηση είναι γενίκευση της αναζήτησης σε δύαδικό δένδρο αναζήτησης
  - Η διαπέραση είναι γενίκευση της inorder διαπέρασης για δυαδικό δένδρο αναζήτησης ($\Longrightarrow$ ταξινόμηση των στοιχείων)



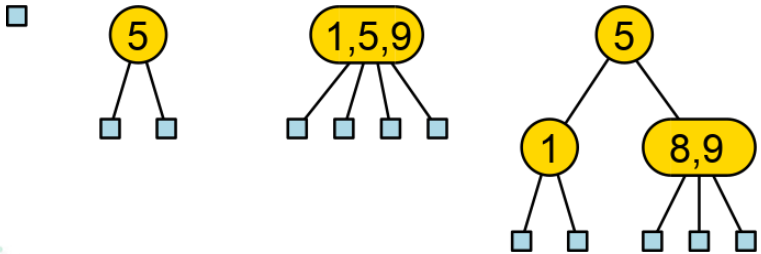
Β-Δένδρα [B-Trees]

- Ένα Β-Δένδρο τάξεως-m ικανοποιεί:
  1. Είναι ένα δένδρο πολλαπλής αναζήτησης
  2. Η ρίζα είναι εξωτερικός κόμβος ή έχει από 2 έως m παιδιά
  3. Κάθε εσωτερικός κόμβος (η ρίζα μπορεί να αποτελεί εξαίρεση) έχει από $\left\lceil \frac{m}{2} \right\rceil$ έως m παιδιά
  4. Όλοι οι εξωτερικοί κόμβοι έχουν το ίδιο βάθος

- Οι συνθήκες (2) και (3) επιτρέπουν τη χρήση μνήμης σταθερού μεγέθους σε κάθε κόμβο [m αναφορές, $m-1$ κλειδιά, 1 μετρητής πλήθους κλειδιών]
- Η συνθήκη (4) εγγυάται ισοσκελισμένο δένδρο

Παράδειγμα:

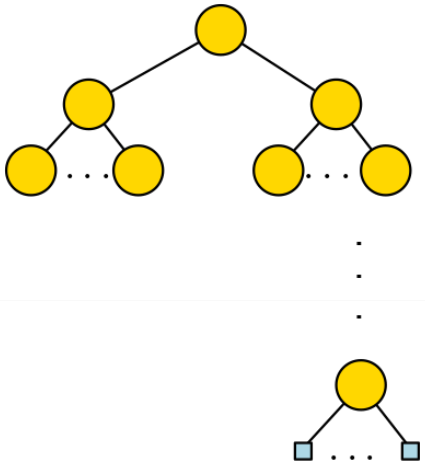
- Β-δένδρα τάξεως **4**



Β Δένδρα [B Trees]

Θεώρημα: Έστω Τ ένα Β-δένδρο τάξεως m, με ύψος h και $n \geq 1$ εγγραφές. Τότε:

$$n \geq 2\lceil m/2 \rceil^{h-1} - 1$$



Βάθος	Ελάχιστος αριθμός κόμβων
0	1
1	2
2	$2\lceil m/2 \rceil$
$h-1$	$2\lceil m/2 \rceil^{h-2}$
h	0

- Εξαιρώντας τη ρίζα, ο αριθμός των κόμβων είναι:

$$2 + 2\lceil m/2 \rceil + \cdots + 2\lceil m/2 \rceil^{h-2} = 2\frac{\lceil m/2 \rceil^{h-1} - 1}{\lceil m/2 \rceil - 1}$$

- Κάθε κόμβος περιέχει $\lceil m/2 \rceil - 1$ εγγραφές. Άρα, συμπεριλαμβάνοντας το στοιχείο στη ρίζα, συνολικά έχουμε:

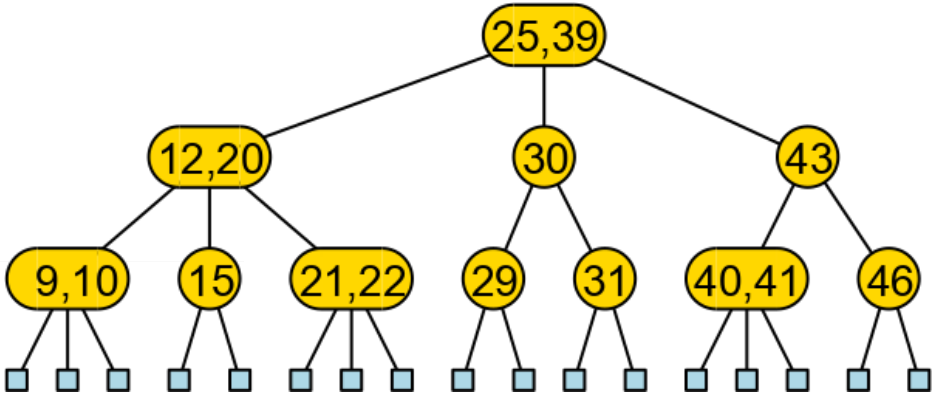
$$n \geq 2\left(\lceil m/2 \rceil^{h-1} - 1\right) + 1 = 2\lceil m/2 \rceil^{h-1} - 1$$

$$\Rightarrow h \leq \log_{\lceil \frac{m}{2} \rceil}\left(\frac{n+1}{2}\right) + 1$$

Ερώτηση: Ποιος ο μέγιστος αριθμός εγγραφών σε ένα Β-δένδρο τάξεως-m και ύψους h ;

B-Δένδρα [B-Trees]

- B-Δένδρο τάξεως-3 [γνωστό και ως "2-3 δένδρο"]
  - Έχουν 2 ή 3 παιδιά ανά κόμβο
  - Είναι ιδανικά για υλοποίηση διατεταγμένου χάρτη (SortedMap) σε εσωτερική μνήμη (internal memory)



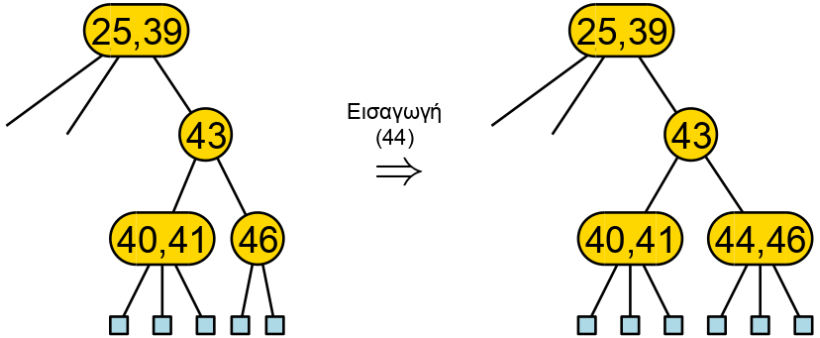
B-Δένδρα [B-Trees]

- B-Δένδρα τάξεως-m, για μεγάλο m ($m \geq 128$)
  - Μικρό βάθος
  - Εφαρμογή σε βάσεις δεδομένων (αποθήκευση στοιχείων στο δίσκο/εξωτερική μνήμη)
  - Κόμβος ↔ block δίσκου
  - Γρήγορη αναζήτηση (προσπέλαση σε μικρό αριθμό blocks)
    - Η πιο χρονοβόρα λειτουργία είναι η προσπέλαση ενός block του δίσκου (απαιτεί περιστροφή του δίσκου και κίνηση των κεφαλών ανάγνωσης/εγγραφής)
  - Στους εσωτερικούς κόμβους αποθηκεύονται μόνο τα κλειδιά
  - Στους εξωτερικούς κόμβους αποθηκεύονται οι εγγραφές (φυλλο-προσανατολισμένο δένδρο)

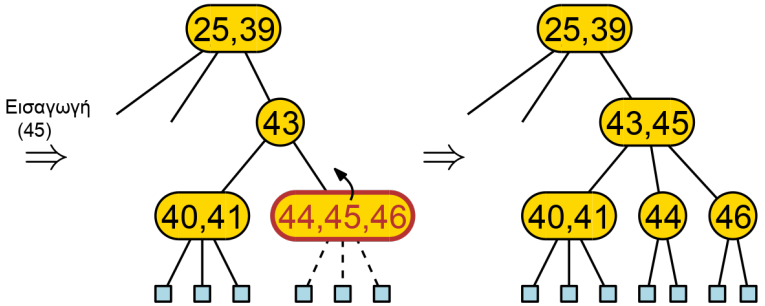
Ερώτηση: Πόσοι εξωτερικοί κόμβοι υπάρχουν;

B-Δένδρα [B-Trees]

