Δένδρα

Γιώργος Στάμου

Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Εισαγωγή

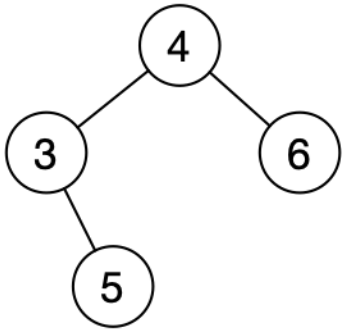
- Τα δένδρα είναι δομές δεδομένων που αναπαριστούν κάποιου είδους ιεραρχία
- Είναι ιδιαίτερα χρήσιμα για μια μεγάλη κατηγορία υπολογισμών και προβλημάτων
- Μπορούμε να χρησιμοποιήσουμε τα δένδρα ως αποτελεσματικές δομές αναπαράστασης και διαχείρισης δεδομένων
  - Μπορούν να υποστηρίζουν αποδοτικά: την εισαγωγή, διαγραφή ή τροποποίηση στοιχείων και την αναζήτησή τους

Ορισμός δένδρου

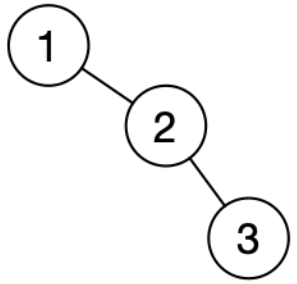
- Ένα δένδρο Τ είναι μία πεπερασμένη, πιθανά κενή λίστα
- $T = [r ; T_1, T_2, \ldots, T_n]$
  - με τις εξής ιδιότητες:
  - Το ορισμένο στοιχείο της λίστας (αν υπάρχει), δηλαδή το **r**, καλείται ρίζα του δένδρου
  - Καθένα από τα υπόλοιπα στοιχεία $T_1, T_2, \ldots, T_n$ της λίστας ($n \geq 0$) είναι δένδρα (τα ονομάζουμε υποδένδρα του Τ)
- (Αναφέρουμε τα ορισμένα στοιχεία της λίστας (μαζί με αυτά όλων των υποδένδρων) ως κόμβους του δένδρου)

Παραδείγματα δένδρων

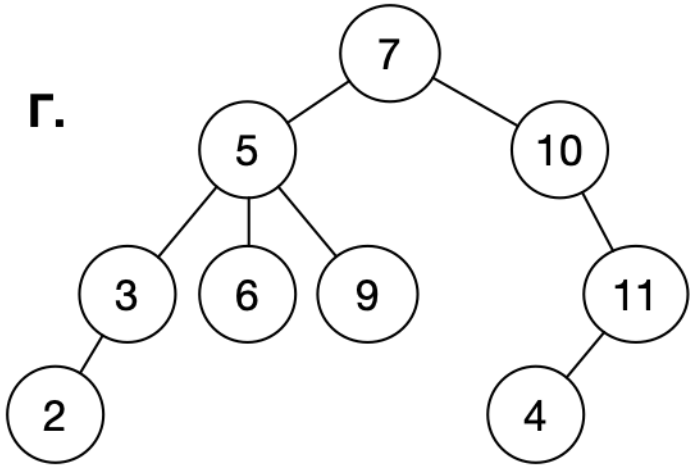
Α.



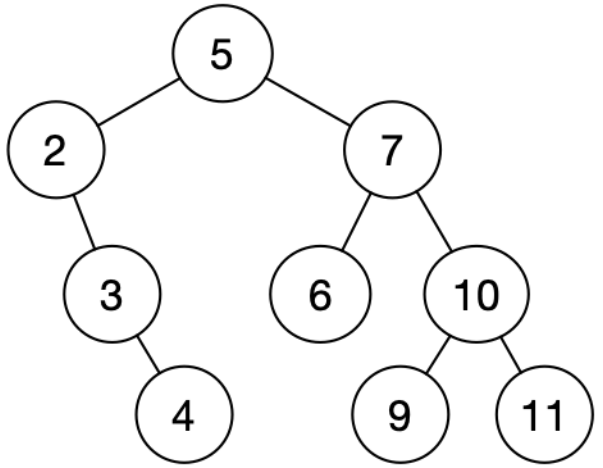
Β.



Γ.

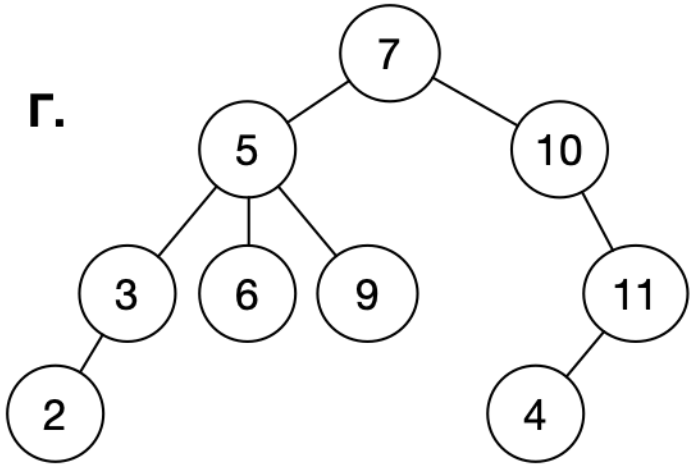


Δ.



Παραδείγματα δένδρων

Γ.



T = [7 ; [5 ; [3 ; [2]], [6], [9]], [10 ; [11 ; [4]]]]

T = [7 ; [5 ; [3 ; [2]], [6], [9]], [10 ; [], [11 ; [4]]]]

Παράδειγμα δενδρικής δομής

```
D: begin
      E: begin
            F: statement
      end;
      G: begin
            H: begin
                  I: statement
            end;
            J: begin
                  K: statement;
                  L: statement
            end;
            M: statement
      end
end
```

Ορολογία

– Έστω ένα δένδρο $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$.

– Ο βαθμός ενός κόμβου είναι ο αριθμός των υποδένδρων που συνδέονται με αυτόν τον κόμβο. Για παράδειγμα, ο βαθμός της ρίζας r του δένδρου T είναι n.

– Ένας κόμβος με βαθμό μηδέν δεν έχει κανένα υποδένδρο. Ένας τέτοιος κόμβος ονομάζεται φύλλο.

– Κάθε μία από τις ρίζες r_i των υποδένδρων T_i του δένδρου T ονομάζεται παιδί του r. (Ο όρος εγγόνι ορίζεται κατά παρόμοιο τρόπο.)

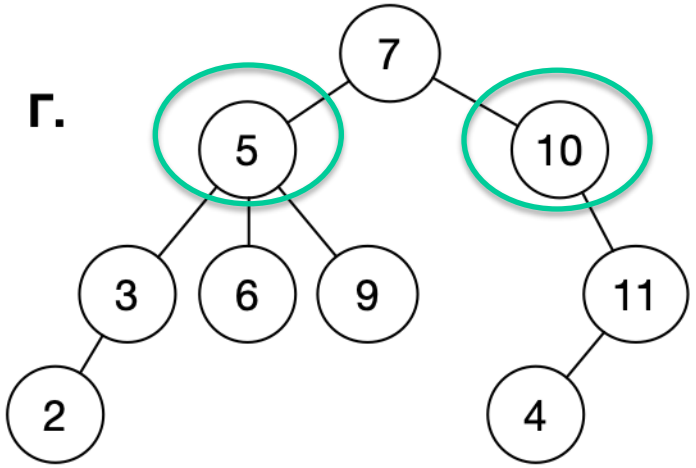
Ορολογία

– Έστω ένα δένδρο $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$.

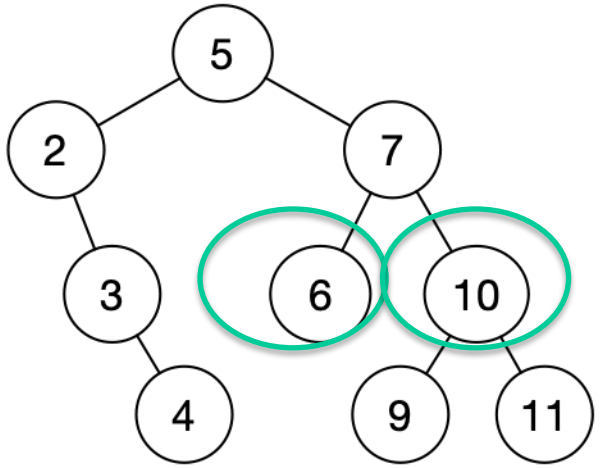
– Η ρίζα r του δένδρου T είναι ο πατέρας όλων των ριζών r_i των υποδένδρων T_i, $1 \leq i \leq n$. (Ο όρος παππούς ορίζεται κατά παρόμοιο τρόπο.)

– Οι ρίζες r_i και r_j δύο διαφορετικών υποδένδρων T_i και Tj του δένδρου T ονομάζονται αδέρφια.

Γ.



Δ.



Μονοπάτι δένδρου

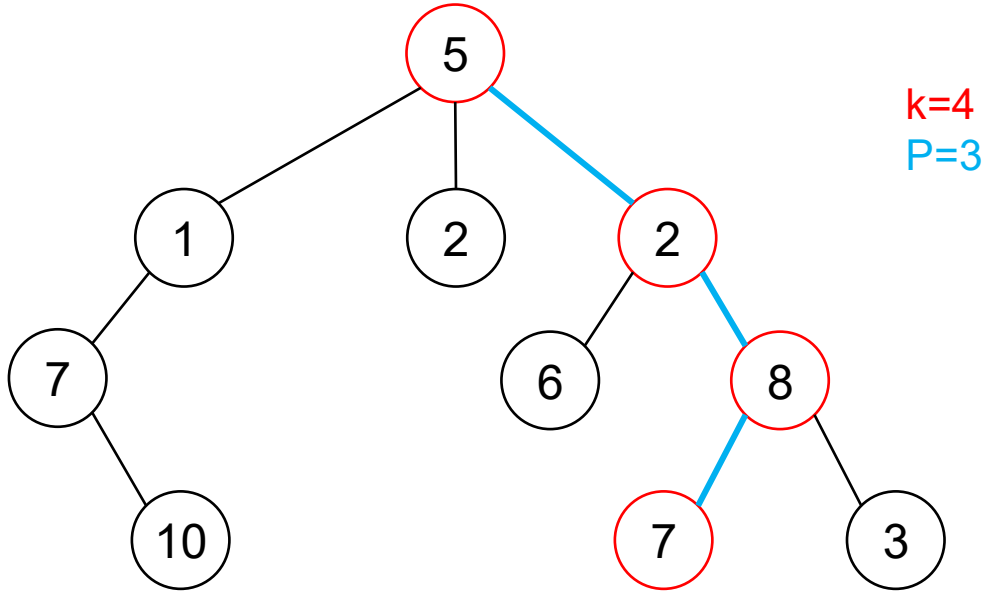
– Έστω ένα δένδρο $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$ το οποίο περιέχει ένα σύνολο κόμβων R.

– Μονοπάτι του δένδρου είναι μία μη-κενή ακολουθία κόμβων:

$$P = r_1, r_2, \ldots, r_k$$

όπου r_i κόμβοι του δένδρου, για $1 \leq i \leq k$ έτσι ώστε ο i-οστός κόμβος στην ακολουθία, r_i, να είναι ο πατέρας του (i+1)-οστού κόμβου στην ακολουθία r_{i+1}.

– Το μήκος του μονοπατιού P είναι k – 1.

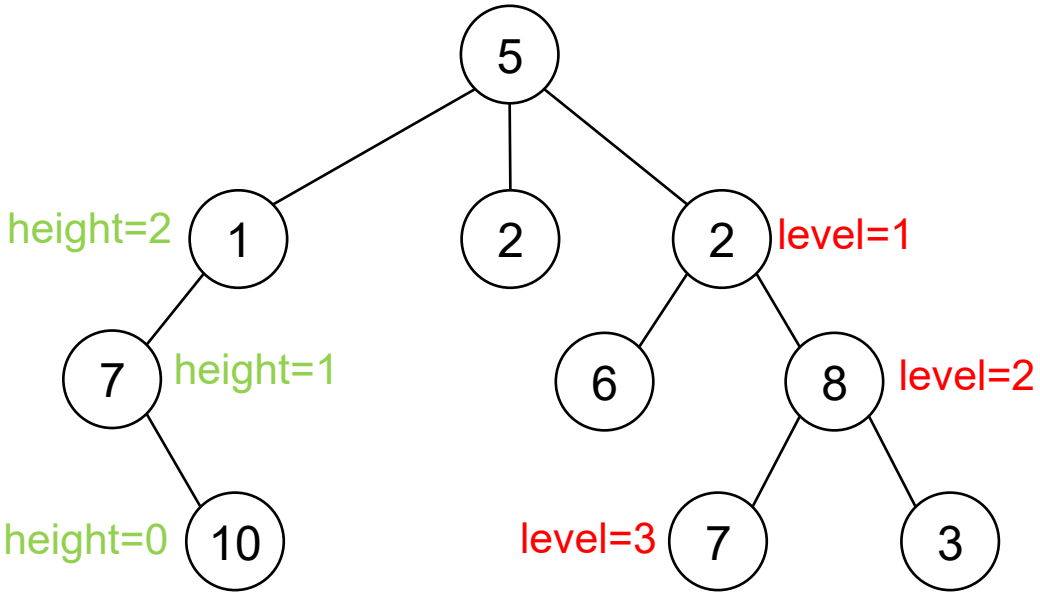


Επιπλέον ορολογία

– Έστω ένα δένδρο $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$ το οποίο περιέχει ένα σύνολο κόμβων R.

– Το επίπεδο (level) ή βάθος (depth) ενός κόμβου r_i του δένδρου T είναι το μήκος του μοναδικού μονοπατιού από την ρίζα r στον κόμβο r_i. Για παράδειγμα, η ρίζα του δένδρου T είναι στο επίπεδο μηδέν και οι ρίζες των υποδένδρων του T είναι όλες στο επίπεδο ένα.

– Το ύψος (height) ενός κόμβου r_i του δένδρου T είναι το μήκος του μεγαλύτερου σε μήκος μονοπατιού από τον κόμβο r_i σε κάποιο φύλλο. Επομένως, όλα τα φύλλα έχουν ύψος ίσο με μηδέν.



Επιπλέον ορολογία

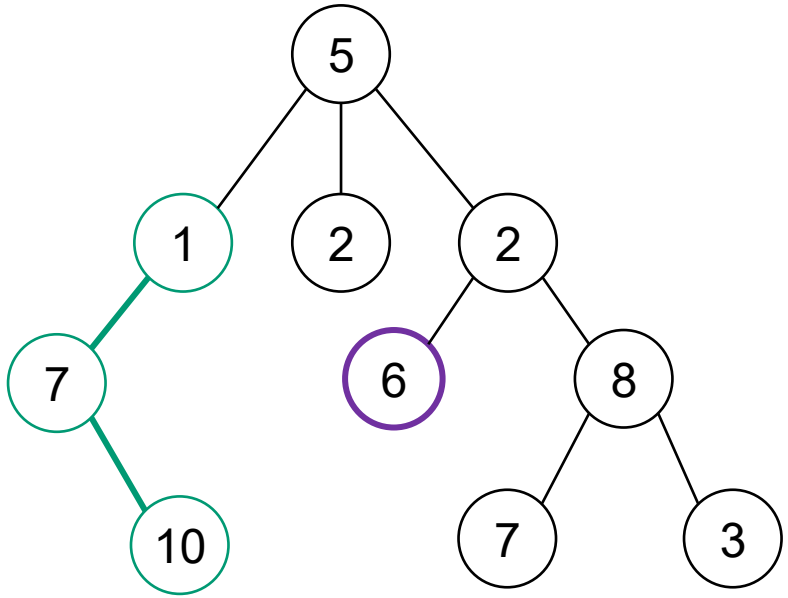
– Έστω ένα δένδρο $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$ το οποίο περιέχει ένα σύνολο κόμβων R.

– Το ύψος ενός δένδρου T είναι το ύψος της ρίζας του r.

– Θεωρούμε (κατά σύμβαση) ότι το κενό δένδρο έχει ύψος -1.

– Έστω δύο κόμβοι r_i και r_j ενός δένδρου T.

- Ο κόμβος r_i καλείται πρόγονος (ancestor) του κόμβου r_j εάν υπάρχει ένα μονοπάτι στο T από τον r_i στον r_j.
- Ο κόμβος r_i καλείται γνήσιος πρόγονος (proper ancestor) εάν υπάρχει ένα μονοπάτι p στο δένδρο T από τον κόμβο r_i στον κόμβο r_j με μήκος μονοπατιού p να είναι διάφορο του μηδενός.



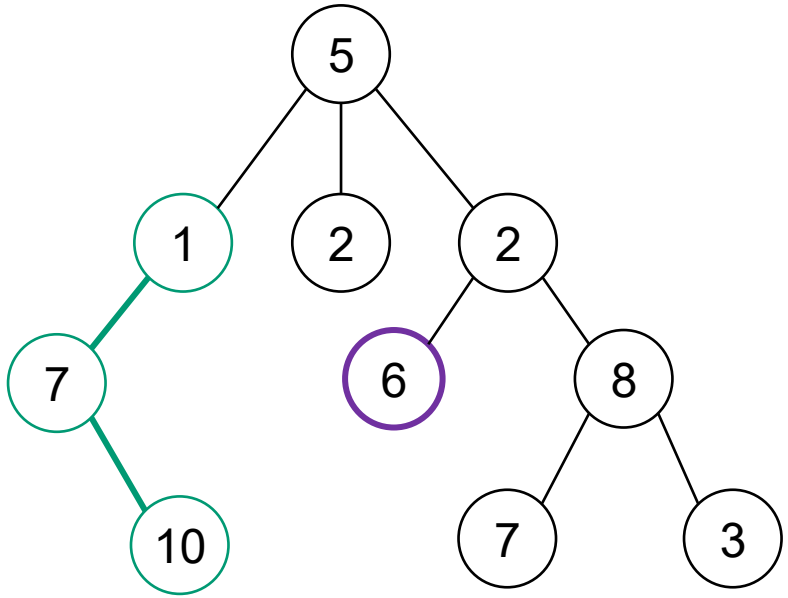
Επιπλέον ορολογία

– Έστω ένα δένδρο $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$ το οποίο περιέχει ένα σύνολο κόμβων R.

– Ο κόμβος r_j καλείται απόγονος (descendant) του κόμβου r_i εάν υπάρχει ένα μονοπάτι στο T από τον r_i στον r_j.

– Καθώς οι κόμβοι r_i και r_j μπορεί να συμπίπτουν, ένας κόμβος είναι απόγονος του εαυτού του.

– Ο κόμβος r_j καλείται γνήσιος απόγονος (proper descendant) εάν υπάρχει ένα μονοπάτι p στο T από τον r_i στον r_j τέτοιο ώστε το μήκος του μονοπατιού p να είναι διάφορο του μηδενός.



Υλοποίηση γενικών δένδρων

```
// Κόμβος με πληροφορία τύπου int
struct Node {
    int key;
    vector<Node *> children;
};

// Δημιουργία κόμβου
Node *newNode(int newKey) {
    Node *temp = new Node;
    temp->key = newKey;
    return temp;
}
```

Υλοποιούμε το κενό δέντρο (T = [])
με nullptr

```
// Ύψος
int height(Node *r) {
    if (r == nullptr) return -1;
    int h = -1;
    for (Node *child: r->children)
        h = max(h, height(child));
    return h+1;
}

// Αναζήτηση
Node *searchTree(Node *r, int k) {
    if (r == nullptr) return nullptr;
    if (r->key == k) return r;
    for (Node *child: r->children) {
        Node *s = searchTree(child, k);
        if (s != nullptr) return s;
    }
    return nullptr;
}
```

Υλοποίηση γενικών δένδρων

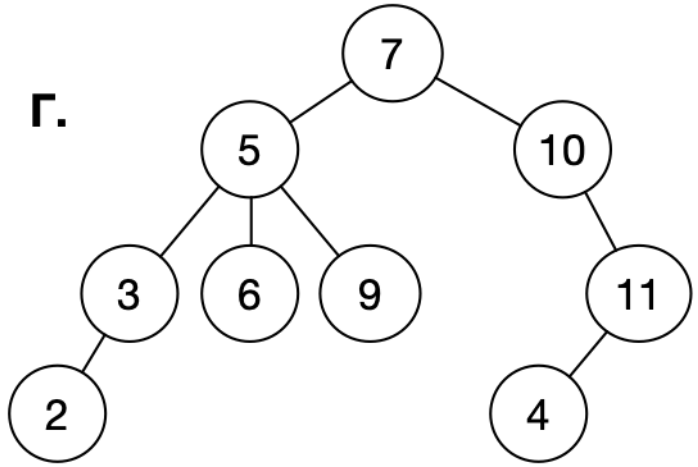
```
// Διάσχιση κατά πλάτος
void LevelOrderTraversal(Node * root) {
   if (root == nullptr) return;

   queue<Node *> q;
   q.push(root);
   while (!q.empty()) {
      int n = q.size();                    // τρέχον μέγεθος της ουράς q
      while (n > 0) {
         Node * p = q.front();             // αφαίρεση του πρώτου στοιχείου της ουράς
         q.pop(); n--;
         cout << p->key << " ";            // επεξεργασία, π.χ. εκτύπωση

         // Αν έχει παιδιά, βάλε τα στην ουρά
         for (Node *child : p->children)
            q.push(child);
      }
      cout << endl;                        // επόμενο επίπεδο
   }
}
```

Υλοποίηση γενικών δένδρων


```
Node *root = newNode(7);
root->children.push_back(newNode(5));
root->children.push_back(newNode(10));
root->children[0]->children.push_back(newNode(3));          // root->children[0] περιέχει το 5
root->children[0]->children.push_back(newNode(6));
root->children[0]->children.push_back(newNode(9));
root->children[0]->children[0]->children.push_back(newNode(2));
root->children[1]->children.push_back(newNode(11));         // root->children[0] περιέχει 10
root->children[1]->children[0]->children.push_back(newNode(4));

cout << "Level order traversal\n";
LevelOrderTraversal(root);
cout << "h=" << height(root) << endl;

Node *S = searchTree(root, 99);
if (S != nullptr)
   cout << "found\n";
else
   cout << "not found\n";
```

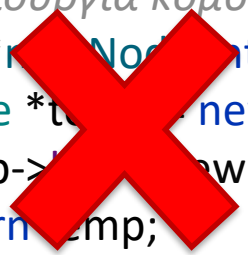
```
7
5 10
3 6 9 11
2 4
h=3
not found
```

Γ.



Υλοποίηση γενικών δένδρων (ξανά)

```
// Κόμβος με πληροφορία τύπου int
struct Node {
    int key;
    vector<Node *> children;

    Node(int k, vector<Node *> cs = {}) : key(k), children(cs) {}
};

// Δημιουργία κόμβου
Node *n
Node
nt newKey) {
    Node *t
new Node;
    temp->
ewKey;
    return
emp;
}
```



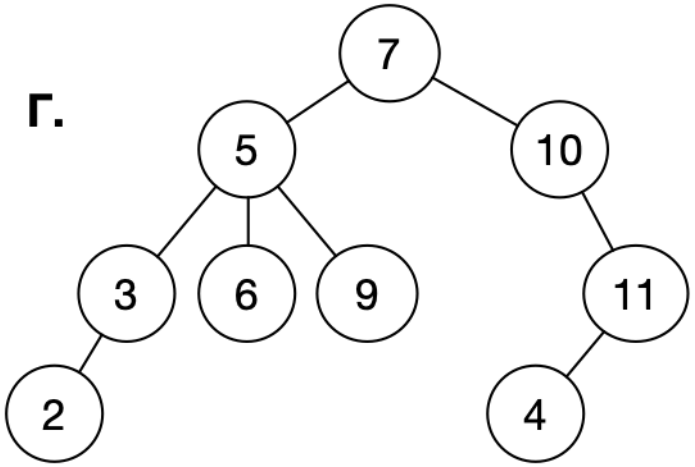
Υλοποίηση γενικών δένδρων, ξανά

```
Node *root = new Node(7, {
    new Node(5, {
        new Node(3, {
            new Node(2)
        }),
        new Node(6),
        new Node(9)
    }),
    new Node(10, { new Node(11, { new Node(4) }) })
});
```


```
cout << "Level order traversal\n";
LevelOrderTraversal(root);
cout << "h=" << height(root) << endl;

Node *S = searchTree(root, 99);
if (S != nullptr)
   cout << "found\n";
else
   cout << "not found\n";
```

```
7
5 10
3 6 9 11
2 4
h=3
not found
```

Γ.



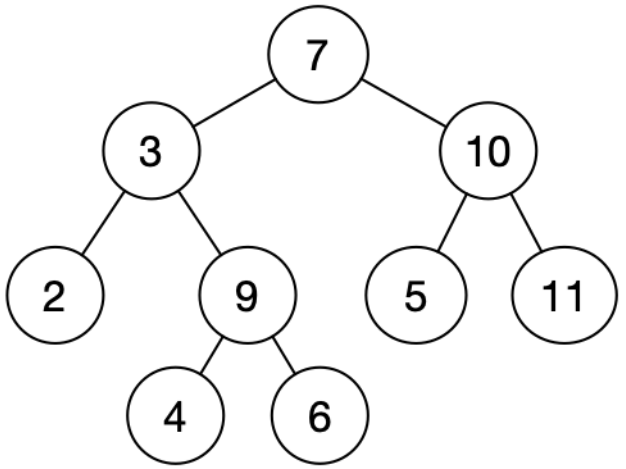
Δυαδικά δένδρα

- Ένα δυαδικό δένδρο Τ είναι ένα δένδρο του οποίου κάθε κόμβος αποτελείται από μία ρίζα, r, και αποκλειστικά και μόνο από δύο επιπλέον δυαδικά δένδρα T_L και T_R δηλαδή:
- $T = [r ; T_L, T_R]$
- Το δένδρο T_L ονομάζεται αριστερό υποδένδρο του Τ, και το δένδρο T_R ονομάζεται δεξί υποδένδρο του Τ.
- Προφανώς, τα δυαδικά δένδρα είναι υποπερίπτωση των δένδρων $T = [r ; T_1, T_2, \ldots, T_n]$, $n \geq 0$ με $n=2$ παντού.

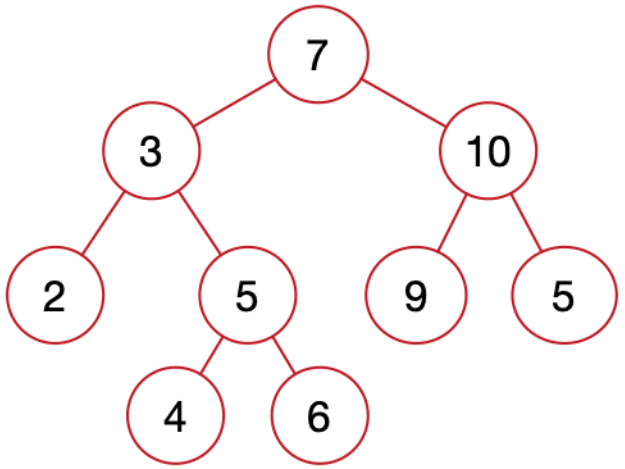
- Θυμηθείτε ότι τα T_1 και T_2 μπορούν να είναι κενά δένδρα! (και προφανώς για αυτά δεν είναι n=2)

Δυαδικά δένδρα

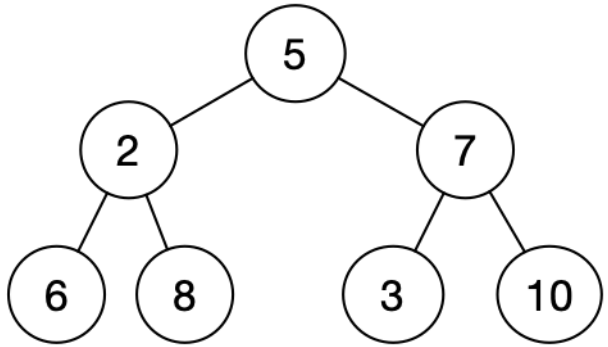
Α.



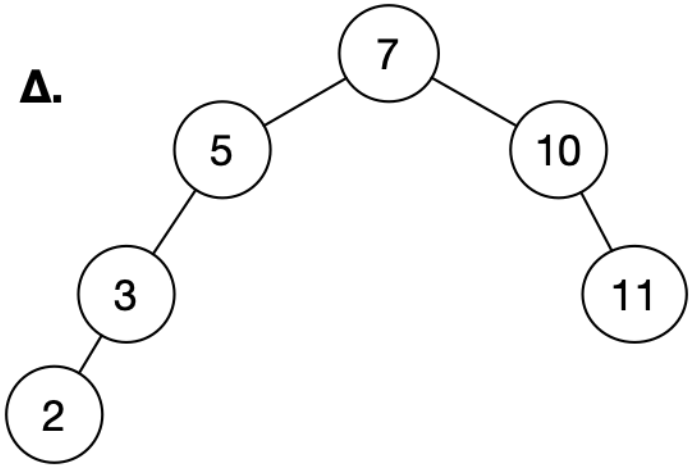
Β.



Γ.

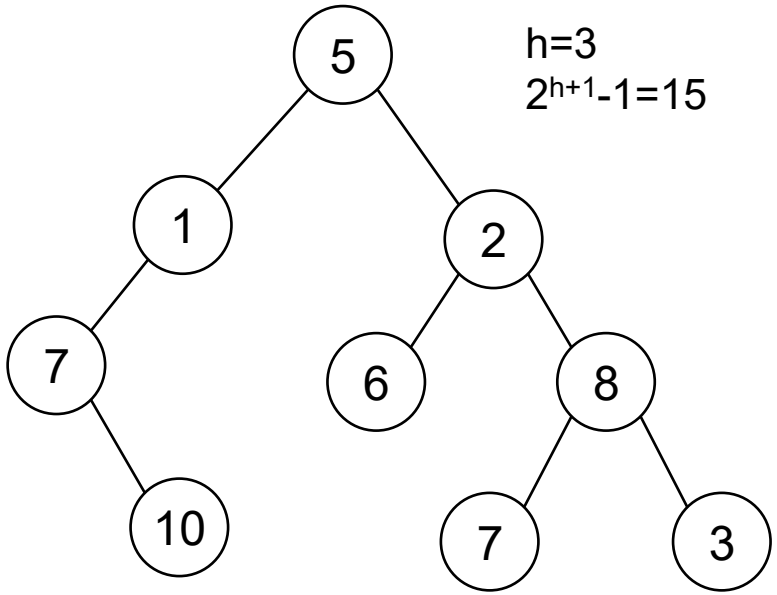


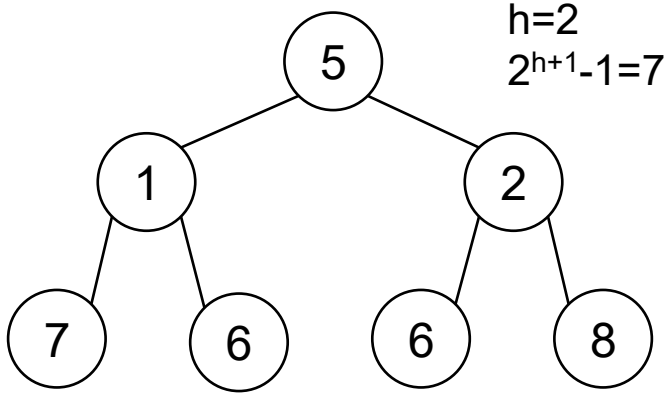
Δ.



Ιδιότητες δυαδικών δένδρων

- Ένα δυαδικό δένδρο ύψους h έχει το πολύ 2^{h+1} -1 κόμβους.
- Αντιστρόφως, το ύψος ενός δυαδικού δένδρου με n κόμβους είναι τουλάχιστον $\lceil \log_2 n + 1 \rceil - 1$.
- Δηλαδή το ύψος ενός δυαδικού δένδρου το οποίο έχει n κόμβους είναι της τάξης Ω(log n).
- Ένα δυαδικό δένδρο ύψους $h \geq 0$ έχει το πολύ 2^h φύλλα.
- Αντιστρόφως, το ύψος ενός δυαδικού δένδρου το οποίο έχει k φύλλα είναι τουλάχιστον $\lceil \log_2 k \rceil$.
- Έτσι, το ύψος ενός δυαδικού δένδρου με k φύλλα είναι της τάξης Ω(log k).

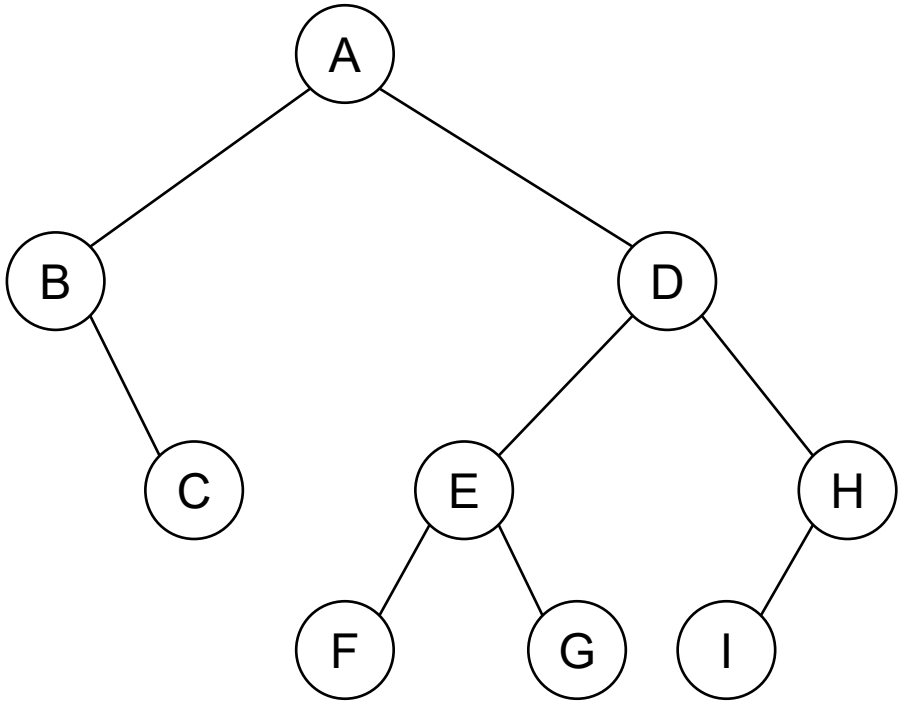




Διάσχιση δένδρων

- Η διαδικασία κατά την οποία συστηματικά επισκεπτόμαστε όλους τους κόμβους του δένδρου ονομάζεται διάσχιση δένδρου (tree traversal).
- Δύο διαφορετικές μέθοδοι για να επισκεφθεί κανείς συστηματικά όλους τους κόμβους ενός δένδρου:
  - διάσχιση κατά βάθος (depth-first traversal)
  - διάσχιση κατά πλάτος (breadth-first traversal)
- Μέθοδοι κατά βάθος διάσχισης:
  - προδιατεταγμένη διάσχιση (preorder traversal)
  - ενδοδιατεταγμένη διάσχιση (inorder traversal)
  - μεταδιατεταγμένη διάσχιση (postorder traversal)

Διάσχιση δένδρων



Βάθος = 0

Βάθος = 1

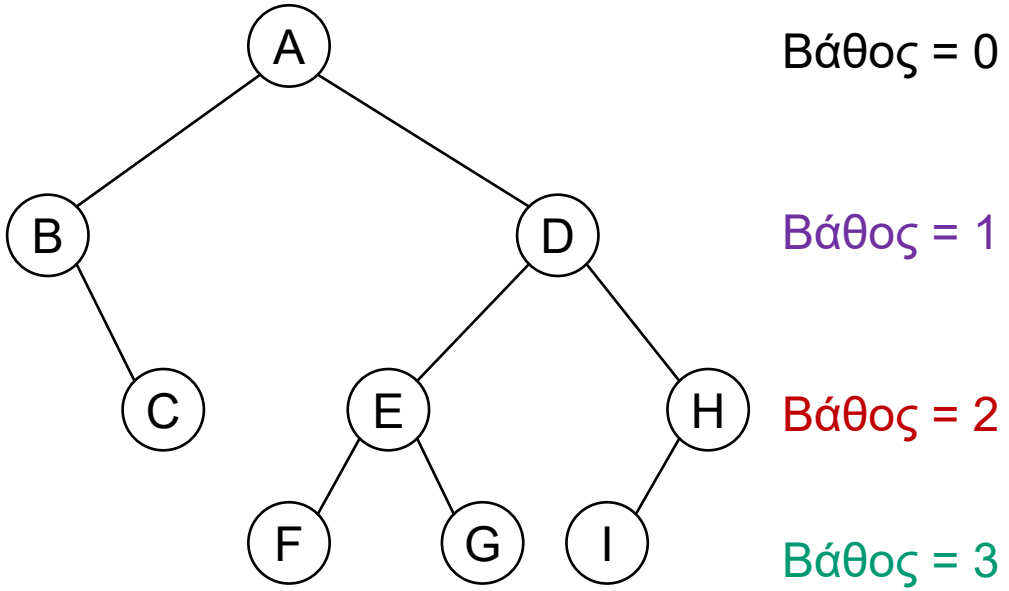
Βάθος = 2

Βάθος = 3

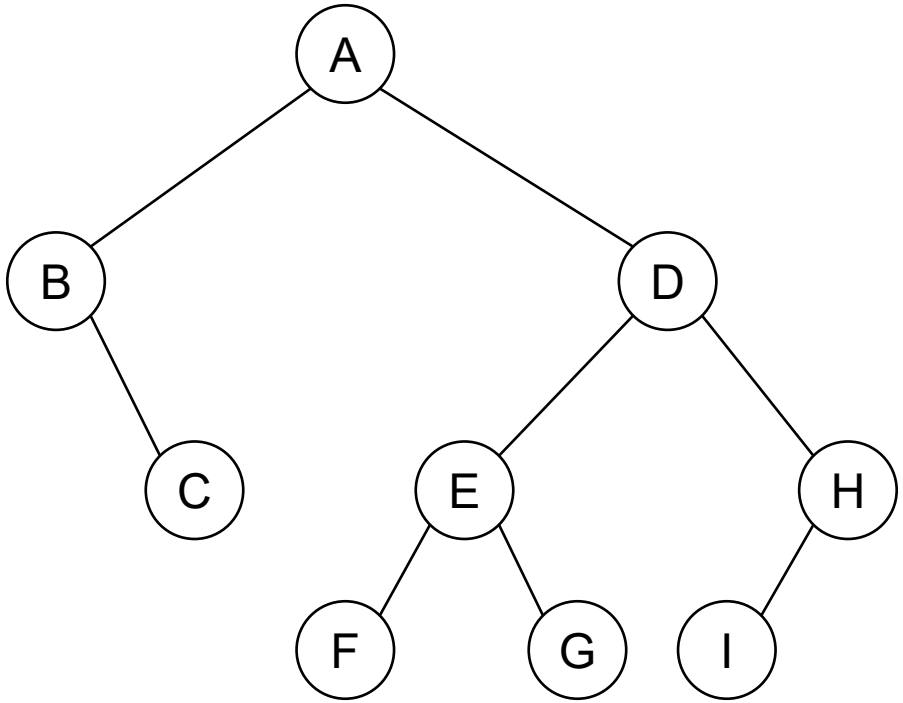
Διάσχιση κατά βάθος

- Κατά την προδιατεταγμένη διάσχιση ενός γενικού δένδρου:
  - Επισκεπτόμαστε πρώτα τη ρίζα και στη συνέχεια
  - Εκτελούμε μία προδιατεταγμένη διάσχιση διαδοχικά σε κάθε ένα από τα υποδένδρα της ρίζας, με τη σειρά που δίνονται

- Στην περίπτωση ενός δυαδικού δένδρου:
  - Επισκεπτόμαστε πρώτα τη ρίζα
  - Διασχίζουμε το αριστερό υποδένδρο
  - Διασχίζουμε το δεξί υποδένδρο



Διάσχιση δένδρων (προδιατεταγμένη)



Βάθος = 0

Βάθος = 1

Βάθος = 2

Βάθος = 3

A, B, C, D, E, F, G, H, I

Διάσχιση δένδρων (προδιατεταγμένη)

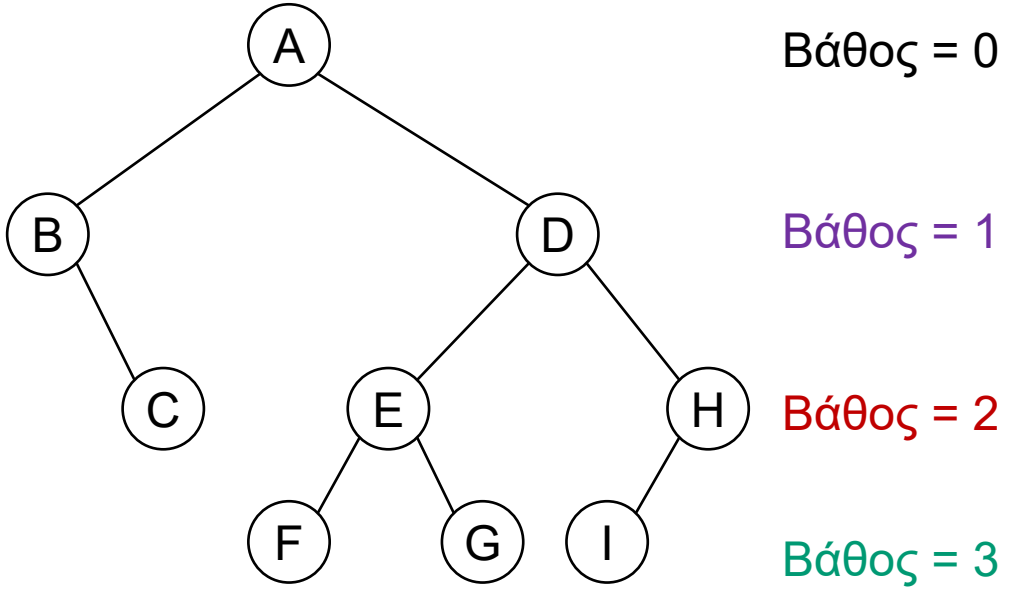
```
struct Node {
    char data;
    Node *left, *right;

    Node(char c, Node *l = nullptr, Node *r = nullptr)
        : data(c), left(l) right(r) {}
};

void printPreorder(Node* node) {
    if (node == nullptr) return;
    cout << node->data << " ";     // visit node
    printPreorder(node->left);     // visit left subtree
    printPreorder(node->right);    // visit right subtree
}
```

```
void preorderDemo() {
    Node* root =
        new Node('A',
            new Node('B', nullptr, new Node('C')),
            new Node('D',
                new Node('E', new Node('F'), new Node('G')),
                new Node('H', new Node('I'))
            )
        );
    cout << "Depth-first preorder traversal of binary tree \n";
    printPreorder(root);
}
```

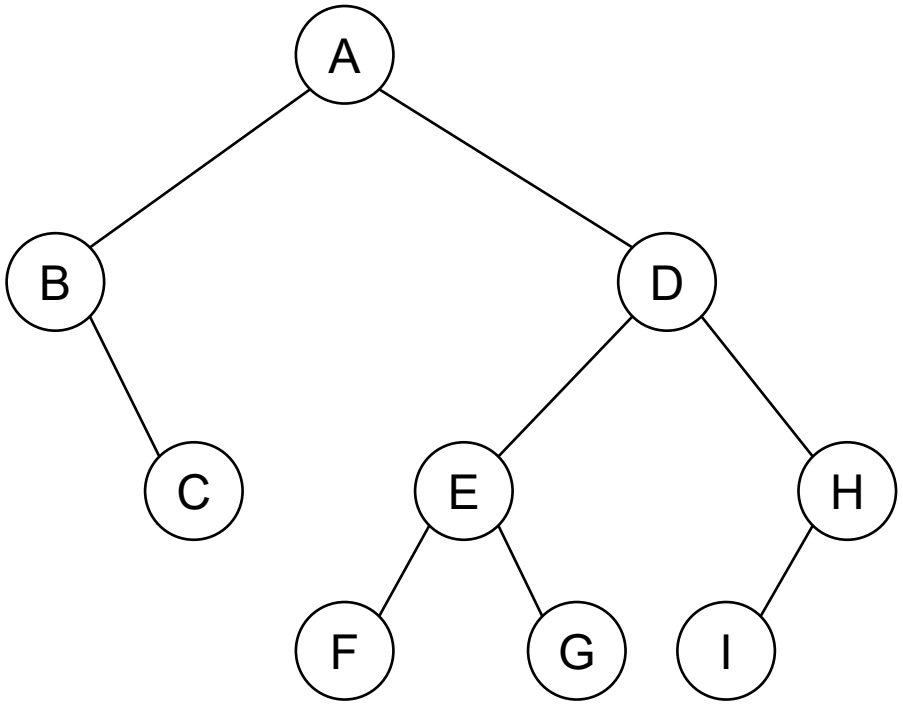
Διάσχιση κατά βάθος

- Στη μεταδιατεταγμένη διάσχιση ενός γενικού δένδρου:
  - Εκτελούμε μία μεταδιατεταγμένη διάσχιση διαδοχικά σε κάθε ένα από τα υποδένδρα της ρίζας με τη σειρά που αυτά δίνονται
  - Επισκεπτόμαστε τη ρίζα

- Στην περίπτωση ενός δυαδικού δένδρου:
  - Διασχίζουμε το αριστερό υποδένδρο
  - Διασχίζουμε το δεξί υποδένδρο
  - Επισκεπτόμαστε τη ρίζα



Διάσχιση δένδρων (μεταδιατεταγμένη)



Βάθος = 0

Βάθος = 1

Βάθος = 2

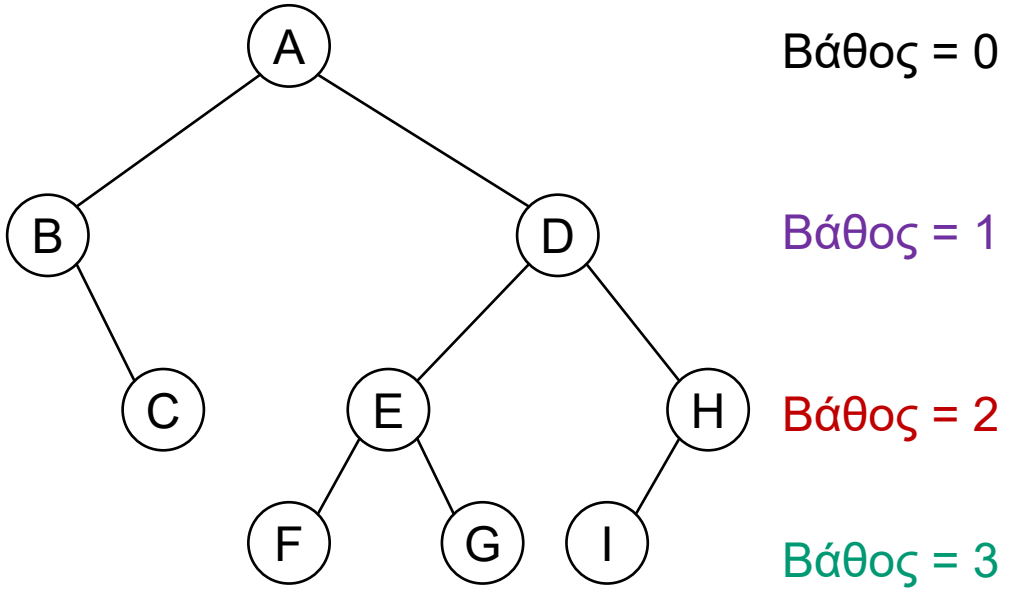
Βάθος = 3

```
void printPostorder(Node* node) {
   if (node == nullptr) return;
   printPostorder(node->left);     // visit left subtree
   printPostorder(node->right);   // visit right subtree
   cout << node->data << " ";     // visit node
}
```

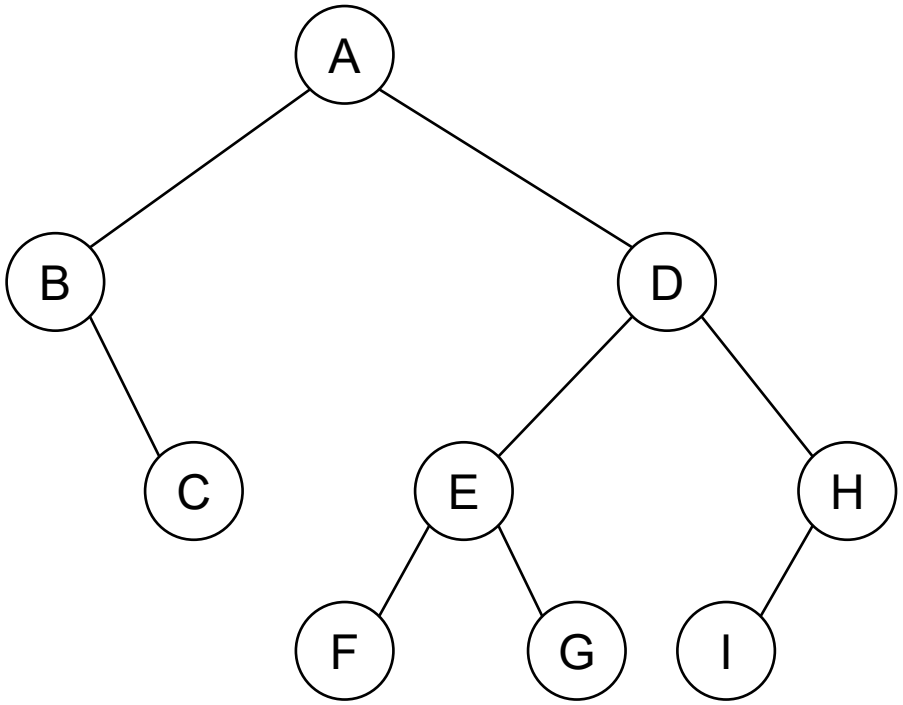
C, B, F, G, E, I, H, D, A.