- Εισαγωγή εγγραφής
  - Παράδειγμα B-δένδρου βαθμού-3 (2-3 δένδρο)
  - Οι εισαγωγές γίνονται στους κόμβους μέγιστου βάθους

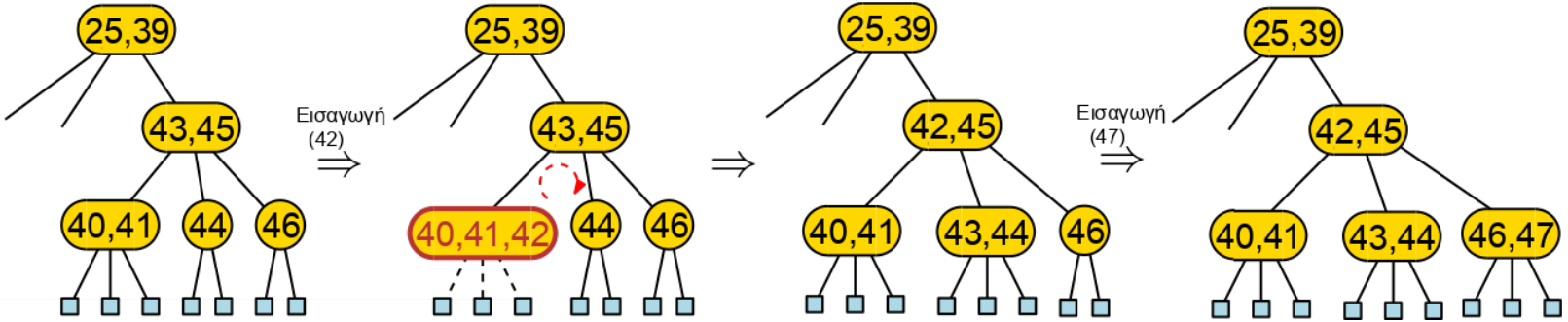


  - Εάν ο κόμβος είναι πλήρης τότε δημιουργούμε αρχικά έναν προβληματικό κόμβο και στη συνέχεια τον αναπροσαρμόζουμε



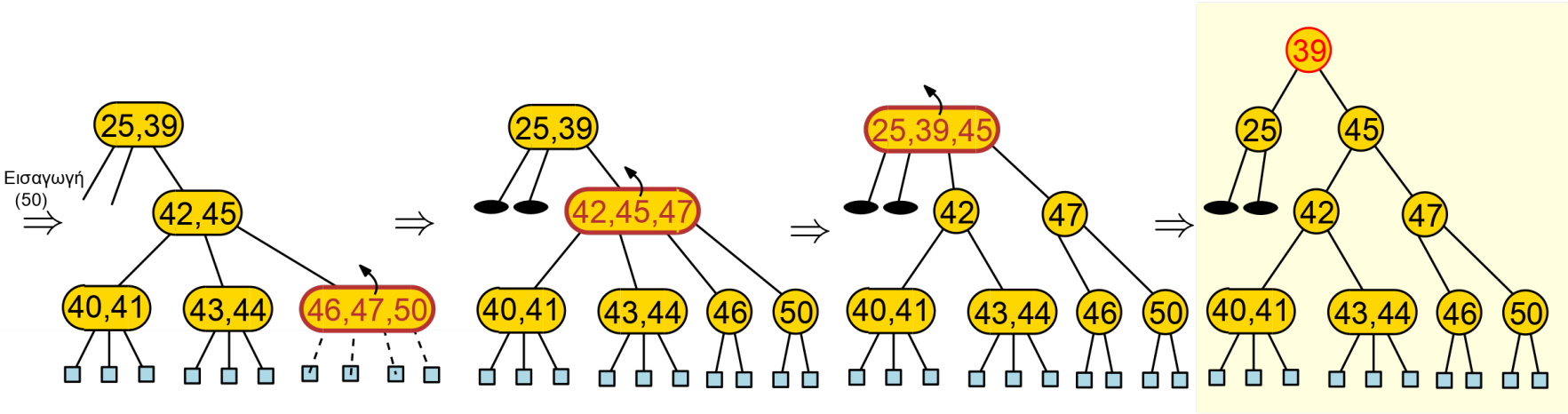
B-Δένδρα [B-Trees]

- Εισαγωγή εγγραφής (*συνέχεια*)
  - Εφαρμογή «περιστροφών» σε υπερπλήρεις κόμβους