Διάσχιση κατά βάθος

- Η ενδοδιατεταγμένη διάσχιση έχει νόημα μόνο στην περίπτωση ενός δυαδικού δένδρου:
  - Διασχίζουμε το αριστερό υποδένδρο
  - Επισκεπτόμαστε τη ρίζα
  - Διασχίζουμε το δεξί υποδένδρο.



Διάσχιση δένδρων



Βάθος = 0

Βάθος = 1

Βάθος = 2

Βάθος = 3

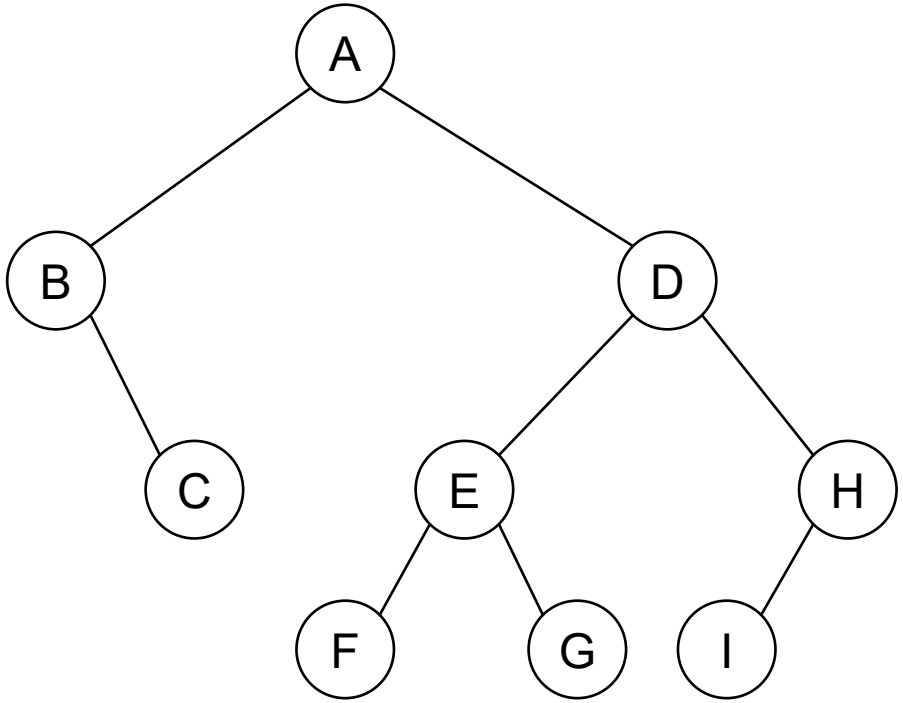
```
void printInorder(Node* node) {
   if (node == nullptr) return;
   printInorder(node->left);      // visit left subtree
   cout << node->data << " ";     // visit node
   printInorder(node->right);     // visit right subtree
}
```

B, C, A, F, E, G, D, I, H.

Διάσχιση κατά πλάτος

- Ενώ οι μέθοδοι διάσχισης κατά βάθος ορίζονται αναδρομικά, η αναζήτηση κατά πλάτος (breadth-first traversal) γίνεται καλύτερα κατανοητή χωρίς τη χρήση αναδρομής.
- Για να εκτελέσουμε τη διάσχιση ενός δένδρου κατά πλάτος, επισκεπτόμαστε τους κόμβους του δένδρου κατά σειρά βάθους.
- Η διάσχιση κατά πλάτος πρώτα επισκέπτεται όλους τους κόμβους που έχουν βάθος 0 (δηλαδή τη ρίζα), στη συνέχεια όλους τους κόμβους που έχουν βάθος 1 και ούτω καθεξής.
- Για κάθε τιμή βάθους επισκεπτόμαστε τους κόμβους από τα αριστερά προς τα δεξιά.

Διάσχιση κατά πλάτος



Βάθος = 0

Βάθος = 1

Βάθος = 2

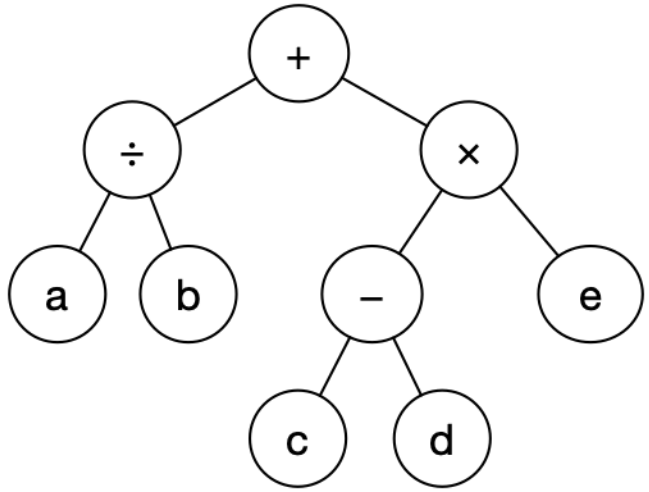
Βάθος = 3

A, B, D, C, E, H, F, G, I.

Διάσχιση κατά πλάτος

```
void printBForder(Node* root) {
    if (root == nullptr) return;
    queue<Node *> q;                    // Create an empty queue
    q.push(root);                       // Enqueue root
    while (!q.empty()) {
        Node *node = q.front();
        cout << node->data << " ";      // Print front of queue and remove it from queue
        q.pop();
        if (node->left != nullptr)
            q.push(node->left);         // Enqueue left child
        if (node->right != nullptr)
            q.push(node->right);        // Enqueue right child
    }
}
```

Δένδρα εκφράσεων



- Διάσχιση
  - προδιατεταγμένη `+÷ab×−cde` `+(÷(a,b),×(−(c,d),e))` *προθεματική*
  - ενδοδιατετατμένη `a÷b+c−d×e` `((a÷b) + ((c−d) × e))` *ενδοθεματική*
  - μεταδιατεταγμένη `ab÷cd−e×+` *μεταθεματική*

Δυαδικά δένδρα αναζήτησης

Γιώργος Στάμου

Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Εισαγωγή στα δυαδικά δένδρα αναζήτησης

- Μέχρι τώρα θεωρήσαμε δένδρα στα οποία οι σχετικές θέσεις των κόμβων μέσα στο δένδρο δεν έχουν περιορισμούς.
- Δηλαδή, ένα αντικείμενο μπορεί να εμφανίζεται οπουδήποτε μέσα στο δένδρο: δε γνωρίζουμε προς τα πού να αναζητήσουμε το αντικείμενο μέσα στο δένδρο.
- Μπορεί να χρειαστεί να κάνουμε μία πλήρη διάσχιση του δένδρου (στη χειρότερη περίπτωση) προκειμένου να το βρούμε.
- Προκειμένου να διευκολύνουμε την αναζήτηση, μπορούμε να βάλουμε περιορισμούς ως προς τις σχετικές θέσεις των αντικειμένων μέσα στο δένδρο.

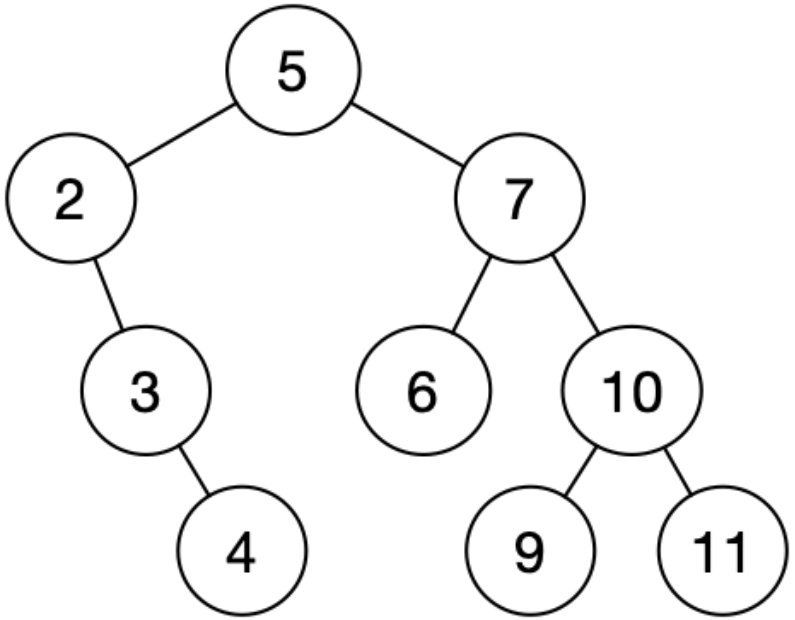
Δένδρα αναζήτησης

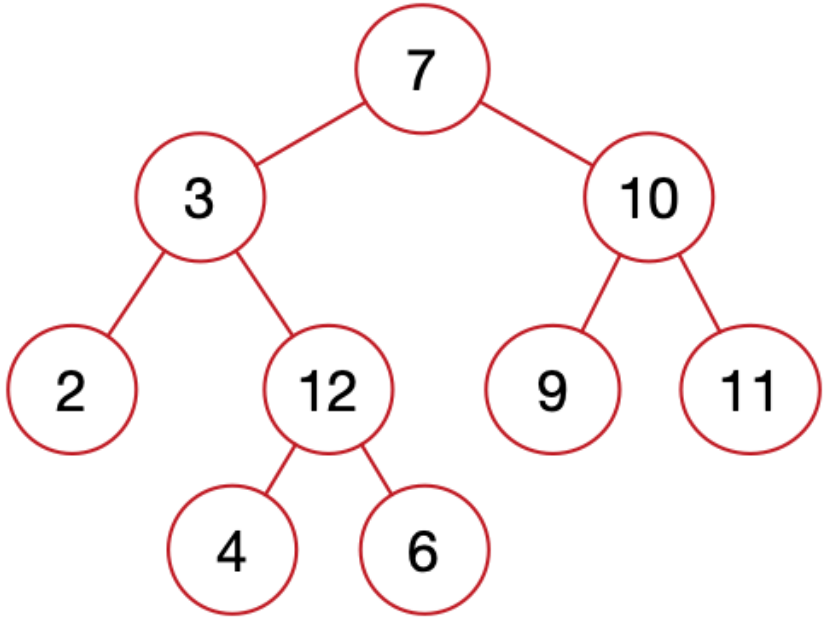
- Ένα δένδρο το οποίο υποστηρίζει αποδοτικές λειτουργίες αναζήτησης, εισαγωγής και διαγραφής ονομάζεται *δένδρο αναζήτησης*.
- Τα δένδρα αναζήτησης χρησιμοποιούνται προκειμένου να αποθηκεύσουν ένα πεπερασμένο σύνολο κλειδιών που προέρχονται από ένα *ολικά διατεταγμένο* σύνολο κλειδιών K.
- Αυτό που κάνει ένα δένδρο να είναι δένδρο αναζήτησης είναι το ότι τα κλειδιά δεν εμφανίζονται σε αυθαίρετους κόμβους του δένδρου: υπάρχει ένα *κριτήριο διάταξης δεδομένων* το οποίο καθορίζει το πού μπορεί να εμφανιστεί ένα κλειδί μέσα στο δένδρο σε σχέση με τα άλλα κλειδιά του δένδρου.

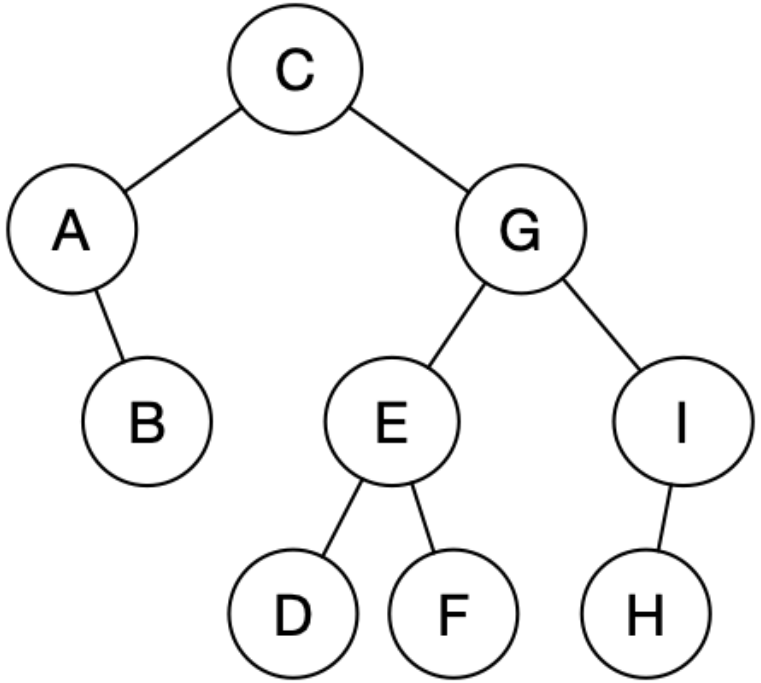
Δυαδικό δένδρο αναζήτησης – Ορισμός

- Ένα δυαδικό δένδρο αναζήτησης T (Binary Search Tree - BST) είναι ένα δυαδικό δένδρο $T = \{r; T_L, T_R\}$ με τις εξής ιδιότητες:
  - Όλοι οι κόμβοι που περιέχονται στο αριστερό υποδένδρο, T_L , έχουν ρίζες μικρότερες από το r, δηλαδή $\forall k \in T_L: k < r$.
  - Όλοι οι κόμβοι που περιέχονται στο δεξί υποδένδρο, T_R , έχουν ρίζες μεγαλύτερες από το r, δηλαδή $\forall k \in T_R: k > r$.
  - Τα T_L, T_R είναι δυαδικά δένδρα αναζήτησης.

Παραδείγματα BST







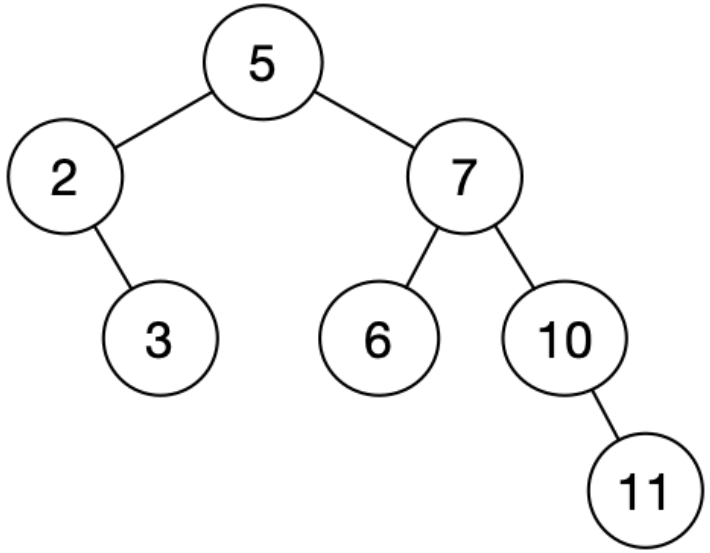
Αναζήτηση στοιχείων σε BST

- Η αναζήτηση ξεκινά από τη ρίζα του δένδρου.
- Εάν το στοιχείο που αναζητούμε, x , είναι ίσο με τη ρίζα r, η αναζήτηση τερματίζεται επιτυχώς.
- Εάν όχι, τότε, εάν το x είναι μικρότερο από το r, θα γίνει αναζήτηση στο αριστερό υποδένδρο, αλλιώς, το x πρέπει να είναι μεγαλύτερο από το r και στην περίπτωση αυτή θα γίνει αναζήτηση στο δεξί υποδένδρο.

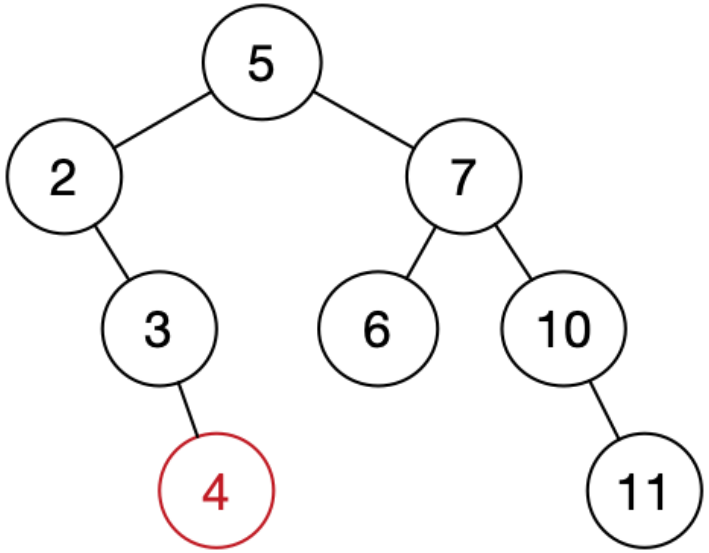
Εισαγωγή στοιχείων σε BST

- Ακολουθούμε το μονοπάτι που θα ακολουθούσε η μέθοδος αναζήτησης, προκειμένου να προσδιορίσουμε πού θα ήταν το στοιχείο (εάν αυτό υπήρχε στο δένδρο).
- Υποθέτοντας ότι το στοιχείο δεν υπάρχει ήδη στο δένδρο, η αναζήτηση θα είναι ανεπιτυχής και θα τερματίσει σε ένα κενό υποδένδρο.
- Αυτή ακριβώς είναι η θέση στην οποία θα πρέπει να εισαχθεί το νέο στοιχείο (ως φύλλο αντί για το κενό υποδένδρο)!

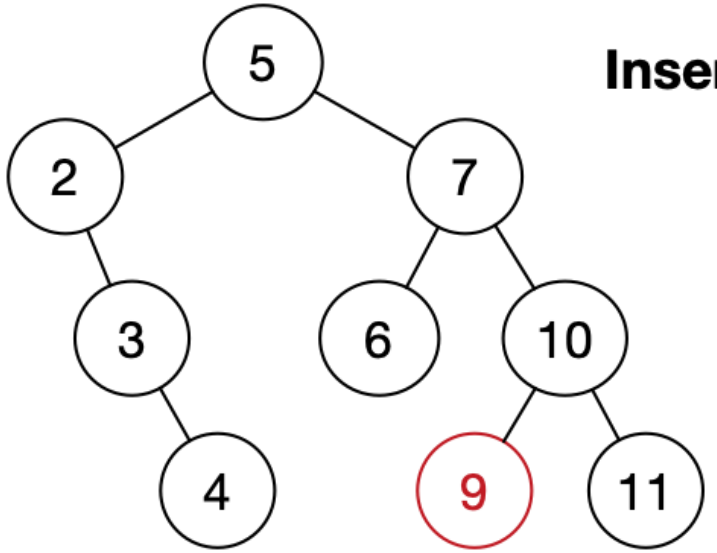
Εισαγωγή στοιχείων σε BST



Insert 4



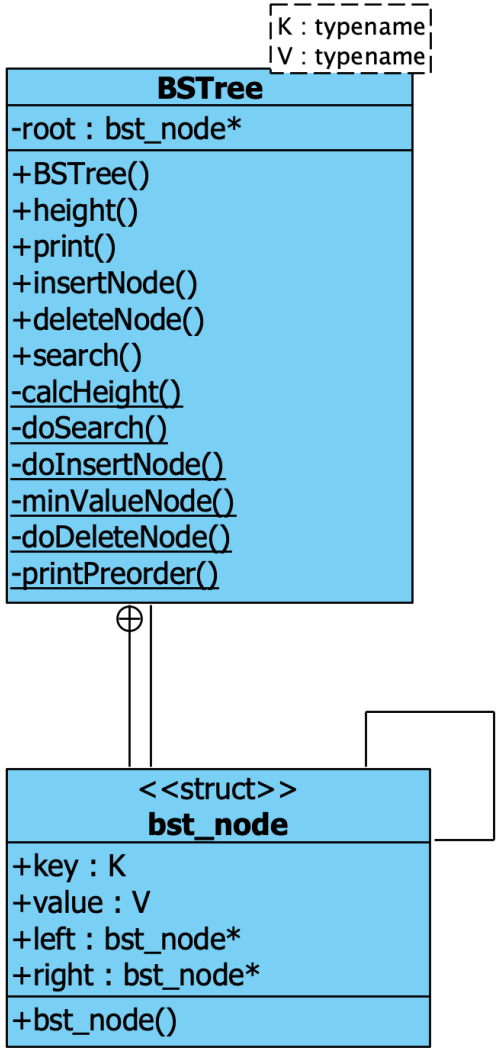
Insert 9



Υλοποίηση BST με template

```
template <typename K, typename V>
class BSTree {
 public:
  BSTree(): root(nullptr) {}
  int height() const { return calcHeight(root); }
  void print() const { printPreorder(root); cout << "\nh= " << height() << endl; }
  void insertNode(const K &k, const V &v) { root = doInsertNode(root, k, v); }
  void deleteNode(const K &k) { root = doDeleteNode(root, k); }
  V & search(const K &k) const {
     bst_node *s = doSearch(root, k);
     if (s == nullptr) throw "Not found";
     return s->value;
  }

 private:
  struct bst_node {
     K key;
     V value;
     bst_node *left, *right;
     bst_node(const K &k, const V &v) : key(k), value(v), left(nullptr), right(nullptr) {}
  };

  bst_node *root;
```

Σε αυτό το παράδειγμα,
σε κάθε κόμβο αποθηκεύεται
το κλειδί τύπου K
και μία τιμή τύπου V



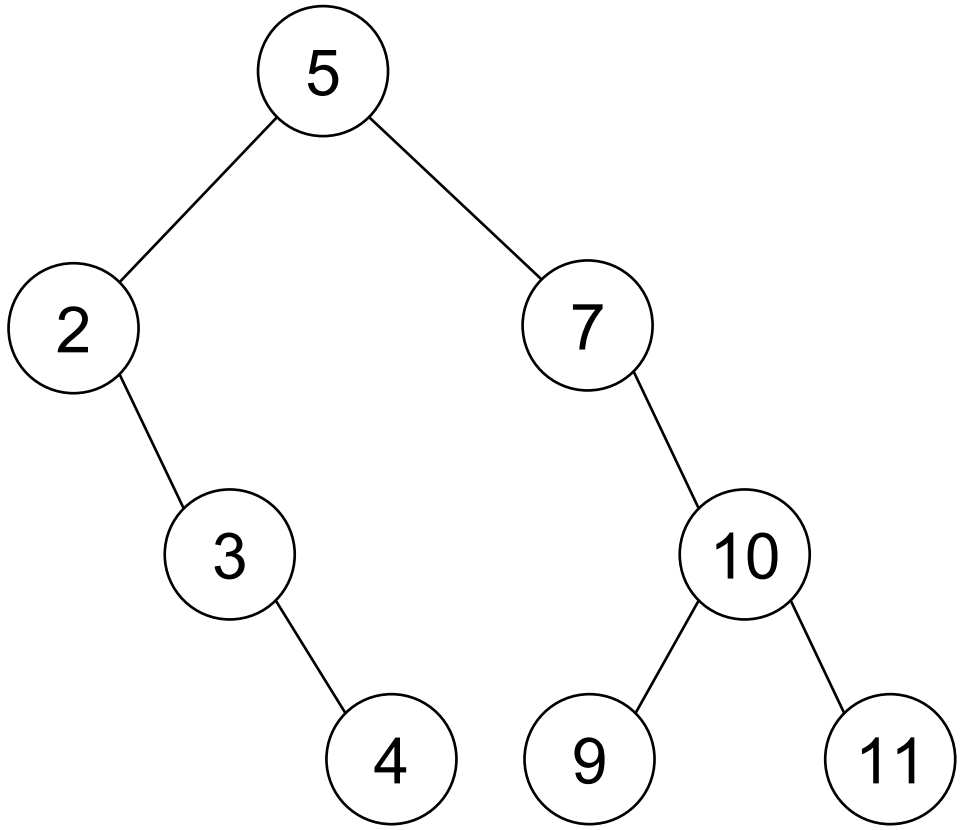
Υλοποίηση BST

```
private:
 struct bst_node {
    K key;
    V value;
    bst_node *left, *right;
    bst_node(const K &k, const V &v)
       : key(k), value(v), left(nullptr), right(nullptr) {}
 };

 bst_node *root;

 static int calcHeight(bst_node *r) {
    if (r == nullptr)
       return -1;
    return 1 + max(calcHeight(r->left), calcHeight(r->right));
 }
```

```
static bst_node * doSearch(bst_node *r, const K &k) {
   if (r == nullptr)
      return nullptr;
   if (r->key == k)
      return r;
   if (k < r->key)
      return doSearch(r->left, k);
   else
      return doSearch(r->right, k);
}

static bst_node * doInsertNode(bst_node *r, const K &k, const V &v) {
   if (r == nullptr)
      return new bst_node(k, v);
   if (r->key == k)
      r->value = v;          // update if it exists
   else if (k < r->key)
      r->left = doInsertNode(r->left, k, v);
   else
      r->right = doInsertNode(r->right, k, v);
   return r;
}
```

Υλοποίηση BST

```
BSTree<int, string> t;
t.insertNode(5, "five");
t.insertNode(2, "two");
t.insertNode(3, "three");
t.insertNode(7, "seven");
t.insertNode(4, "four");
t.insertNode(10, "ten");
t.insertNode(9, "nine");
t.insertNode(11, "eleven");
t.print();

try {
    cout << "Searching 1... " << t.search(1) << " found!"<< endl;
} catch (const char *exc) {
    cout << "\n  Exception searching 1: " << exc << endl;
}
try {
    cout << "Searching 9... " << t.search(9) << " found!"<<endl;
} catch (const char *exc) {
    cout << "\n  Exception searching 9: " << exc << endl;
}
```

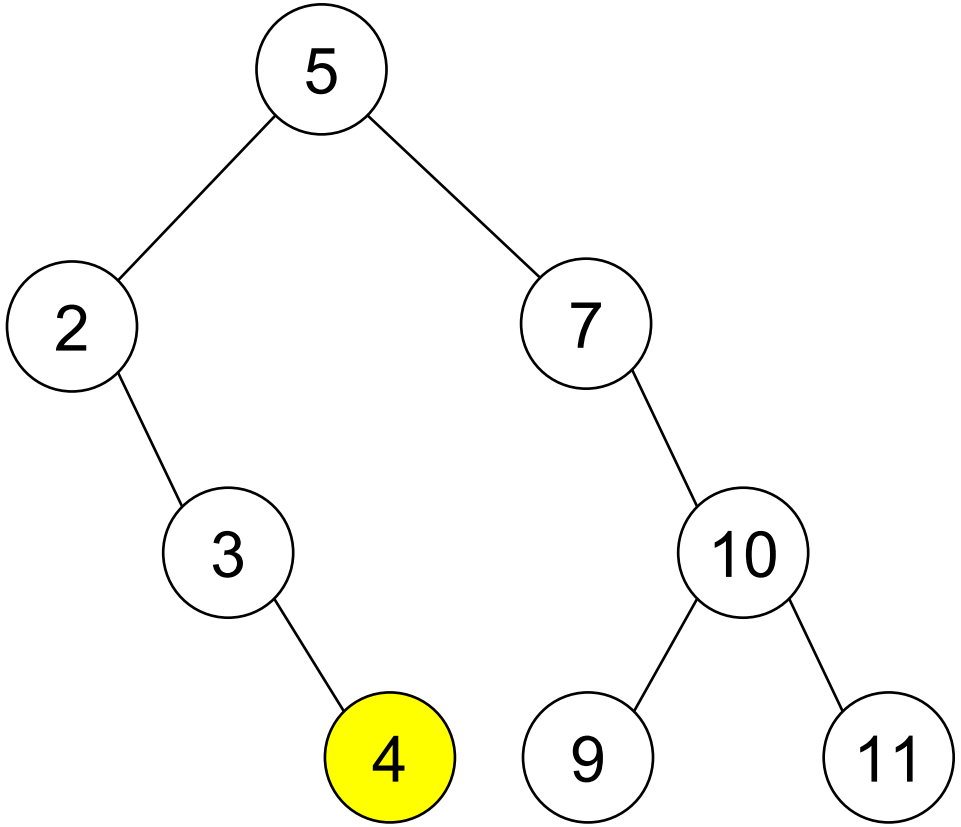


Διαγραφή στοιχείων από BST

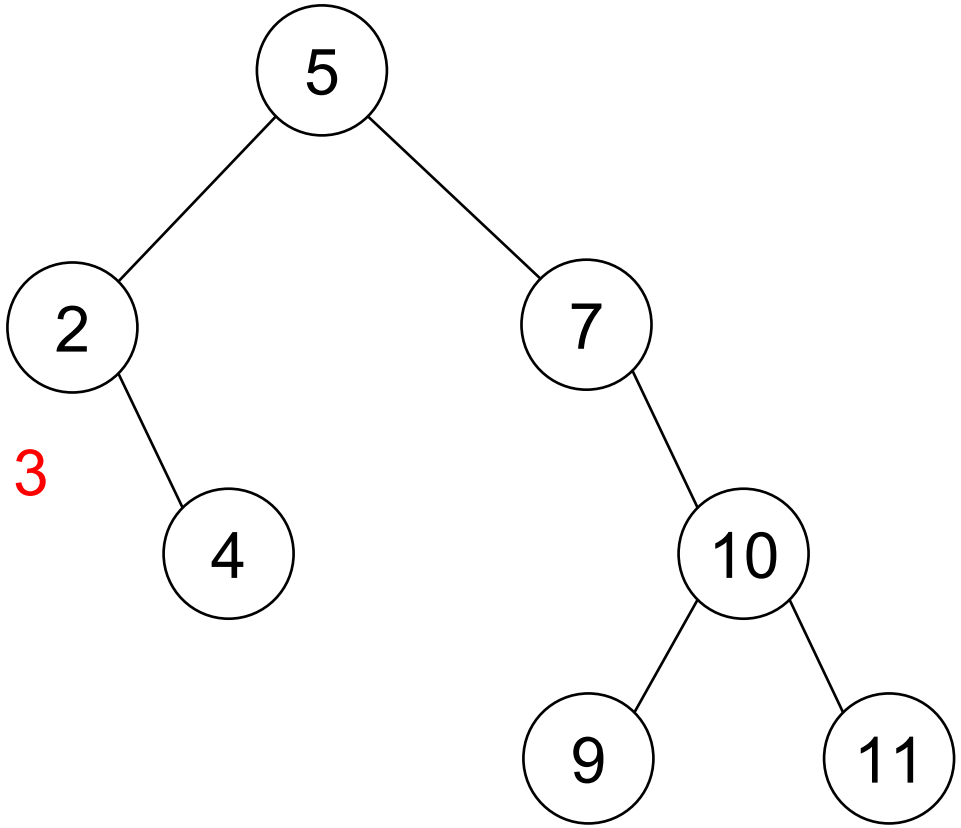
- Ακολουθούμε το μονοπάτι αναζήτησης, προκειμένου να προσδιορίσουμε το στοιχείο (εάν υπάρχει στο δένδρο).
- Αν το στοιχείο υπάρχει στο δένδρο, η αναζήτηση θα είναι επιτυχής τερματίζοντας στον κόμβο που θα πρέπει να διαγραφεί.
- Αν ο κόμβος είναι φύλλο τον διαγράφουμε, μετατρέποντάς τον σε κενό υποδένδρο.
- Αν ο κόμβος αυτός έχει ένα παιδί τον διαγράφουμε, αντικαθιστώντας τον με το μη κενό παιδί του.
- Αν έχει δύο παιδιά, αντικαθιστούμε τη ρίζα του με τον κόμβο με την ελάχιστη τιμή του δεξιού υποδένδρου και διαγράφουμε εκείνον

ή και με τη μέγιστη τιμή του αριστερού υποδένδρου!

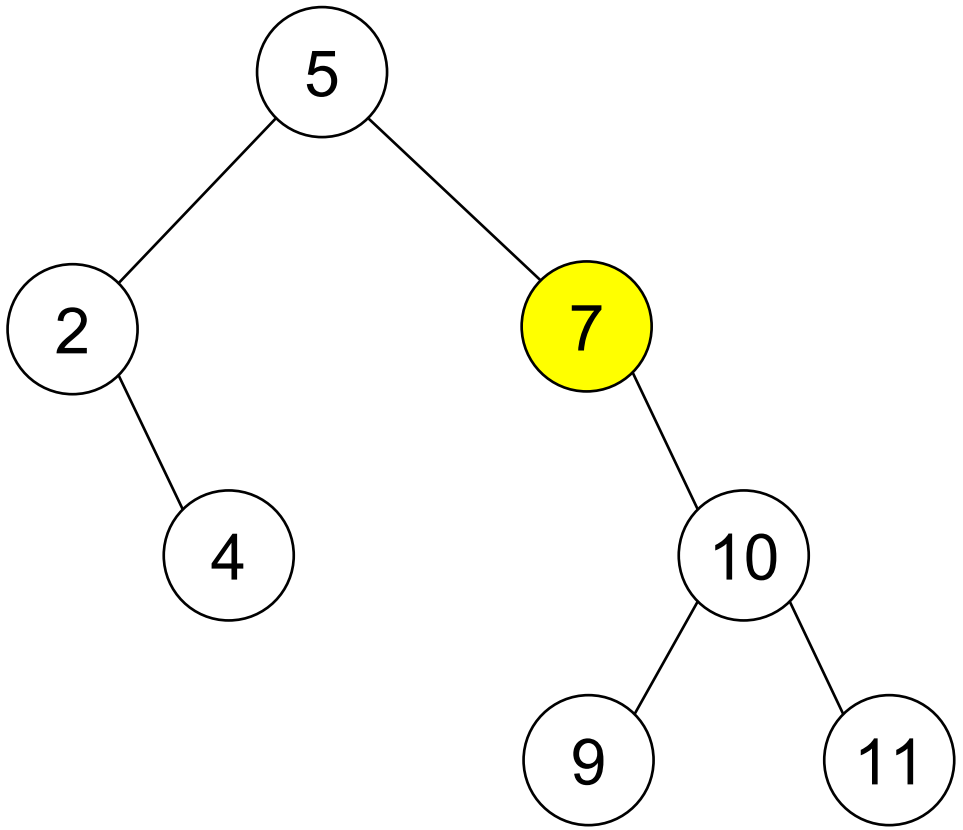
Διαγραφή στοιχείων από BST



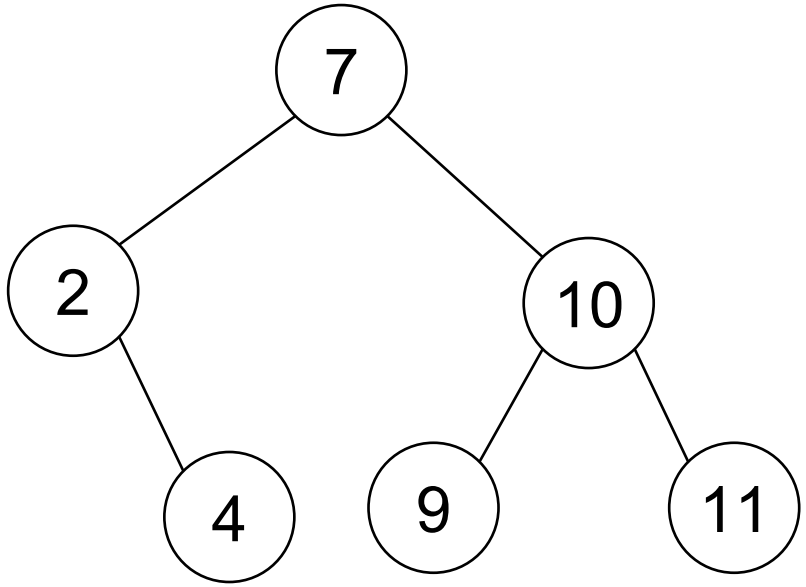
delete 3



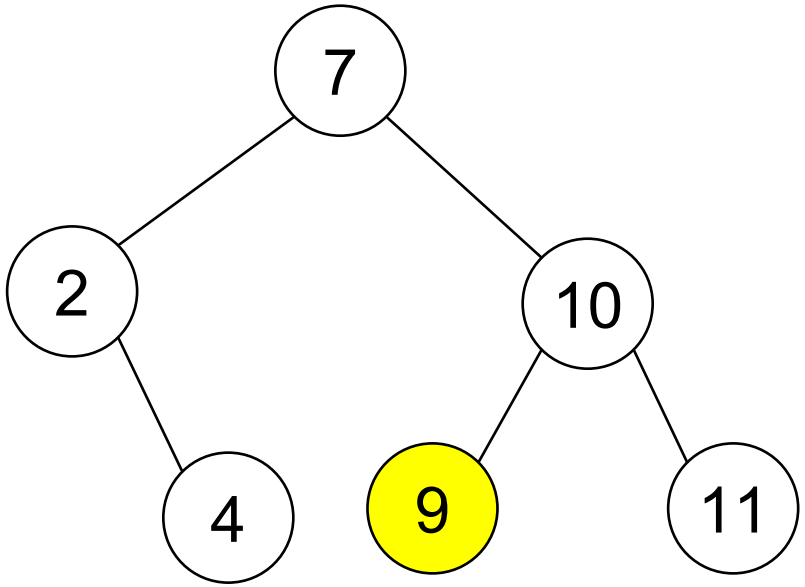
Διαγραφή στοιχείων από BST



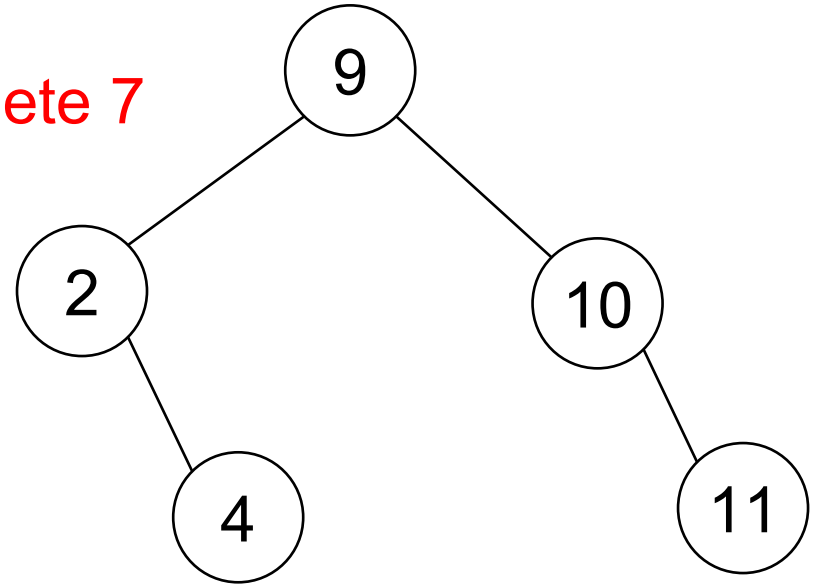
delete 5



Διαγραφή στοιχείων από BST



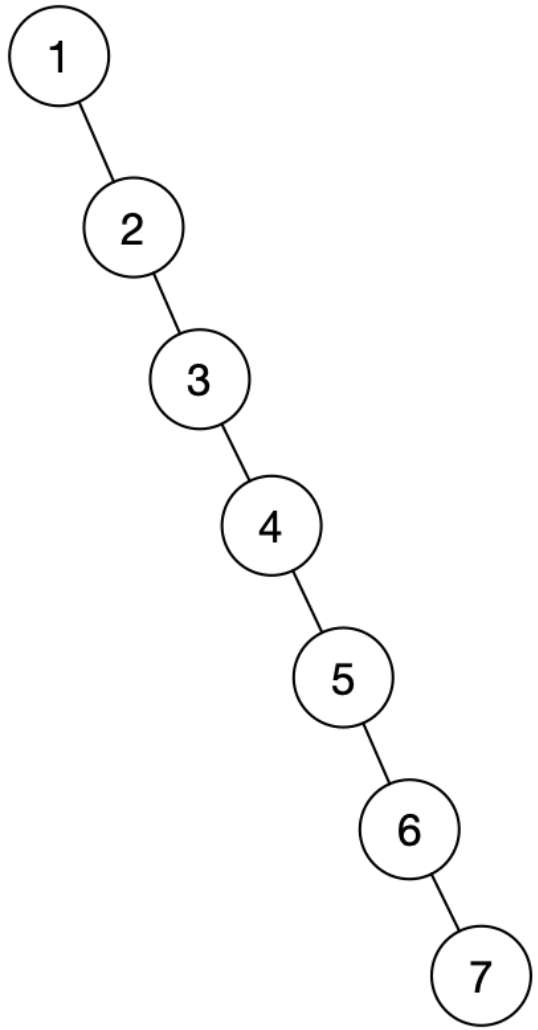
delete 7

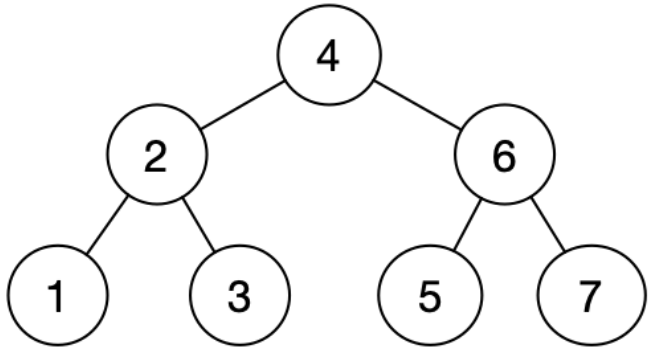


Διαγραφή στοιχείων από BST

```
static bst_node* doDeleteNode(bst_node* n, const K &key) {
   if (n == nullptr) return n;

   if (key < n->key)
      n->left = doDeleteNode(n->left, key);
   else if (key > n->key)
      n->right = doDeleteNode(n->right, key);
   else {
      // this is the node to delete
      if (n->left == nullptr && n->right == nullptr) {
         delete n;       // it's a leaf, just delete it!
         return nullptr;
      } else if (n->left == nullptr) {
         bst_node* temp = n->right;
         delete n;       // it has one right child, replace it with that!
         return temp;
      } else if (n->right == nullptr) {
         bst_node* temp = n->left;
         delete n;       // it has one left child, replace it with that!
         return temp;
      }
      // the node has two children, find its inorder successor
      bst_node* temp = minValueNode(n->right);

      // copy the inorder successor's content to this node
      n->key = temp->key;
      n->value = temp->value;
      // delete the inorder successor
      n->right = doDeleteNode(n->right, temp->key);
   }
   return n;
}

static bst_node *minValueNode(bst_node *n) {
   while (n->left != nullptr) n = n->left;
   return n;
}
```

Παράδειγμα αναζήτησης σε BST – Πολυπλοκότητα





Τέλειο δυαδικό δένδρο – Ορισμός

- Ένα τέλειο δυαδικό δένδρο ύψους h ≥ 0
  είναι ένα δυαδικό δένδρο $T = \{r; T_L, T_R\}$ με τις παρακάτω ιδιότητες:
  - Εάν h=0, τότε T_L και T_R κενά.
  - Αλλιώς, αν h>0, τα T_L και T_R είναι και τα δύο τέλεια δυαδικά δένδρα ύψους h – 1.
- Για h = -1
  - Το κενό δένδρο είναι τέλειο δυαδικό δένδρο με ύψος -1

Τέλειο δυαδικό δένδρο – Ιδιότητες

- Είναι σχετικά εύκολο να δείξουμε ότι ένα τέλειο δυαδικό δένδρο ύψους h έχει ακριβώς $2^{h+1} – 1$ κόμβους.
- Αντιστρόφως, το ύψος ενός τέλειου δυαδικού δένδρου με n εσωτερικούς κόμβους είναι $\log_2 (n + 1) – 1$.
- Εάν έχουμε ένα δένδρο αναζήτησης το οποίο έχει το σχήμα ενός τέλειου δυαδικού δένδρου, τότε κάθε ανεπιτυχής αναζήτηση επισκέπτεται ακριβώς h+1 το πλήθος εσωτερικούς κόμβους.
- Ο χρόνος εκτέλεσης για μία ανεπιτυχή αναζήτηση σε ένα τέλειο δένδρο, στην χειρότερη περίπτωση είναι της τάξης O(log n).

Τέλειο δυαδικό δένδρο – Ιδιότητες

- Συνεπώς η συνθήκη του τέλειου δένδρου διασφαλίζει αποτελεσματικές και αποδοτικές αναζητήσεις.
- Μπορούμε να κατασκευάσουμε τέλεια δένδρα με n κόμβους **μόνο** για n = 1, 3, 7, 15, 31, 63,…
- Άρα, τελικά η συνθήκη του τέλειου δένδρου είναι μία ακατάλληλη (πρακτικά) συνθήκη ισορροπίας, επειδή δεν μπορούμε να κατασκευάσουμε ένα ισορροπημένο δένδρο για κάθε τιμή του n.

Συνθήκες ισορροπίας για BST

- Ποια είναι τα χαρακτηριστικά μίας καλής συνθήκης ισορροπίας;
- Μία καλή συνθήκη ισορροπίας εξασφαλίζει ότι το ύψος ενός δένδρου με n κόμβους είναι της τάξης O(log n).
- Μια καλή συνθήκη ισορροπίας πρέπει να μπορεί να διατηρηθεί αποδοτικά.
- Δηλαδή η πρόσθετη δουλειά η οποία είναι απαραίτητη προκειμένου να ισορροπήσει το δένδρο, όταν ένα στοιχείο εισάγεται ή διαγράφεται, πρέπει να είναι της τάξης O(log n).

Δένδρα AVL

Γιώργος Στάμου

Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

AVL δένδρα – Ορισμός

Ένα δυαδικό δένδρο αναζήτησης $T = \{r; T_L, T_R\}$ είναι *δένδρο AVL* (Adelson-Velsky and Landis) αν είναι κενό ή:

- $|h_L – h_R| \leq 1$, όπου h_L είναι το ύψος του T_L και h_R είναι το ύψος του T_R
- και τα T_L και T_R είναι δένδρα AVL

Δηλαδή, σε κάθε κόμβο, το ύψος των δυο παιδιών διαφέρει το πολύ κατά 1!