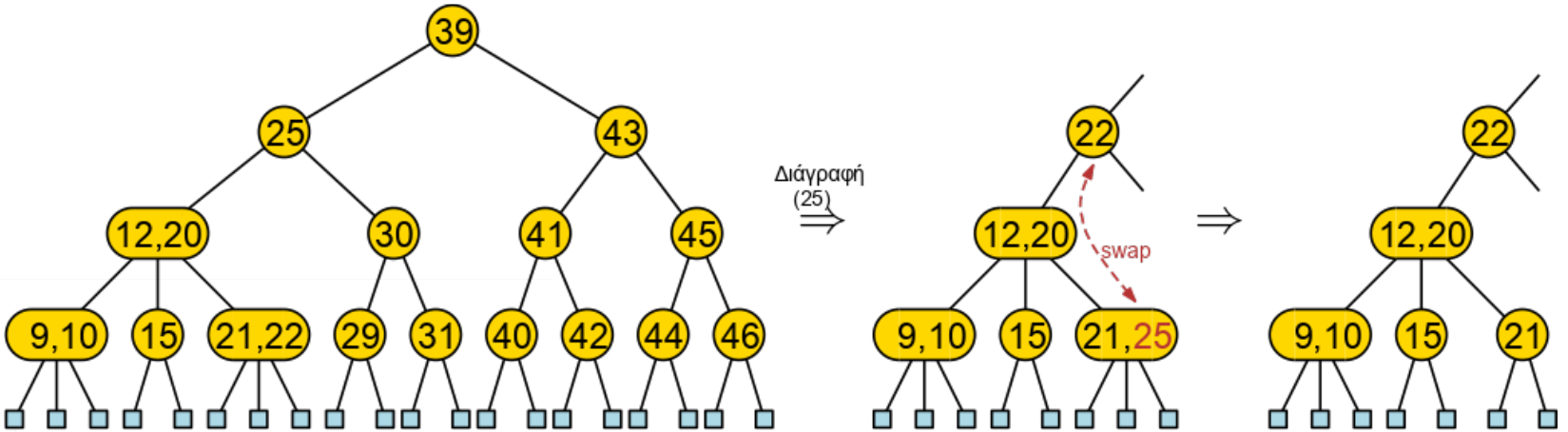
B-Δένδρα [B-Trees]

- Εισαγωγή εγγραφής (*συνέχεια*)
  - Το ύψος του δένδρου μεγαλώνει εάν χωριστεί στα δύο η ρίζα



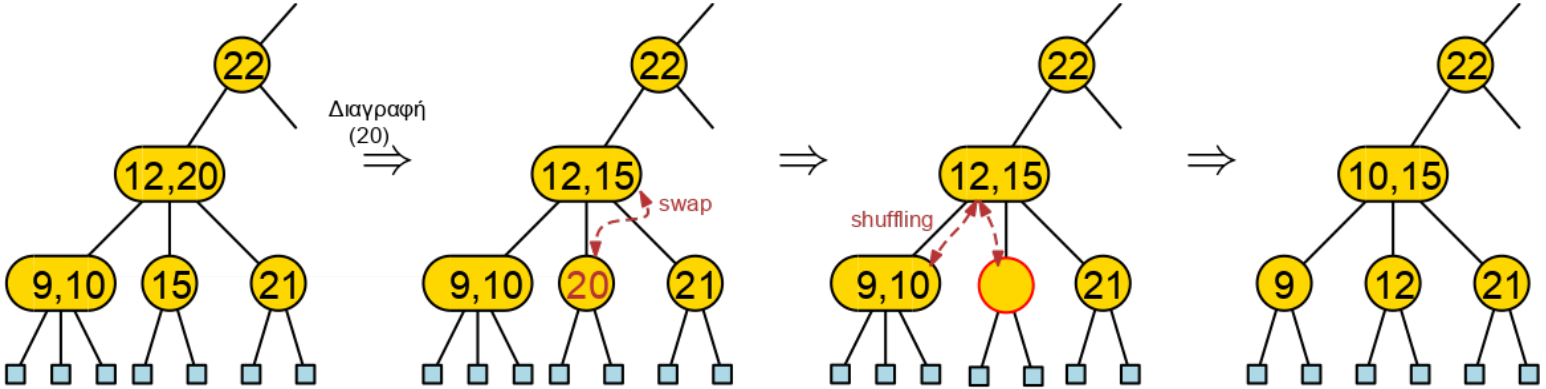
B-Δένδρα [B-Trees]