Δένδρα AVL – Ιδιότητες

- Προφανώς, όλα τα τέλεια δυαδικά δένδρα αναζήτησης είναι AVL.
- Αυτό που δεν είναι τόσο προφανές είναι ότι τα ύψη όλων των δένδρων που ικανοποιούν την AVL συνθήκη ισορροπίας είναι λογαριθμικά ως προς τον αριθμό των εσωτερικών κόμβων.
- Ισχύει ότι το ύψος h ενός AVL ισορροπημένου δένδρου με *n* εσωτερικούς κόμβους ικανοποιεί τη συνθήκη:

$$\log_2(n + 1) < h \leq 1{,}440 \log(n + 2) - 0{,}328$$

Εισαγωγή στοιχείων σε δένδρα AVL

- Η εισαγωγή ενός στοιχείου σε ένα AVL δένδρο είναι μία διαδικασία που ολοκληρώνεται σε δύο φάσεις.
  - Πρώτον, το αντικείμενο εισάγεται στο δένδρο χρησιμοποιώντας την συνηθισμένη μέθοδο εισαγωγής σε δυαδικά δένδρα αναζήτησης.
  - Αφού έχει εισαχθεί το αντικείμενο, είναι απαραίτητο να ελεγχθεί ότι το δένδρο που προκύπτει συνεχίζει να είναι AVL ισορροπημένο και να ισορροπήσουμε το δένδρο στην περίπτωση που δεν είναι.

Εισαγωγή στοιχείων σε δένδρα AVL

- Ακριβώς όπως και σε ένα συνηθισμένο δυαδικό δένδρο αναζήτησης, τα στοιχεία εισάγονται στα AVL δένδρα συνδέοντάς τα ως φύλλα.
- Για να βρούμε τη σωστή θέση εισαγωγής ακολουθούμε το μονοπάτι που διασχίζει η αναζήτηση για να προσδιορίσουμε το σημείο στο οποίο που θα πρέπει να εισαχθεί αυτό το στοιχείο.
- Αν το στοιχείο δεν υπάρχει ήδη στο δένδρο, η αναζήτηση είναι ανεπιτυχής και τερματίζει σε έναν εξωτερικό, κενό κόμβο, οπότε το στοιχείο που θέλουμε να εισάγουμε, τοποθετείται σε αυτόν τον εξωτερικό κόμβο.

Εισαγωγή στοιχείων σε δένδρα AVL

- Η εισαγωγή ενός στοιχείου σε έναν δοθέντα εξωτερικό κόμβο μπορεί να επηρεάσει τα ύψη όλων των κόμβων που βρίσκονται κατά μήκος του *μονοπατιού πρόσβασης*, δηλαδή του μονοπατιού από την ρίζα σε αυτόν τον κόμβο.
- Φυσικά, όταν ένα αντικείμενο εισάγεται σε ένα δένδρο, το ύψος κάποιου υποδένδρου μπορεί να αυξηθεί κατά 1.
- Επομένως, για να εξασφαλίσουμε ότι το δένδρο που προκύπτει είναι ακόμα AVL ισορροπημένο, τα ύψη όλων των κόμβων κατά μήκος του μονοπατιού πρόσβασης πρέπει να υπολογιστούν εκ νέου και η AVL συνθήκη ισορροπίας πρέπει να ελεγχθεί.

Διαγραφή στοιχείων από δένδρα AVL

- Ακολουθούμε το μονοπάτι αναζήτησης, προκειμένου να προσδιορίσουμε το στοιχείο (εάν υπάρχει στο δένδρο).
- Αν το στοιχείο υπάρχει στο δένδρο, η αναζήτηση θα είναι επιτυχής τερματίζοντας στον κόμβο που θα πρέπει να διαγραφεί.
  - Αν ο κόμβος είναι φύλλο τον διαγράφουμε, μετατρέποντάς τον σε κενό υποδένδρο.
  - Αν ο κόμβος αυτός έχει ένα παιδί τον διαγράφουμε, αντικαθιστώντας τον με το μη κενό παιδί του.
  - Αν έχει δύο παιδιά, αντικαθιστούμε τη ρίζα του με τη ρίζα του προηγούμενου κόμβου στην ενδοδιατεταγμένη διάσχιση και διαγράφουμε εκείνο τον κόμβο (ο οποίος έχει ένα παιδί ή είναι φύλλο)

Διαγραφή στοιχείων από δένδρα AVL

- Η διαγραφή ενός στοιχείου από ένα AVL δένδρο μπορεί να επηρεάσει τα ύψη όλων των κόμβων που βρίσκονται κατά μήκος του *μονοπατιού πρόσβασης*, δηλαδή του μονοπατιού από την ρίζα σε αυτόν τον κόμβο.
- Φυσικά, όταν ένα αντικείμενο διαγράφεται σε ένα δένδρο, το ύψος κάποιου υποδένδρου μπορεί να μειωθεί κατά 1.
- Επομένως, για να εξασφαλίσουμε ότι το δένδρο που προκύπτει είναι ακόμα AVL ισορροπημένο, τα ύψη όλων των κόμβων κατά μήκος του μονοπατιού πρόσβασης πρέπει να υπολογιστούν εκ νέου και η AVL συνθήκη ισορροπίας πρέπει να ελεγχθεί.

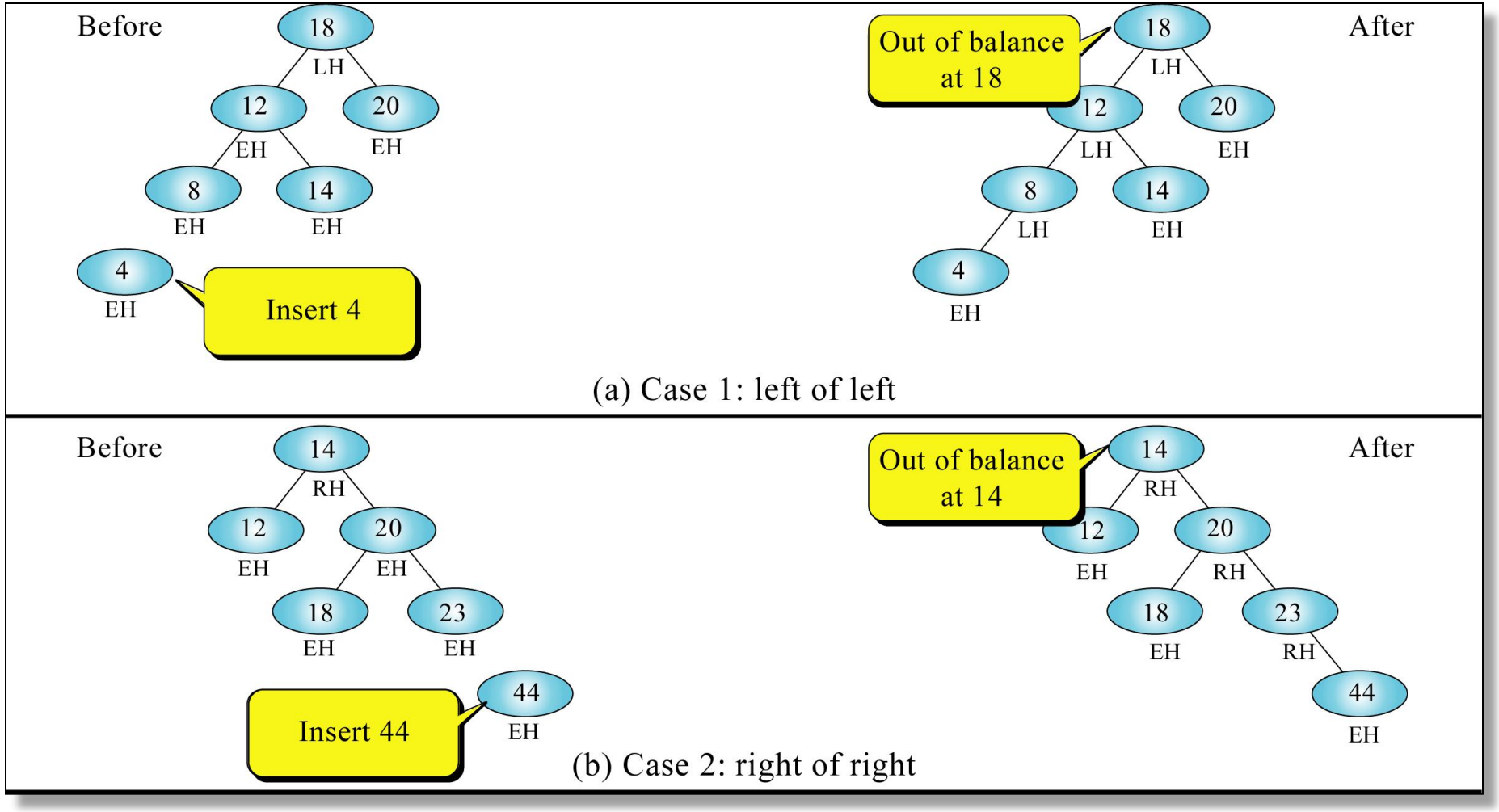
Εξισορρόπηση δένδρων AVL

- Ένα δένδρο AVL λέγεται «ψηλό από αριστερά» (LH) όταν το ύψος του αριστερού υποδένδρου είναι μεγαλύτερο από αυτό του δεξιού
- Αντίστοιχα για το «ψηλό από δεξιά» (RH)
- Αλλιώς είναι «ίσα ψηλό» (EH)
- Υπάρχουν αρκετοί τρόποι με τους οποίους η προσθήκη ή η διαγραφή στοιχείων σε ένα δένδρο AVL παραβιάζει τη συνθήκη χαρακτηρισμού του.
- Η επαναφορά της ιδιότητας σε ισχύ, γίνεται με ***περιστροφή*** του δένδρου, ανάλογα με την περίπτωση αναδιάταξης που συντρέχει.

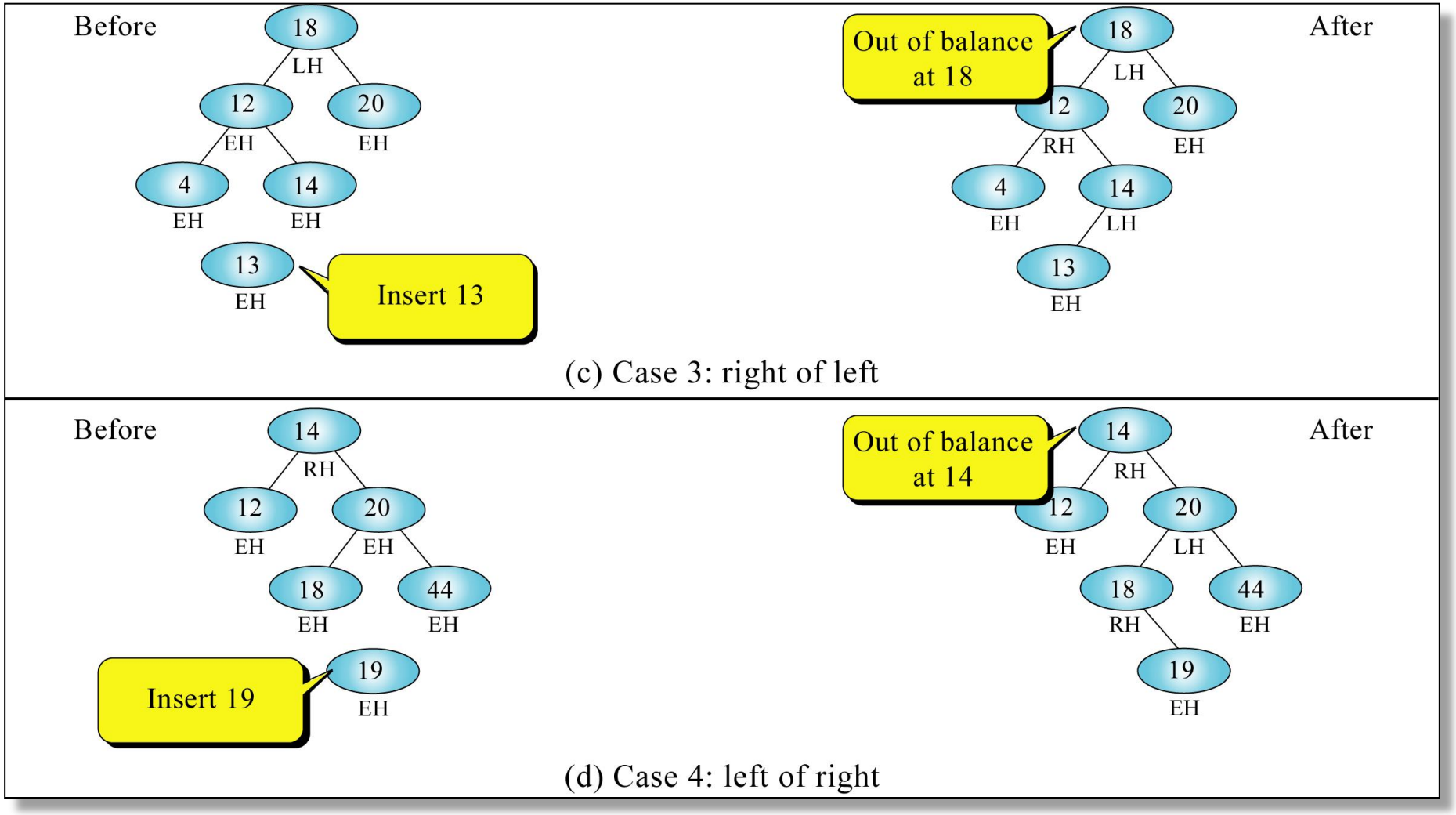
Αναδιάταξη δένδρων AVL

- Περιπτώσεις αναδιάταξης
  - ΑΑ: ένα αριστερό υποδένδρο ενός δένδρου AVL που είναι ψηλό από αριστερά, γίνεται επίσης ψηλό από αριστερά (**left of left**)
  - ΔΔ: τα αντίστοιχα για το δεξί υποδένδρο (**right of right**)
  - ΔΑ: ένα υποδένδρο ενός δένδρου AVL ψηλού από αριστερά, γίνεται ψηλό από δεξιά (**right of left**)
  - ΑΔ: ένα υποδένδρο ενός δένδρου AVL ψηλού από δεξιά, γίνεται ψηλό από αριστερά (**left of right**)

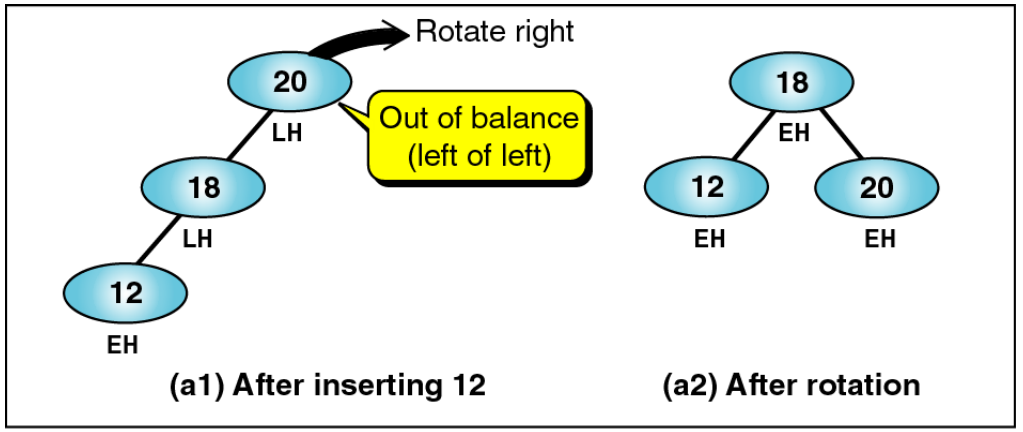
Περιπτώσεις αναδιάταξης (α)



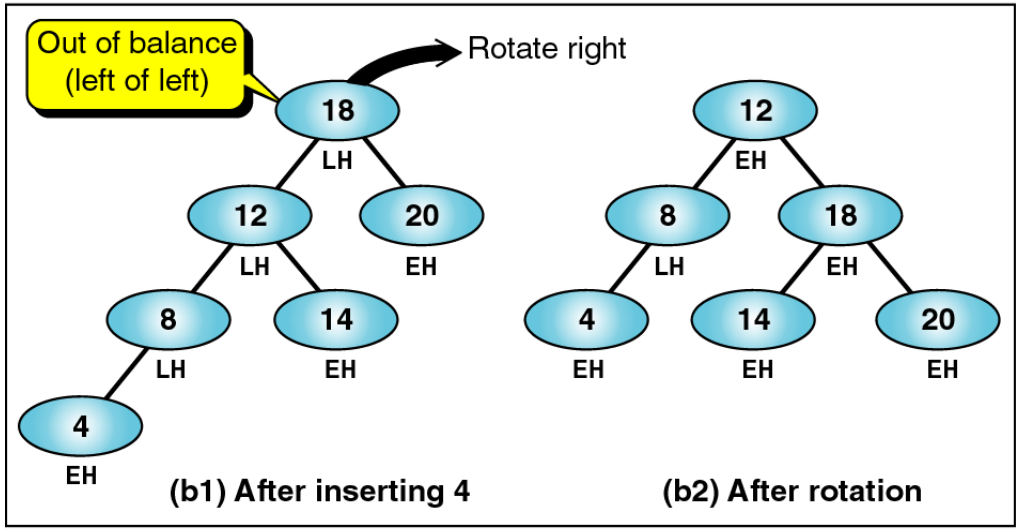
Περιπτώσεις αναδιάταξης (β)



Αναδιάταξη σε περίπτωση AA

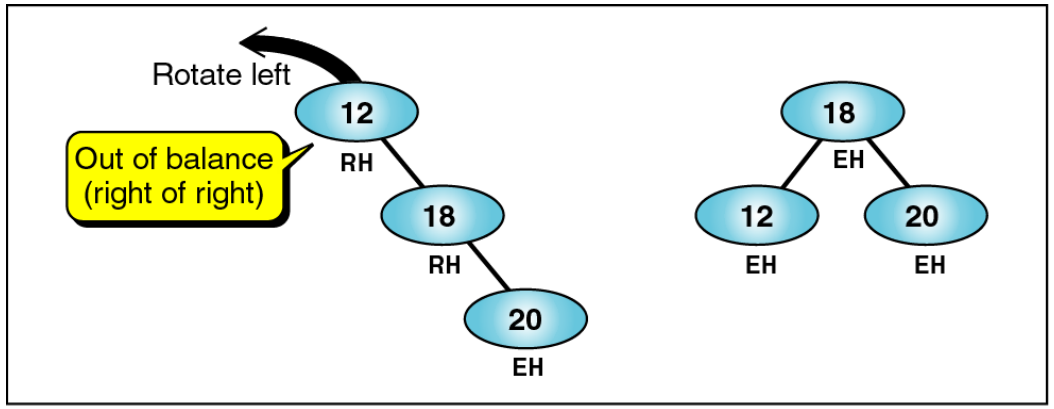


(a) Simple right rotation

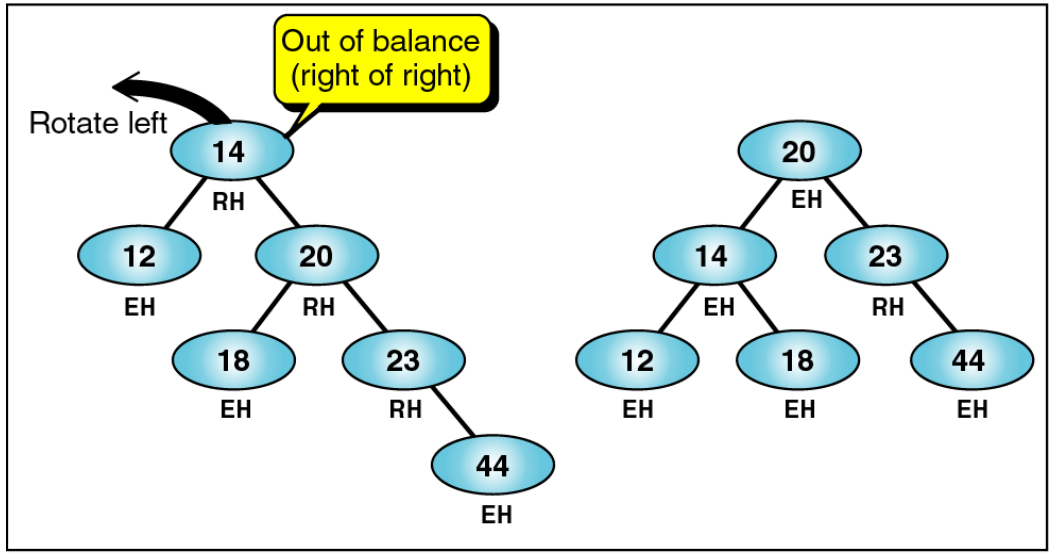


(b) Complex right rotation

Αναδιάταξη σε περίπτωση ΔΔ

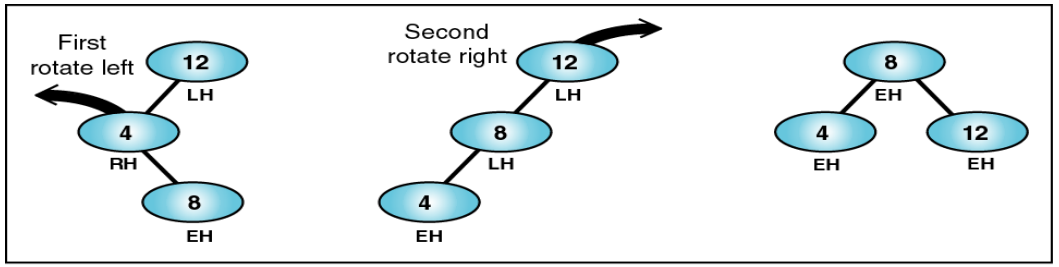


(a) Simple left rotation

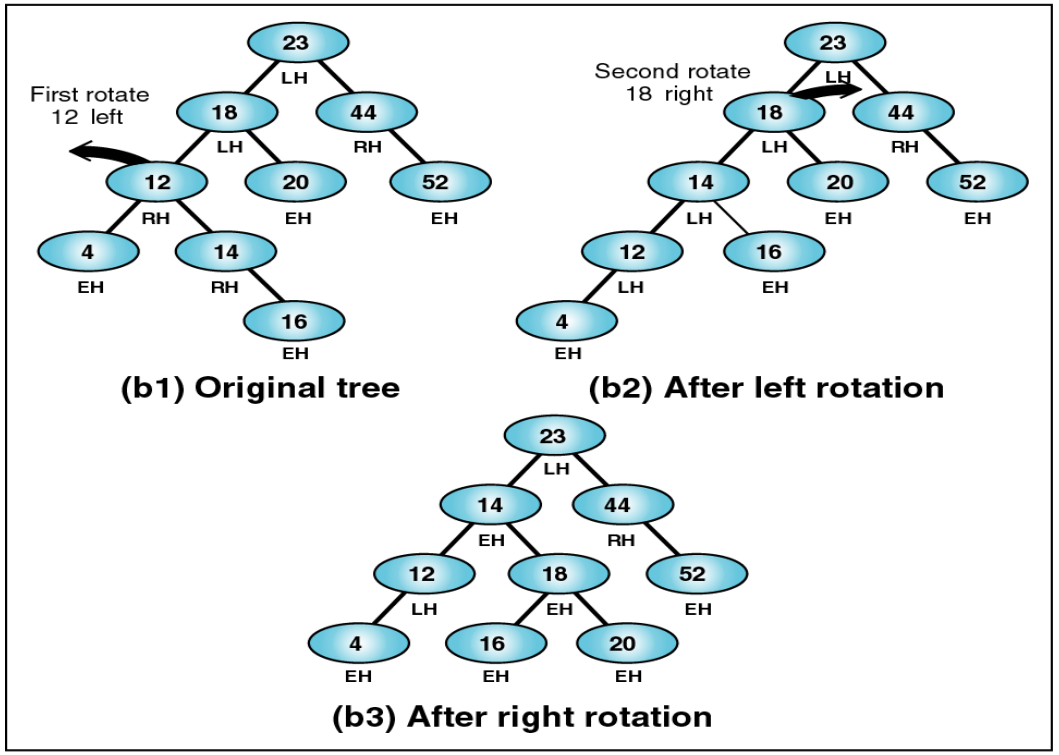


(b) Complex left rotation

Αναδιάταξη σε περίπτωση ΔΑ



(a) Simple double rotation right



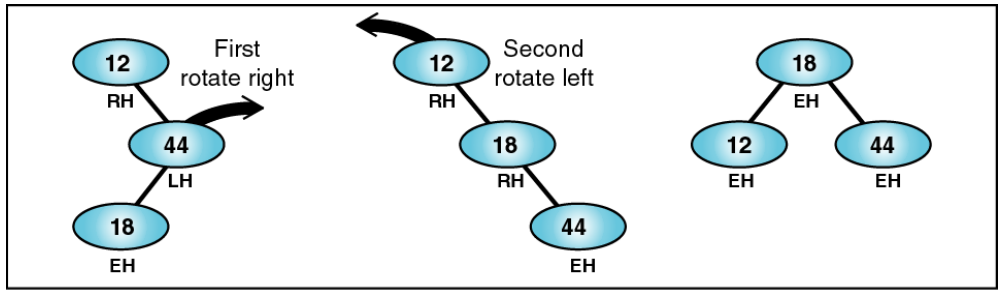
(b1) Original tree

(b2) After left rotation

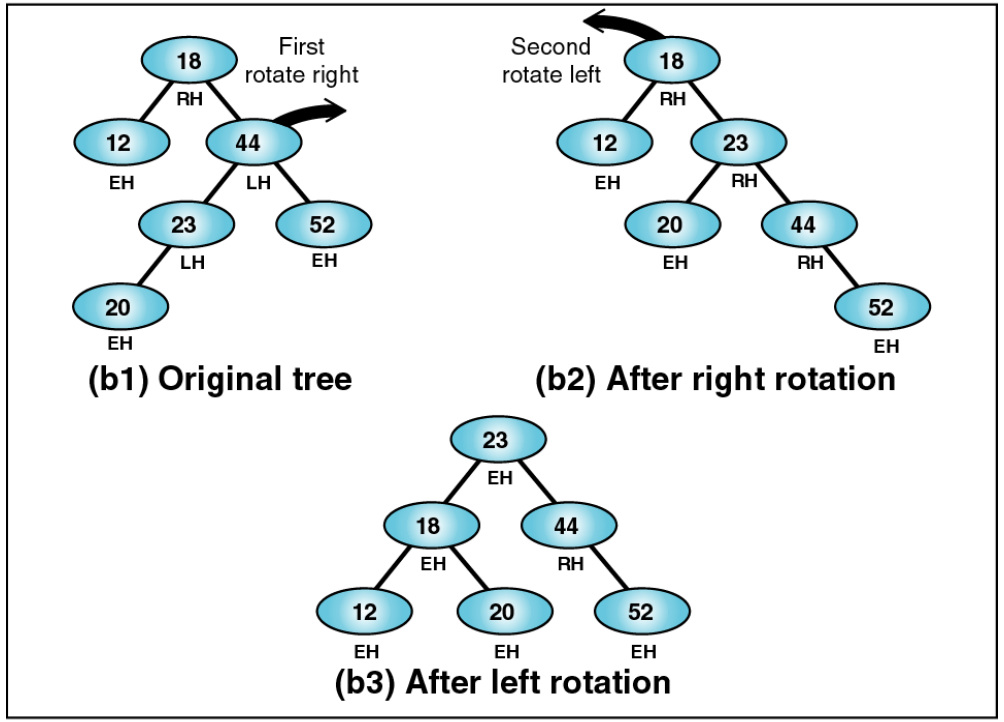
(b3) After right rotation

(b) Complex double rotation right

Αναδιάταξη σε περίπτωση ΑΔ



(a) Simple double rotation right



(b1) Original tree

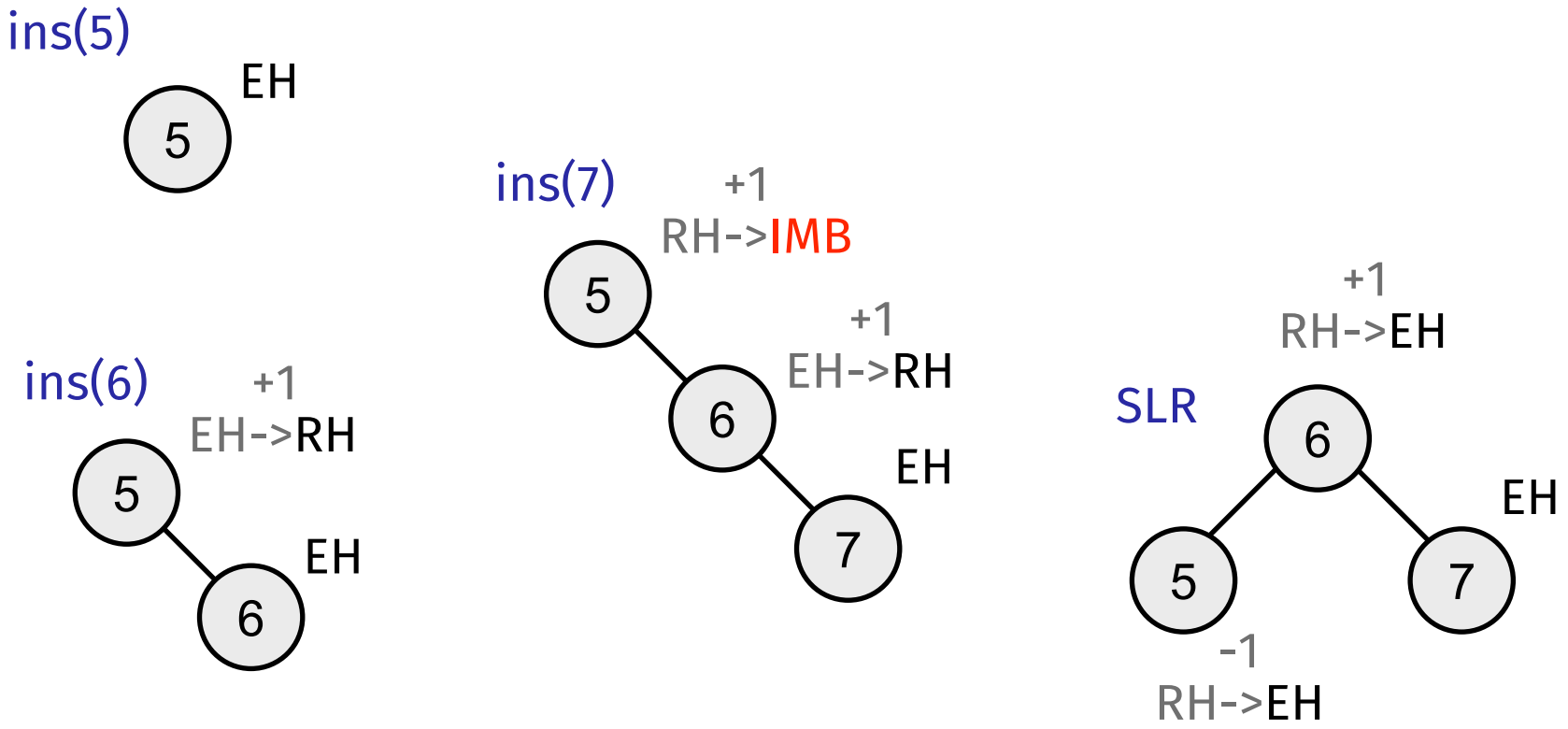
(b2) After right rotation

(b3) After left rotation

(b) Complex double rotation right

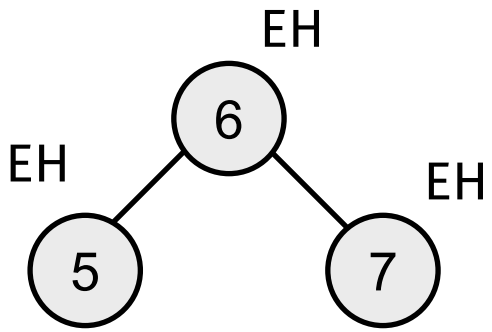
Παράδειγμα

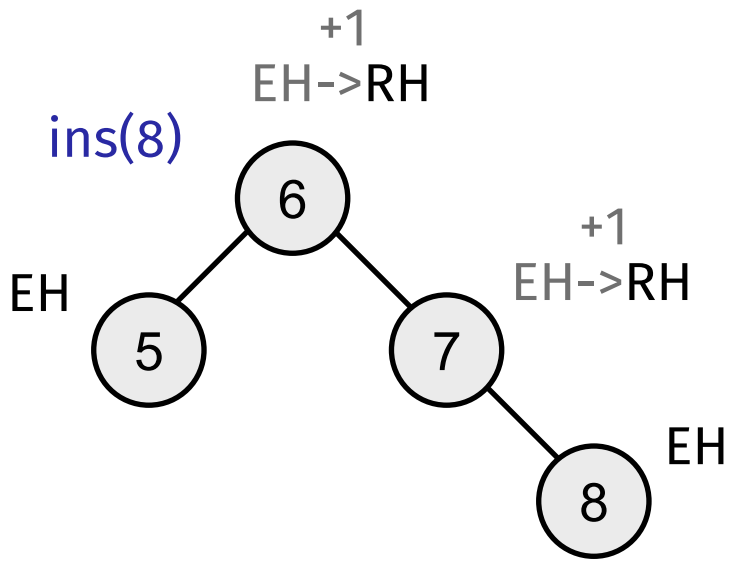
ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)



Παράδειγμα

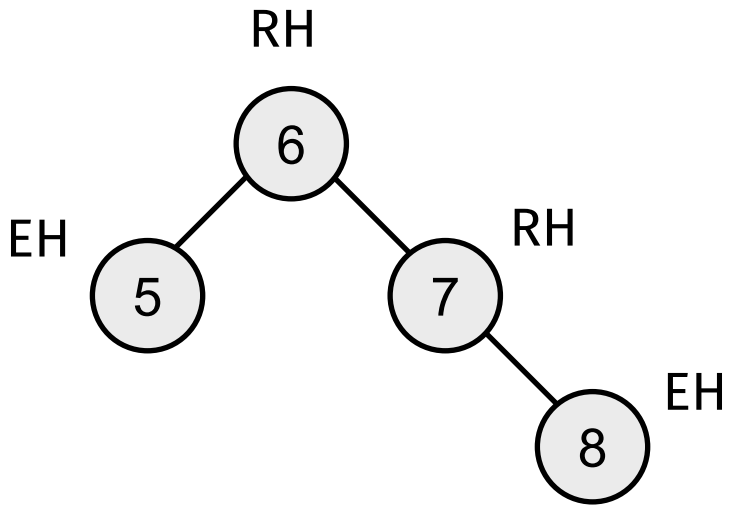
ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)

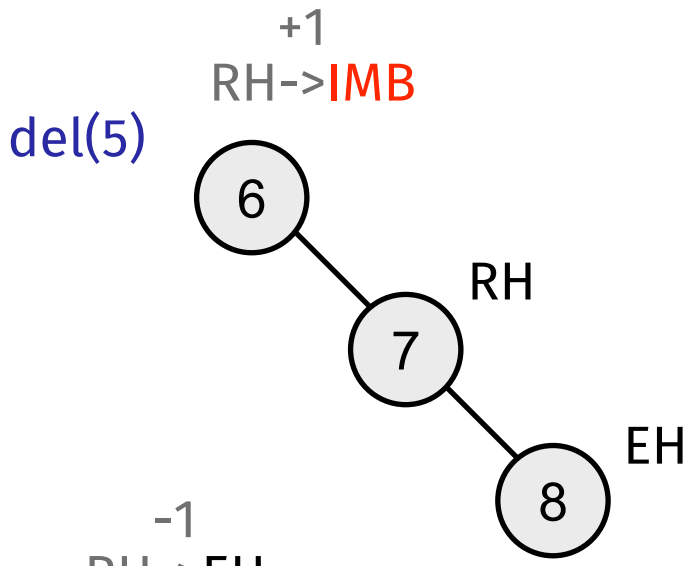


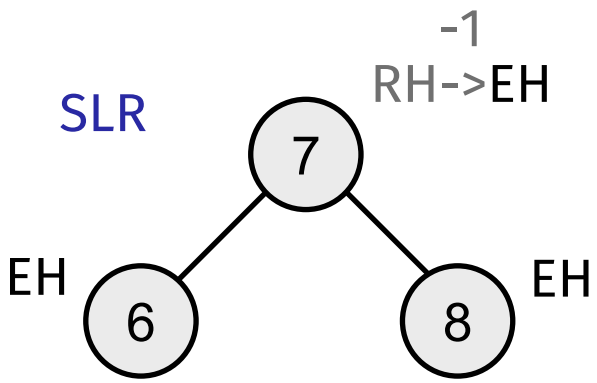


Παράδειγμα

ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)

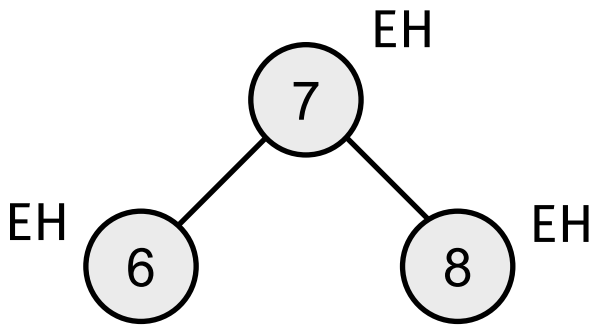


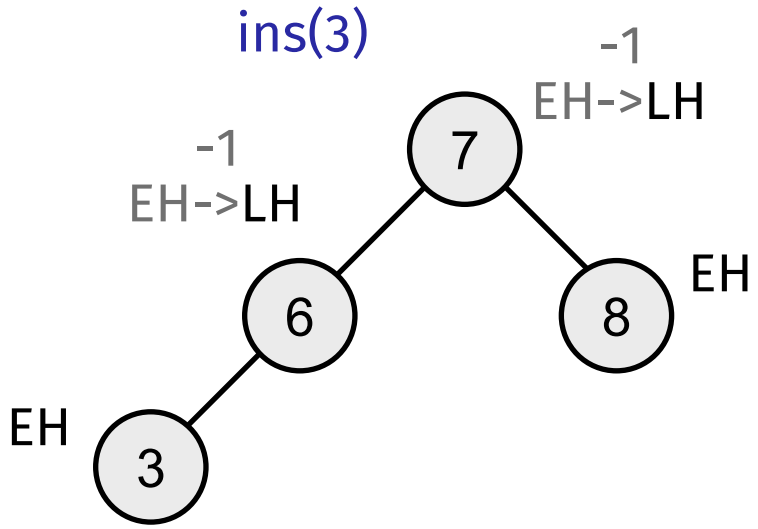




Παράδειγμα

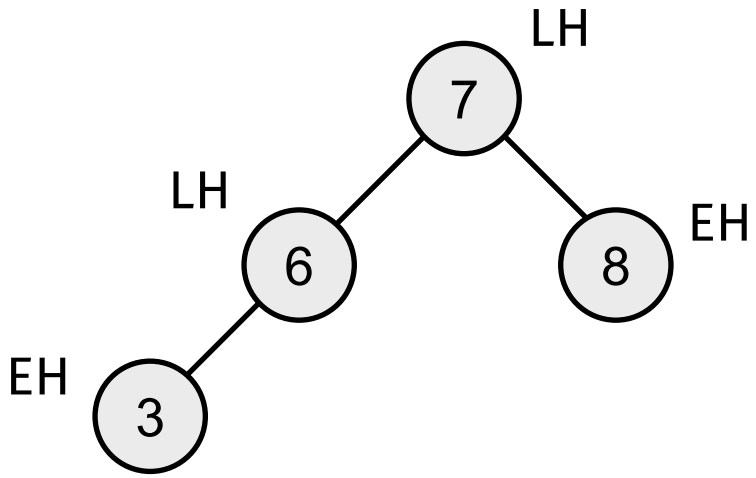
ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)

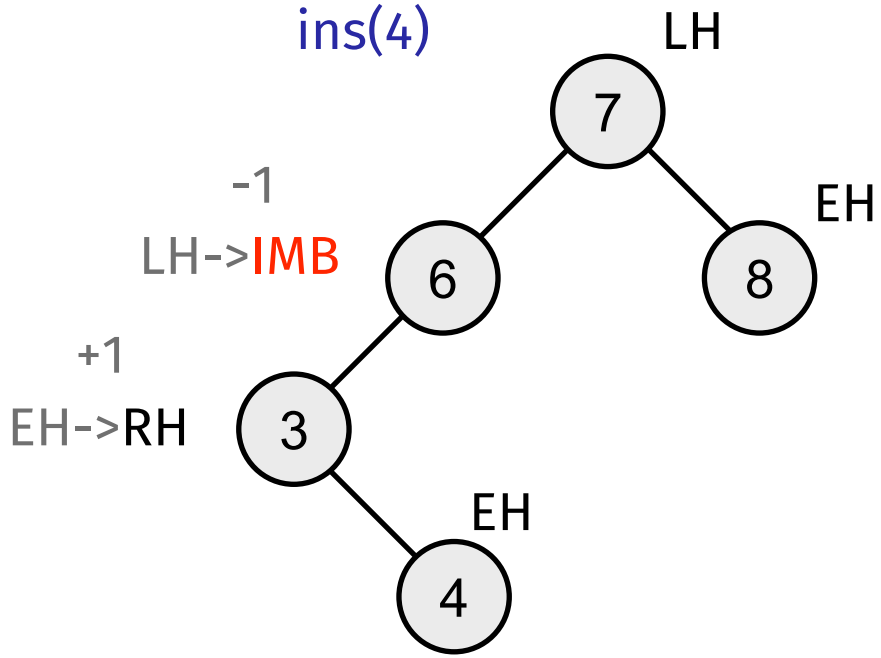




Παράδειγμα

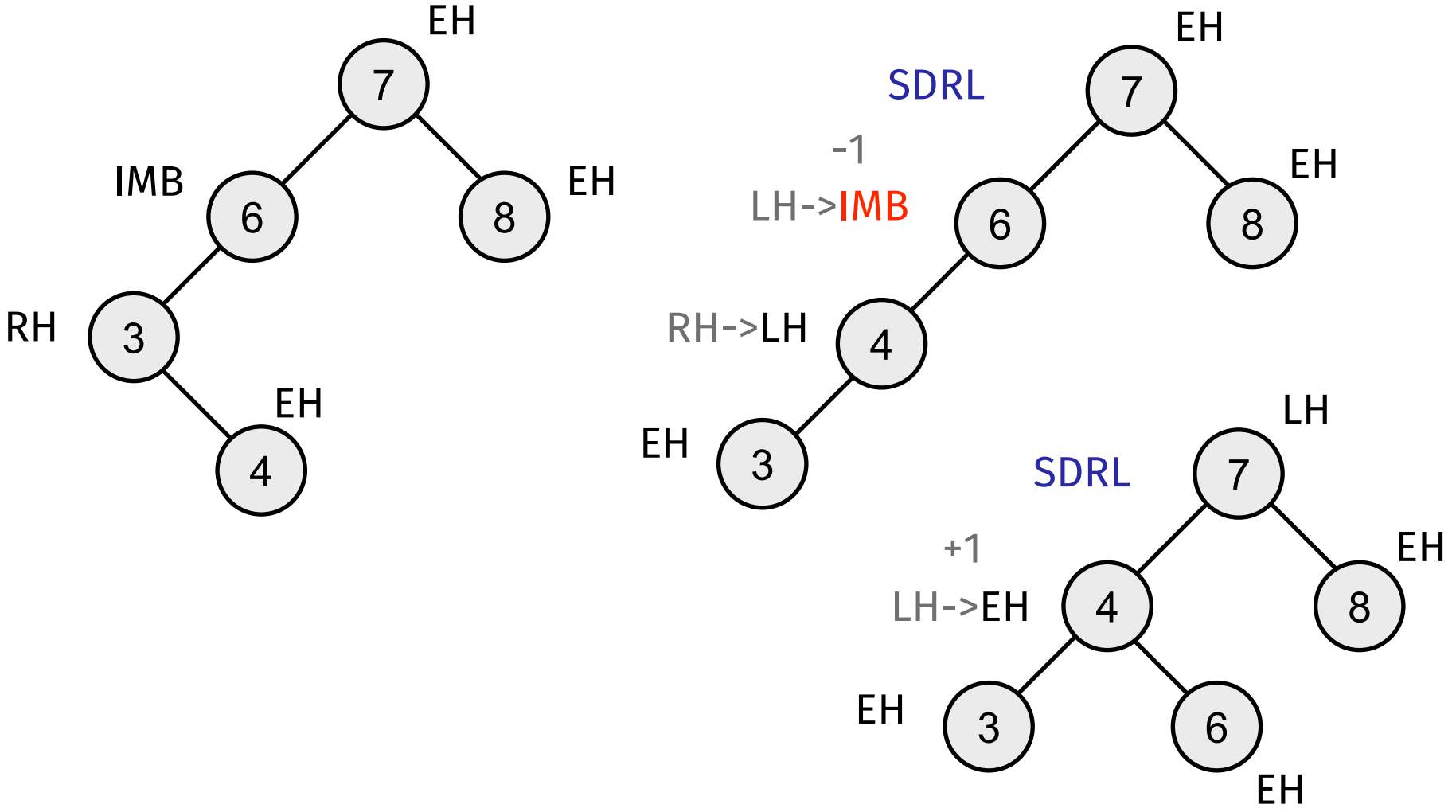
ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)





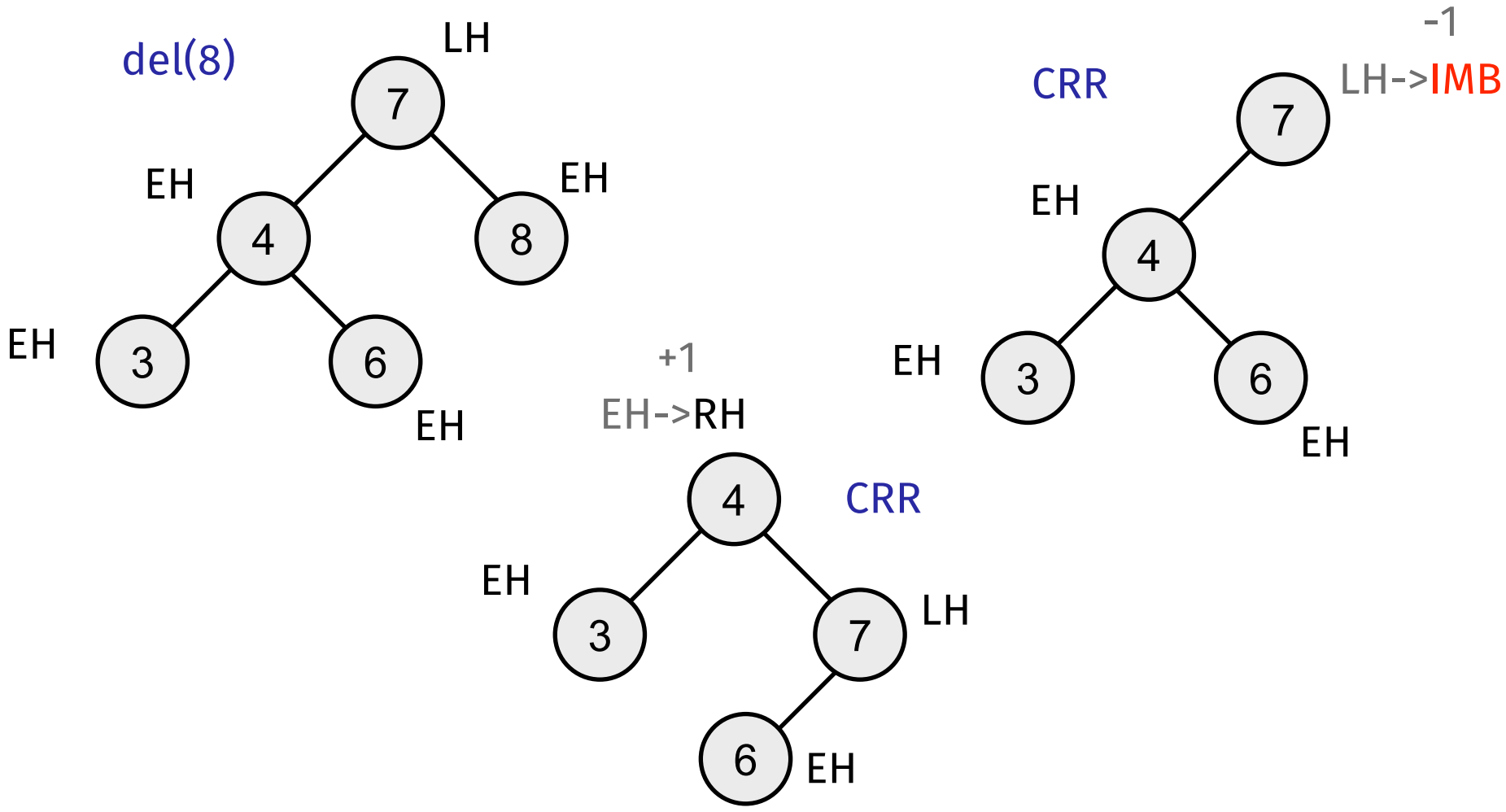
Παράδειγμα

ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)