- Διαγραφή εγγραφής
  - Η διαγραφή αρχίζει από το τελευταίο επίπεδο (ανταλλαγή στοιχείου με την προηγούμενο "inorder" εγγραφή)



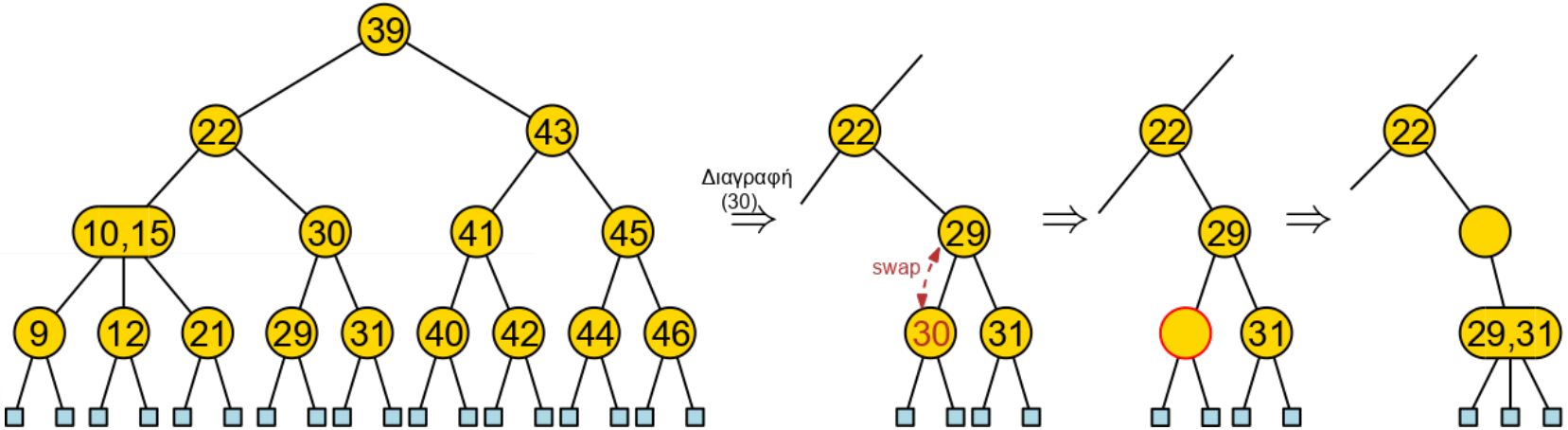
B-Δένδρα [B-Trees]

- Διαγραφή εγγραφής (*συνέχεια*)
  - Μεταφορά εγγραφής από το αριστερό (ή δεξί) αδελφό (shuffling)



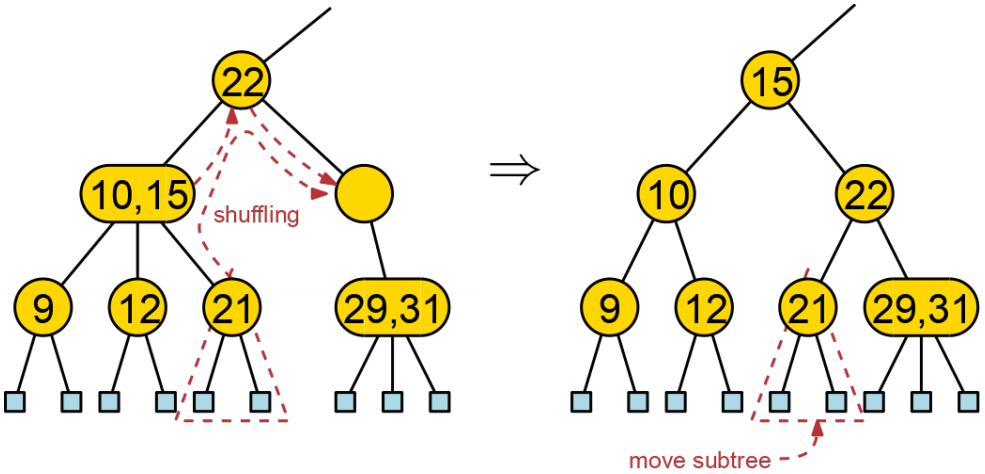
B-Δένδρα [B-Trees]

- Διαγραφή εγγραφής (*συνέχεια*)
  - Αν δεν είναι δυνατή η περιστροφή εγγραφών (shuffling) λόγω έλλειψης εγγραφών, συγχωνεύουμε τον αδελφό με τον πατέρα



B-Δένδρα [B-Trees]

- Διαγραφή εγγραφής (*συνέχεια*)
  - Στη συνέχεια μεταφέρουμε στοιχεία από τον αδελφό του κενού κόμβου (shuffling)
  - Εάν η ρίζα αδειάσει, τότε μικραίνει το ύψος του δένδρου



Συνεχίστε το παράδειγμα διαγράφοντας την εγγραφή **12**.

B-Δένδρα [B-Trees]

- Απόδοση
  - Αναζήτηση:

$$\underset{\substack{\uparrow \\ \text{δυαδική αναζήτηση} \\ \text{σε κόμβο}}}{O(\log m)} \cdot \underset{\substack{\uparrow \\ \text{βάθος δένδρου}}}{O(\log n)} \xrightarrow{m \text{ σταθερό}=O(1)} O(\log n)$$

  - Εισαγωγή:

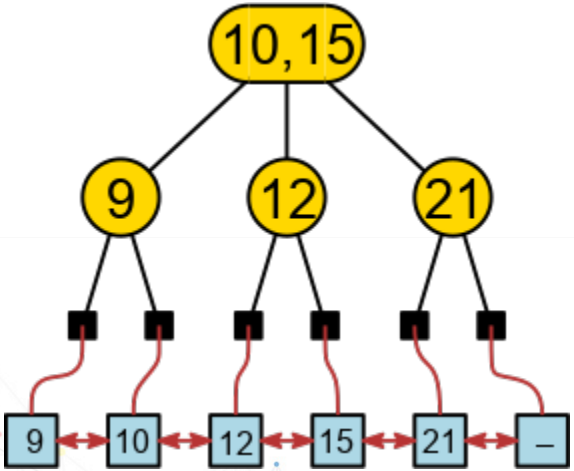
$$\underset{\substack{\uparrow \\ \text{δυαδική αναζήτηση} \\ \text{σε κόμβο + εισαγωγή} \\ \text{(σπρώξιμο εγγραφών δεξιά)}}}{O(m)} \cdot \underset{\substack{\uparrow \\ \text{βάθος δένδρου}}}{O(\log n)} \xrightarrow{m \text{ σταθερό}=O(1)} O(\log n)$$

  - Διαγραφή: (*όμοια με την εισαγωγή, αλλά πιο πολύπλοκη*)

$$\underset{\substack{\uparrow \\ \text{δυαδική αναζήτηση} \\ \text{σε κόμβο + διαγραφή} \\ \text{(αναδιάταξη)}}}{O(m)} \cdot \underset{\substack{\uparrow \\ \text{βάθος δένδρου}}}{O(\log n)} \xrightarrow{m \text{ σταθερό}=O(1)} O(\log n)$$

B-Δένδρα [B-Trees]

- Ερώτηση: Πώς υποστηρίζει τις μεθόδους ενός SortedMap?
  - Μέσω φυλλο-προσανατολισμένων B-δένδρων.
    - Οι εγγραφές αποθηκεύονται στους εξωτερικούς κόμβους
    - Τα κλειδιά αποθηκεύονται στους εσωτερικούς κόμβους
    - Οι εξωτερικοί κόμβοι είναι συνδεδεμένοι σε λίστα



Μέθοδος	B Δένδρο
`put(k,v)`	$O(\log n)$
`get(k)`	$O(\log n)$
`remove(k)`	$O(\log n)$
`ceilingEntry(k)`, `floorEntry(k)`, `lowerEntry(k)`, `higherEntry(k)`	$O(\log n)$
`subMap(k1,k2)`	$O(\log n + S)$

Map

SortedMap

(α,β)-Δένδρα

[(a,b)-Trees]

(α,β)-Δένδρα [(a,b)-Trees]

(α,β) δένδρο, με α ≥ 2 και β ≥ 2α

1. Τα φύλλα έχουν ίδιο βάθος
2. Η ρίζα έχει τουλάχιστον 2 και το πολύ β παιδιά
3. Το πλήθος των παιδιών των εσωτερικών κόμβων είναι μεταξύ α και β
4. Είναι δένδρο πολλαπλής αναζήτησης