Παράδειγμα

ins(5), ins(6), ins(7), ins(8), del(5), ins(3), ins(4), del(8)



Υλοποίηση δένδρων AVL

```
struct Node {
   int key;
   Node *left, *right;
   int height;
};

int height(Node *n) {    // απλός getter, αν το πεδίο height είναι ενημερωμένο σωστά
   if (n == nullptr) return 0;
   return n->height;
}

// Δημιουργεί έναν νέο κόμβο, χωρίς να τον τοποθετήσει στο δένδρο
 Node* newNode(int newkey) {
   Node* newNode = new Node;
   newNode->key = newkey;
   newNode->left = newNode->right = nullptr;
   newNode->height = 1;
   return newNode;
}
```

Υλοποίηση δένδρων AVL

```
Node *rightRotate(Node *y) {            // Δεξιά περιστροφή
   Node *x = y->left;
   Node *tmpNode = x->right;
   x->right = y;  // περιστροφή: x είναι η νέα ρίζα
   y->left = tmpNode;
   y->height = max(height(y->left), height(y->right)) + 1;    // ενημέρωση υψών
   x->height = max(height(x->left), height(x->right)) + 1;
   return x;
}

Node *leftRotate(Node *x) {             // Αριστερή περιστροφή
   Node *y = x->right;
   Node *tmpNode = y->left;
   y->left = x;
   x->right = tmpNode;
   x->height = max(height(x->left), height(x->right)) + 1;  // ενημέρωση υψών
   y->height = max(height(y->left), height(y->right)) + 1;
   return y;
}
```

Υλοποίηση δένδρων AVL

```
int getBalance(Node *n) {
    if (n == nullptr) return 0;
    return height(n->left) - height(n->right);
}

Node* insertNode(Node* n, int key) {
    // 1. εισαγωγή στο BST χωρίς ζύγισμα
    if (n == nullptr)
        return newNode(key);
    if (key < n->key)
        n->left = insertNode(n->left, key);
    else if (key > n->key)
        n->right = insertNode(n->right, key);
    else // Δεν επιτρέπονται διπλά κλειδιά
        return n;

    return rebalanceNode(n);
}
```

Υλοποίηση δένδρων AVL

```
Node* rebalanceNode(Node* n) {
   // 2. ενημέρωση υψών
   n->height = 1 + max(height(n->left), height(n->right));

   // 3. έλεγχος ισορροπίας
   int balance = getBalance(n);

   if (balance > 1) {
      if (getBalance(n->left) >= 0) return rightRotate(n);      // Left Left
      n->left = leftRotate(n->left);                            // Left Right
      return rightRotate(n);
   } else if (balance < -1) {
      if (getBalance(n->right) <= 0) return leftRotate(n);      // Right Right
      n->right = rightRotate(n->right);                         // Right Left
      return leftRotate(n);
   }

   return n;
}
```

Υλοποίηση δένδρων AVL

```
Node* deleteNode(Node* n, int key) {
   // 1. διαγραφή από το BST χωρίς ζύγισμα
   if (n == nullptr) return n;
   if (key < n->key) n->left = deleteNode(n->left, key);
   else if (key > n->key) n->right = deleteNode(n->right, key);
   else {  // this is the node to delete
      Node *temp = n;
      if (n->left == nullptr && n->right == nullptr) { n = nullptr; delete temp; }
      else if (n->left == nullptr) { n = n->right; delete temp; }
      else if (n->right == nullptr) { n = n->left; delete temp; }
      else {
         temp = minValueNode(n->right);
         n->key = temp->key;
         n->right = deleteNode(n->right, temp->key);
      }
   }

   if (n == nullptr) return n;    // Αν το δένδρο άδειασε, τελειώσαμε.
   return rebalanceNode(n);
}
```

Δένδρα πολλαπλών οδεύσεων

Γιώργος Στάμου

Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Δένδρα αναζήτησης m-οδεύσεων

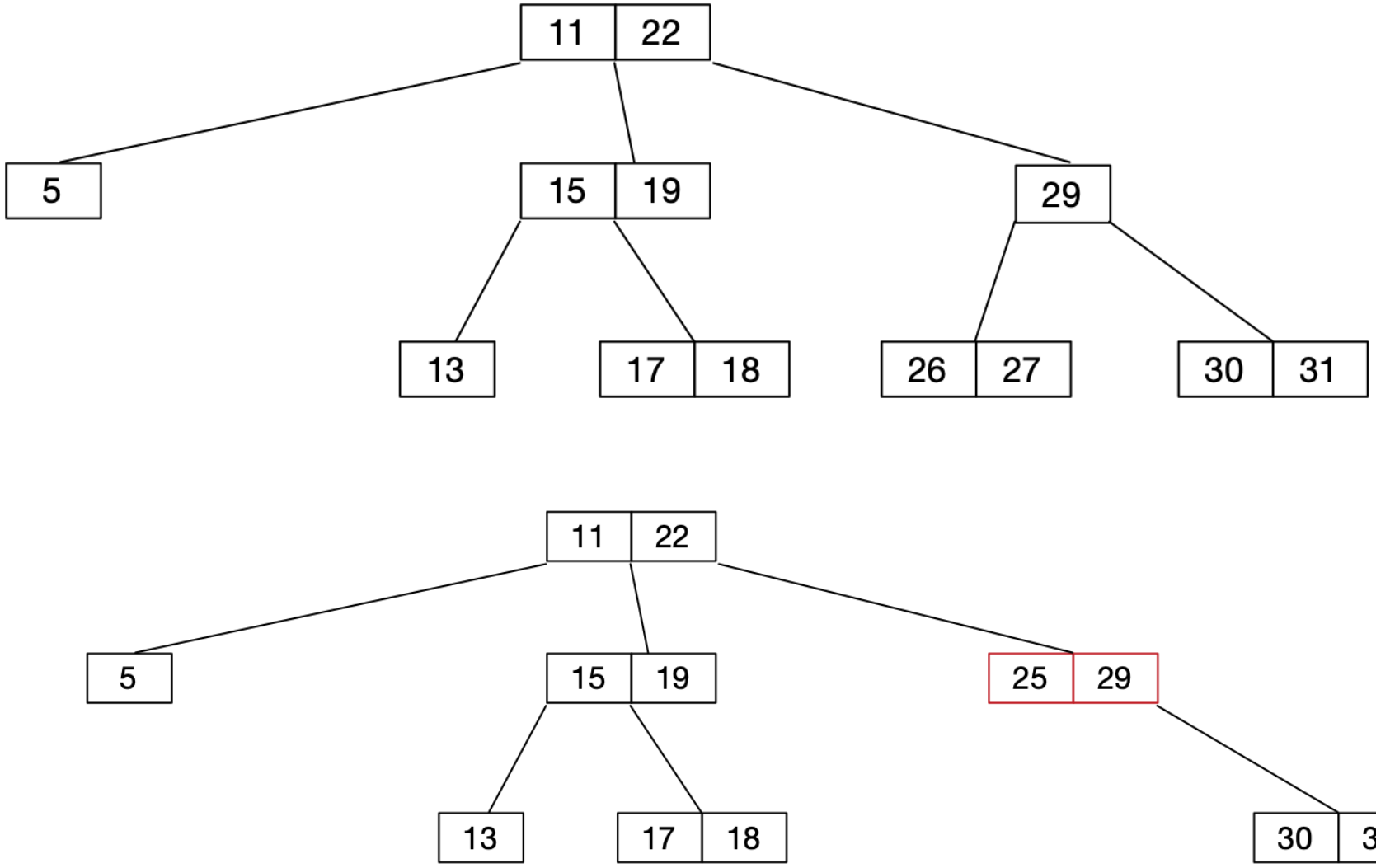
- Ένα *δένδρο αναζήτησης m-οδεύσεων, T,* είναι ένα δένδρο όπου σε κάθε κόμβο υπάρχει ένα πεπερασμένο σύνολο τιμών (κλειδιών).
- Το δένδρο είναι είτε κενό, $T = \varnothing$ ή αποτελείται από n υποδένδρα *m-οδεύσεων* $T_0, T_1, \ldots, T_{n-1}$, με $n - 1$ κλειδιά $k_1, k_2, \ldots, k_{n-1}$,
- Το δένδρο μπορεί να παρασταθεί σαν μία λίστα

$$T = [T_0, k_1, T_1, k_2, T_2, \ldots, k_{n-1}, T_{n-1}]$$

όπου $2 \le n \le M$, και τα κλειδιά και οι κόμβοι ικανοποιούν τις παρακάτω συνθήκες διάταξης:

  - Τα κλειδιά σε κάθε κόμβο είναι διακεκριμένα και ταξινομημένα, δηλαδή, $k_i < k_{i+1}$ για $1 \le i \le n - 2$.
  - Όλα τα κλειδιά που περιέχονται σε ένα υποδένδρο T_i έχουν τιμή μικρότερη από την τιμή του κλειδιού k_{i+1}, δηλαδή, $\forall k \in T_i : k < k_{i+1}$ για $0 \le i \le n - 2$. Το δένδρο T_i ονομάζεται το «αριστερό υποδένδρο» με αναφορά το κλειδί k_{i+1}.
  - Όλα τα κλειδιά που περιέχονται στο υποδένδρο T_i έχουν τιμές μεγαλύτερες από την τιμή του κλειδιού k_i, δηλαδή, $\forall k \in T_i : k > k_i$ για $1 \le i \le n - 1$. Το δένδρο T_{i+1} ονομάζεται το «δεξιό υποδένδρο» με αναφορά το κλειδί k_{i+1}.

Δένδρα αναζήτησης m-οδεύσεων



Αναζήτηση σε δένδρα *m*-οδεύσεων

- Η αναζήτηση ξεκινά από τη ρίζα.
- Εάν το δένδρο είναι κενό η αναζήτηση αποτυγχάνει.
- Αλλιώς, εξετάζουμε τα κλειδιά που περιέχονται στη ρίζα για να δούμε εάν τιμή που αναζητούμε βρίσκεται στο σύνολο των κλειδιών της ρίζας. Εάν βρούμε την τιμή τότε η αναζήτηση τελειώνει με επιτυχία.

Αναζήτηση σε δένδρα m-οδεύσεων

- Εάν δεν βρούμε την τιμή που αναζητούμε, τότε υπάρχουν τρεις περιπτώσεις (εδώ ψάχνουμε για την τιμή x):
  - η τιμή x, είναι μικρότερη από k_1, όπου σε αυτή τη περίπτωση ψάχνουμε στο υποδένδρο T_0
  - η τιμή x είναι μεγαλύτερη από k_{n-1}, όπου σε αυτή τη περίπτωση ψάχνουμε στο υποδένδρο T_{n-1}
  - υπάρχει δείκτης θέσης i τέτοιος ώστε $1 \leq i < n - 1$ για τον οποίο $k_i < x < k_{i+1}$, όπου σε αυτή τη περίπτωση ψάχνουμε στο υποδένδρο T_i

Πολυπλοκότητα αναζήτησης σε δένδρα *m*-οδεύσεων

- Η απλοϊκή μέθοδος χρησιμοποιεί απλή γραμμική αναζήτηση σε κάθε κόμβο
- Η χρονική πολυπλοκότητα για τη χειρότερη περίπτωση είναι της τάξης:

$$(M - 1)(h+1)T_{\texttt{<compare>}} + O(Mh)$$

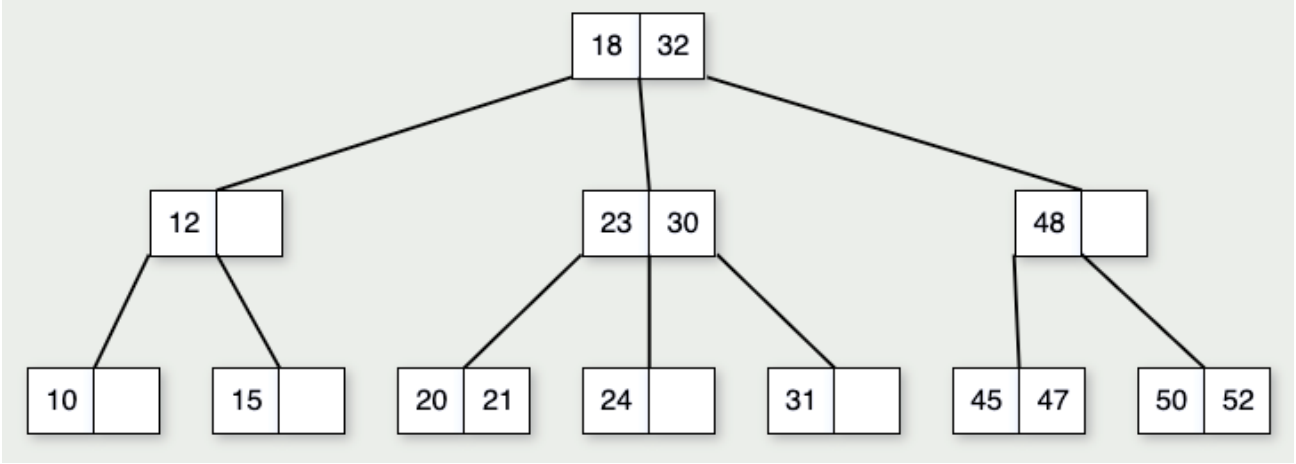
όπου h είναι το ύψος του δένδρου
και $T_{\texttt{<compare>}}$ είναι η πολυπλοκότητα της σύκρισης δύο κλειδιών

- Μια καλύτερη έκδοση χρησιμοποιεί δυαδική αναζήτηση
  Η χρονική πολυπλοκότητα για τη χειρότερη περίπτωση είναι της τάξης:

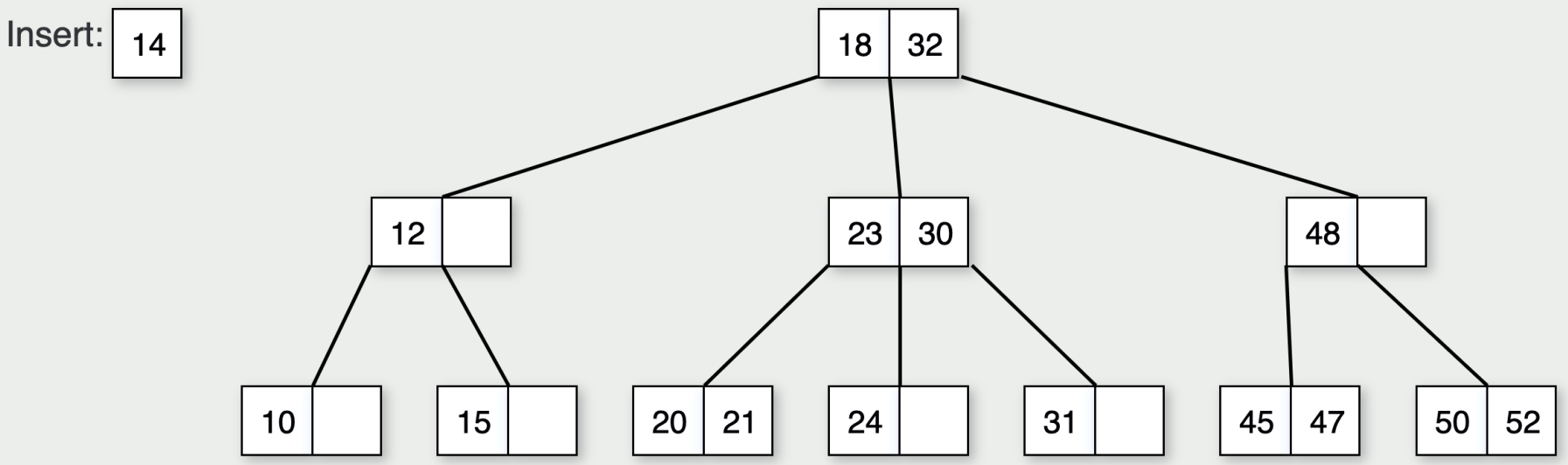
$$(h+1)(\lceil \log_2(M - 1) \rceil + 2)T_{\texttt{<compare>}} + O(h \log M)$$

2-3 trees

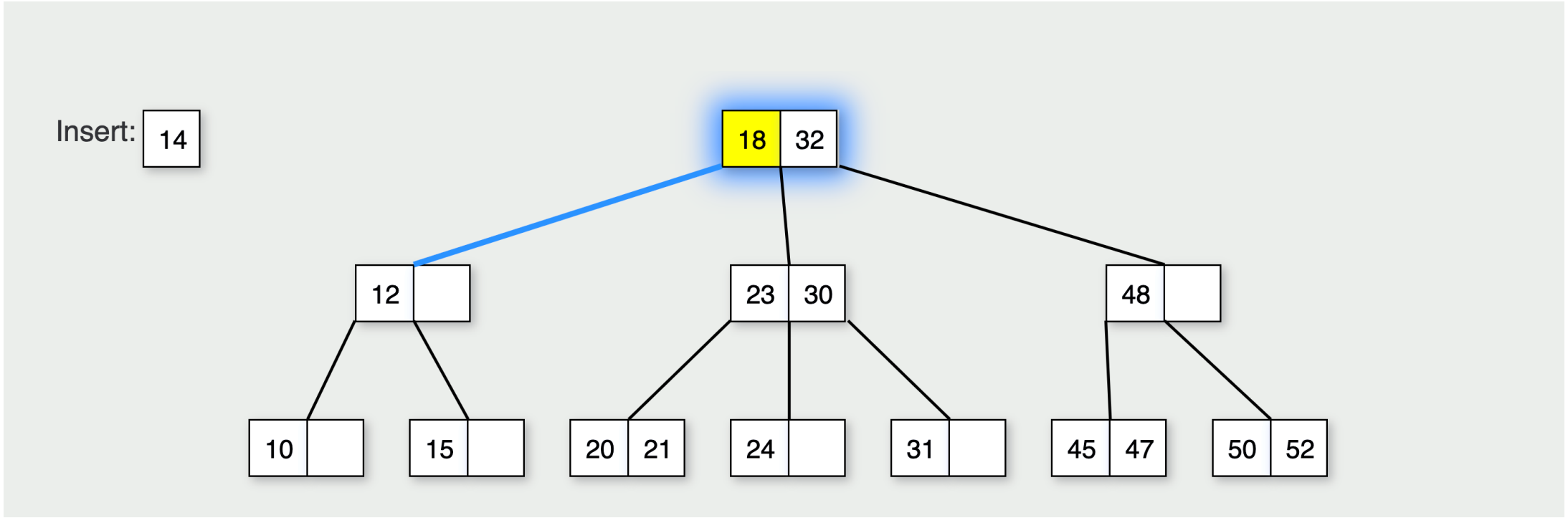
- Δένδρα m-οδεύσεων με τις ακόλουθες ιδιότητες:
  - Κάθε κόμβος περιέχει 1 ή 2 τιμές κλειδιών
  - Κάθε εσωτερικός κόμβος έχει είτε 2 παιδιά (1 κλειδί), είτε 3 παιδιά (2 κλειδιά)
  - Όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο



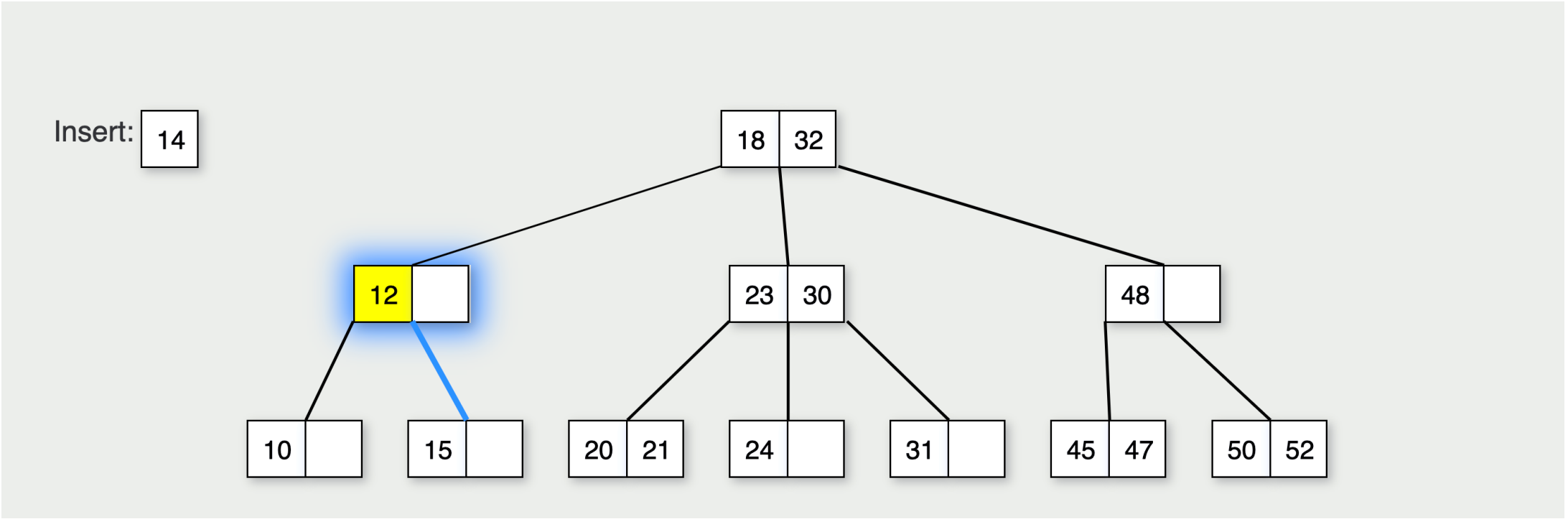
2-3 trees - εισαγωγή



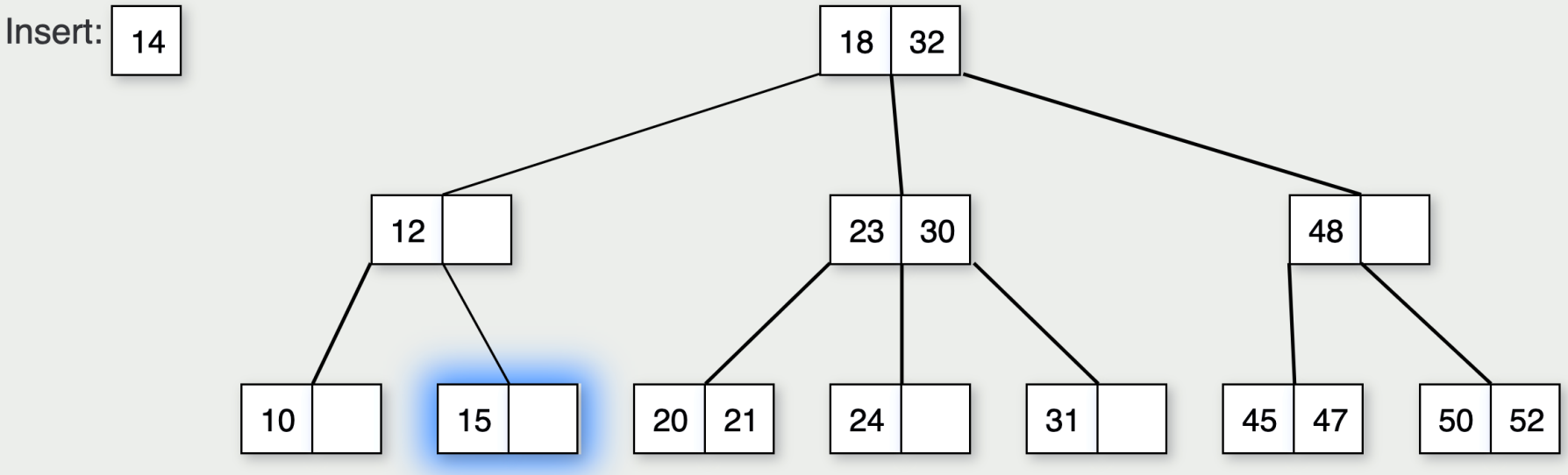
2-3 trees - εισαγωγή



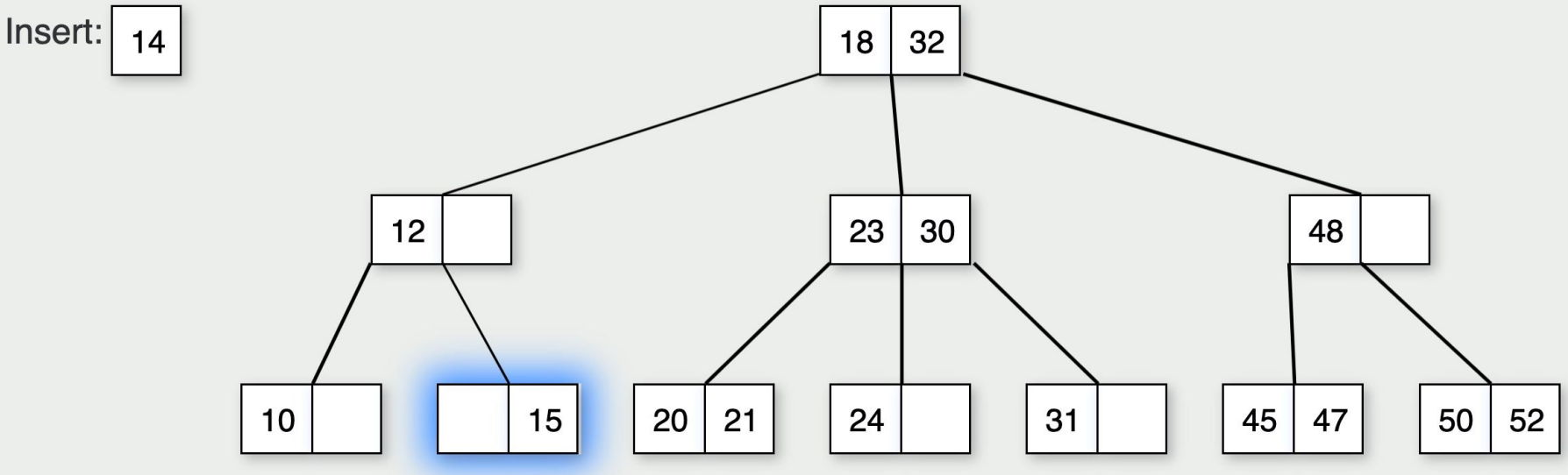
2-3 trees - εισαγωγή



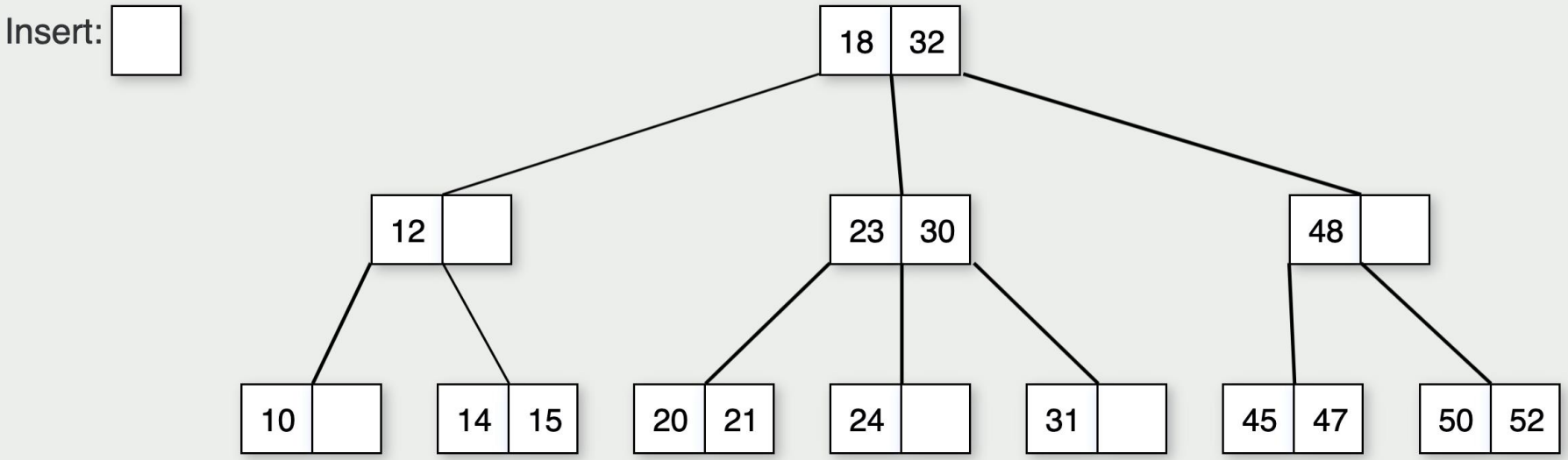
2-3 trees - εισαγωγή



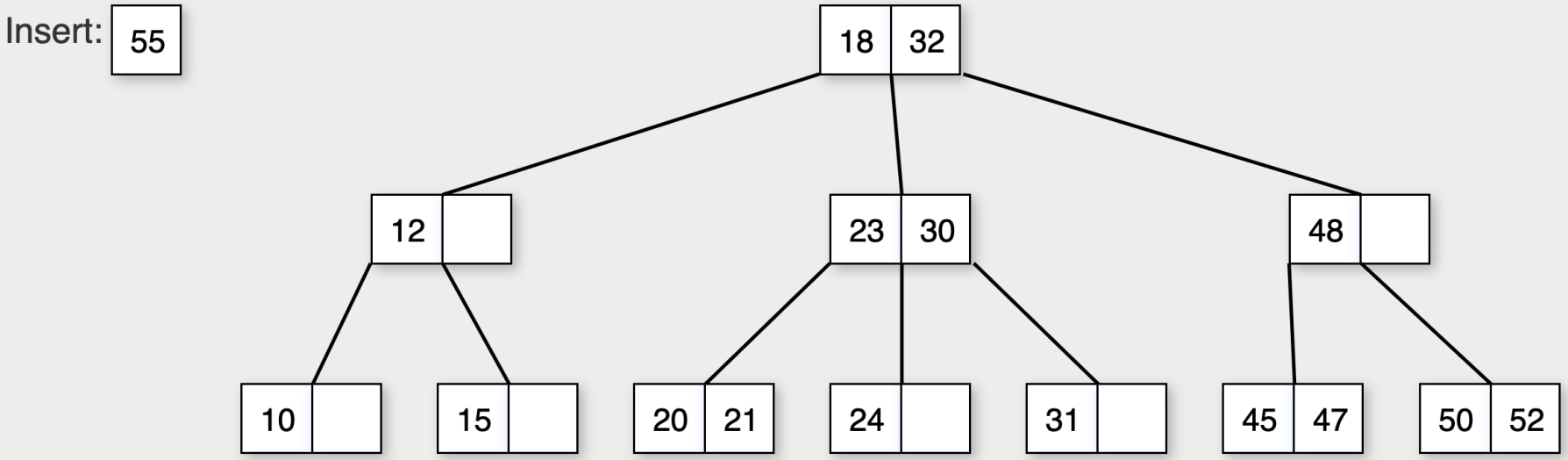
2-3 trees - εισαγωγή



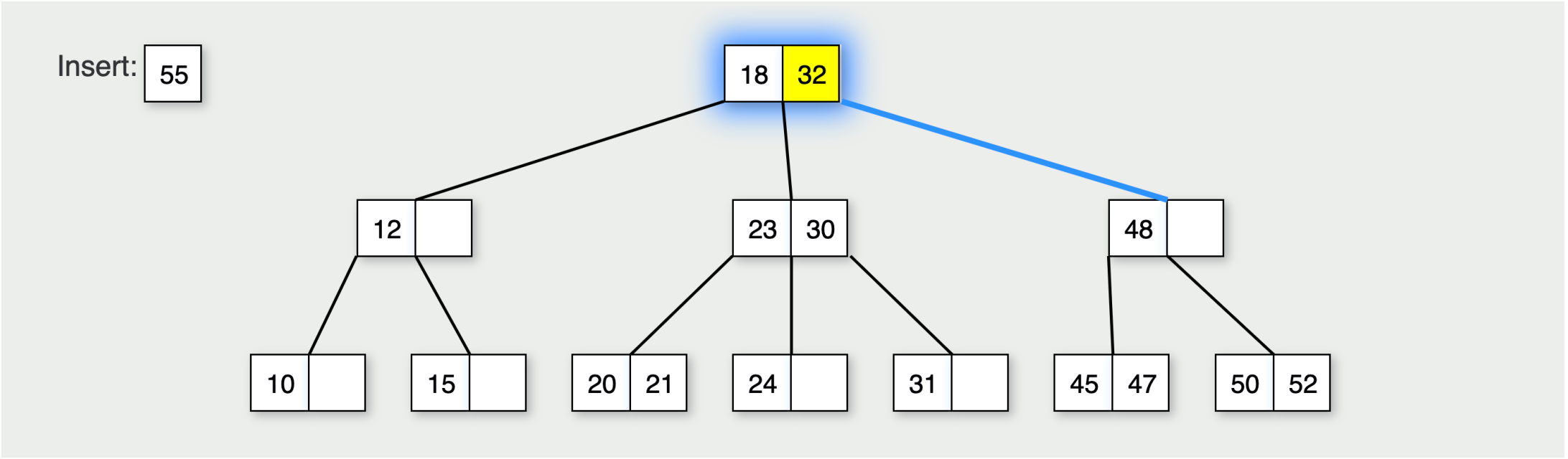
2-3 trees - εισαγωγή



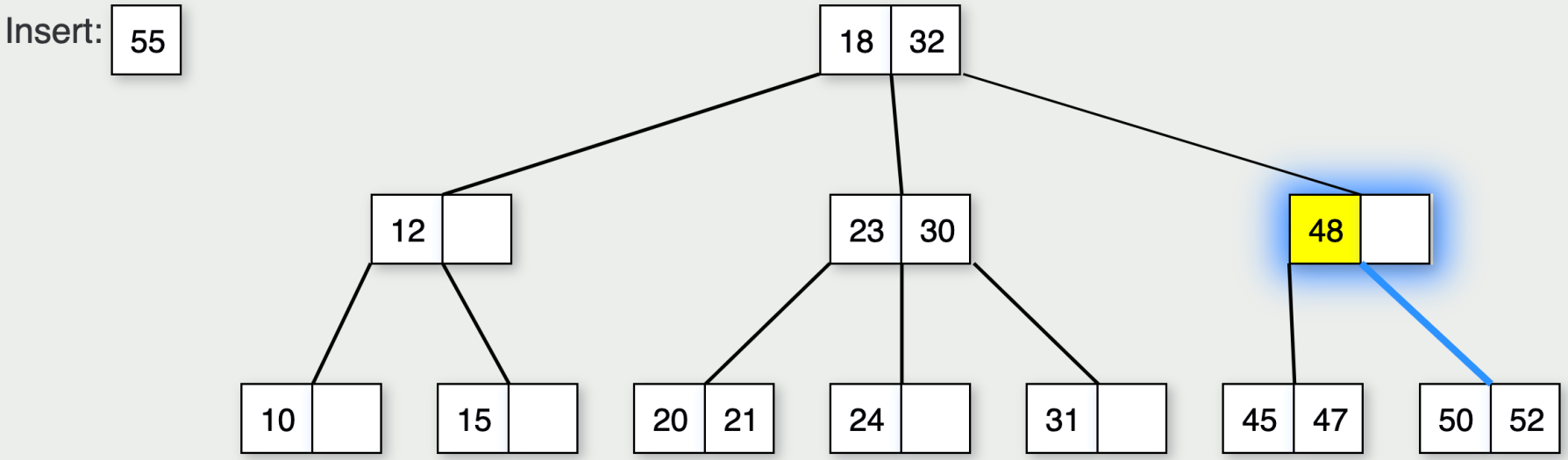
2-3 trees – εισαγωγή με προαγωγή κόμβου



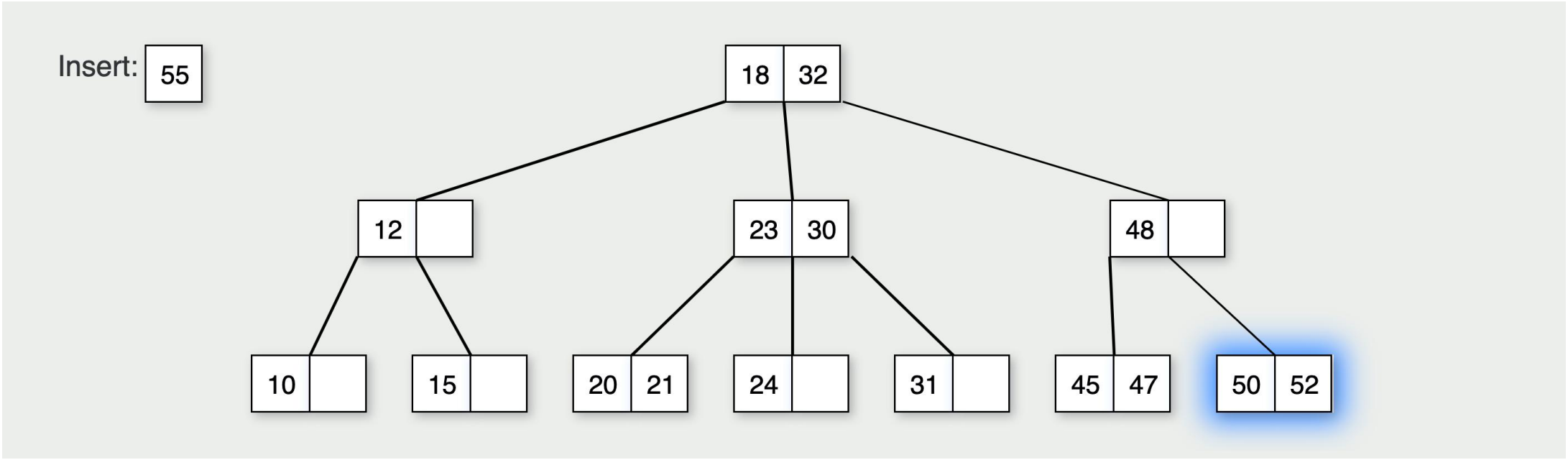
2-3 trees – εισαγωγή με προαγωγή κόμβου



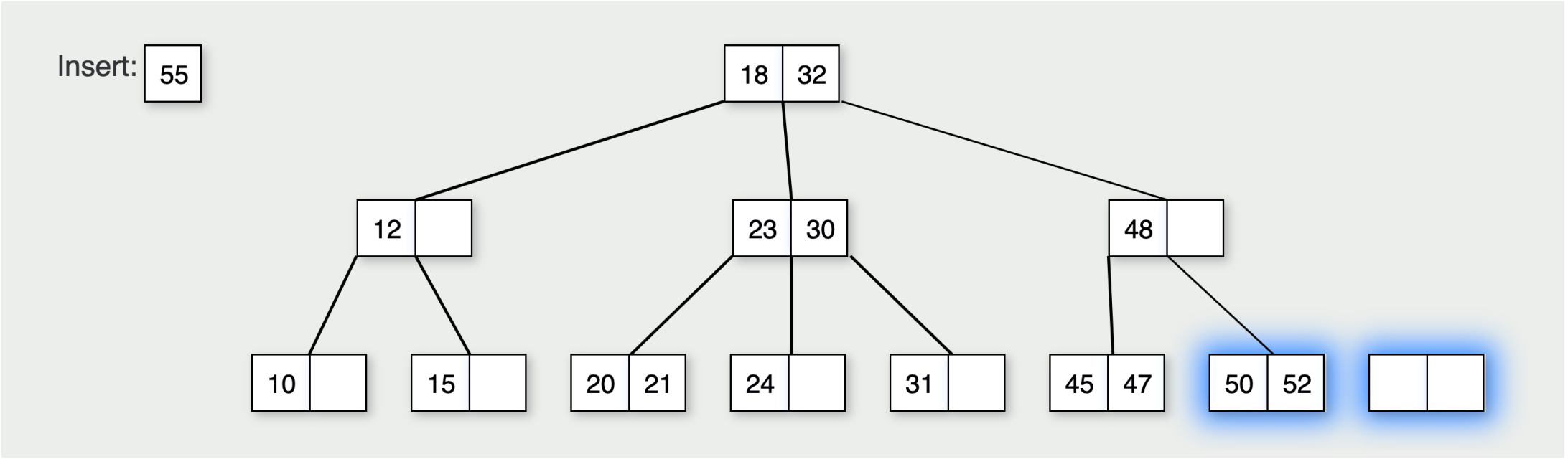
2-3 trees – εισαγωγή με προαγωγή κόμβου



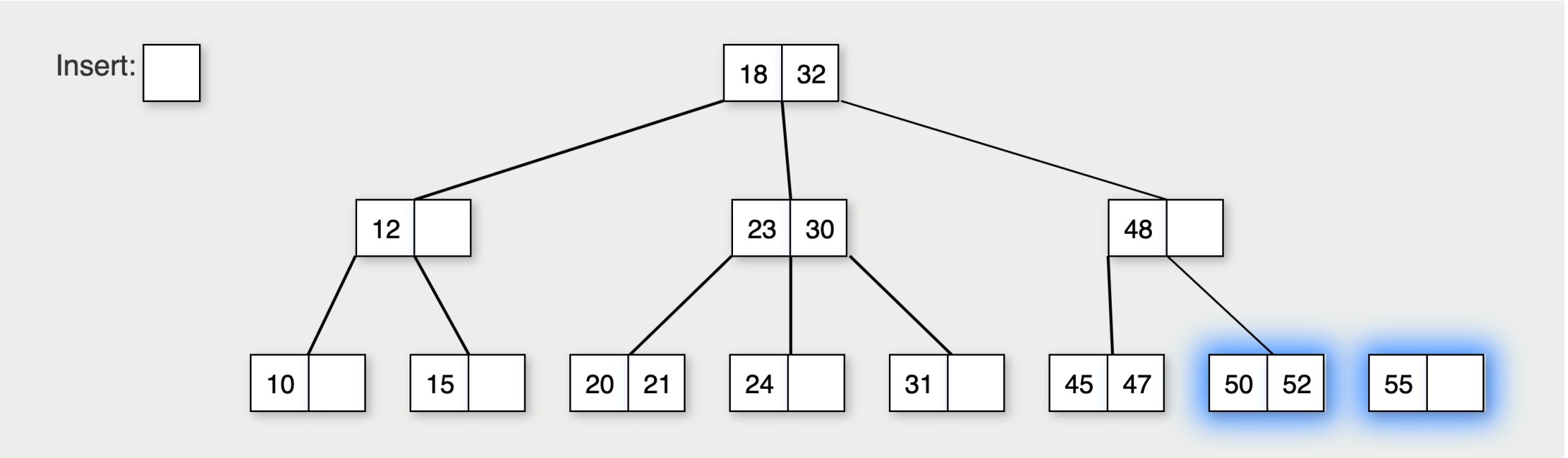
2-3 trees – εισαγωγή με προαγωγή κόμβου



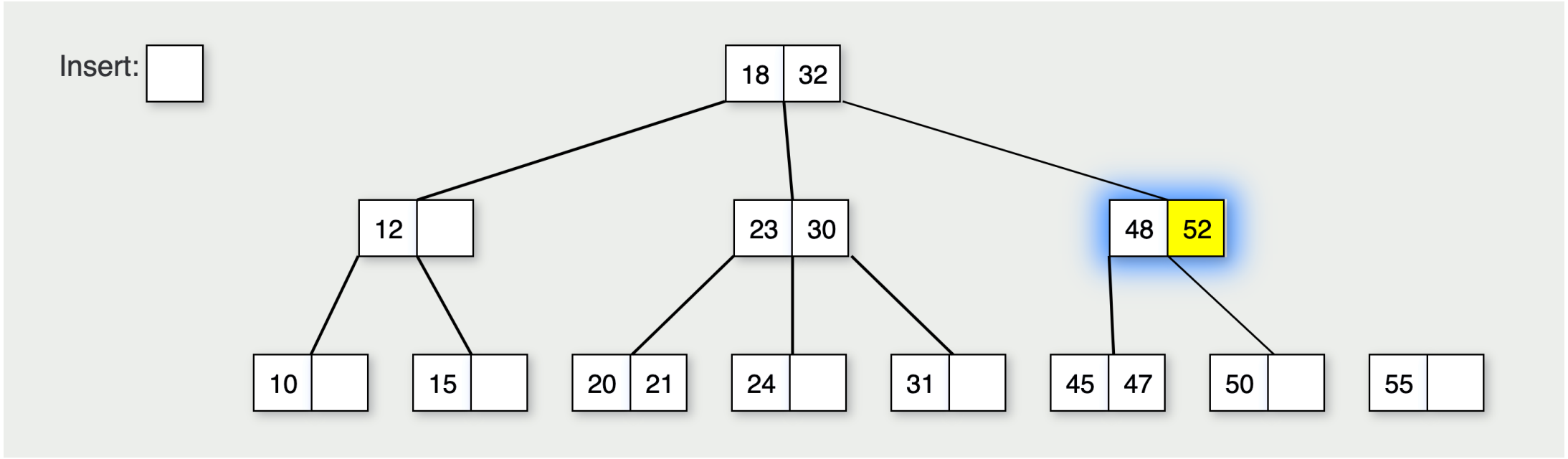
2-3 trees – εισαγωγή με προαγωγή κόμβου



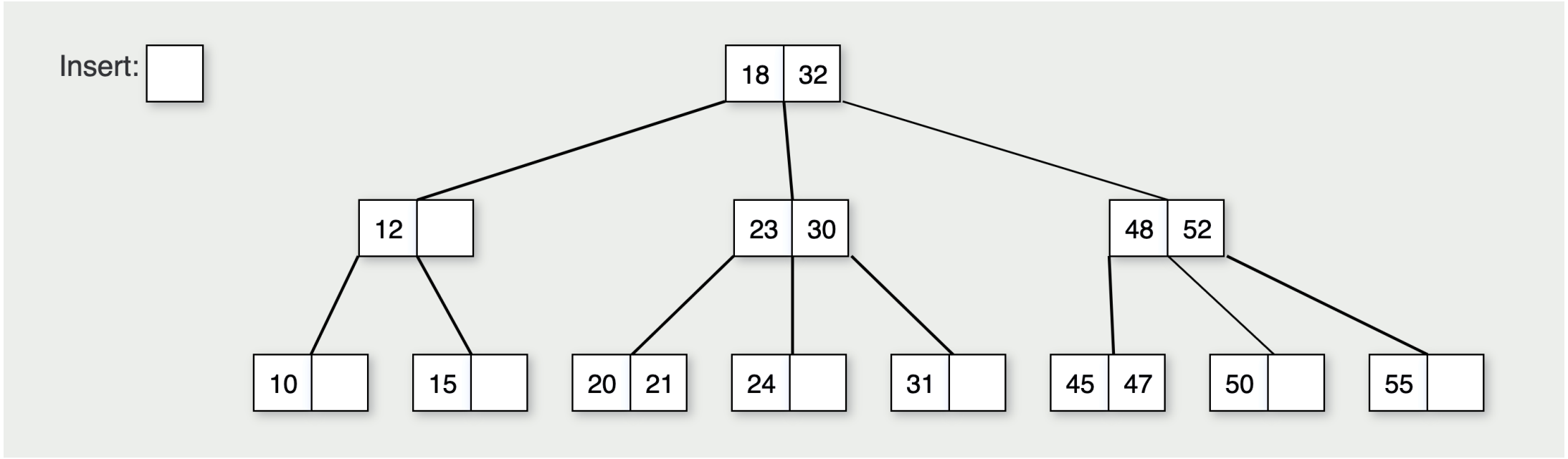
2-3 trees – εισαγωγή με προαγωγή κόμβου



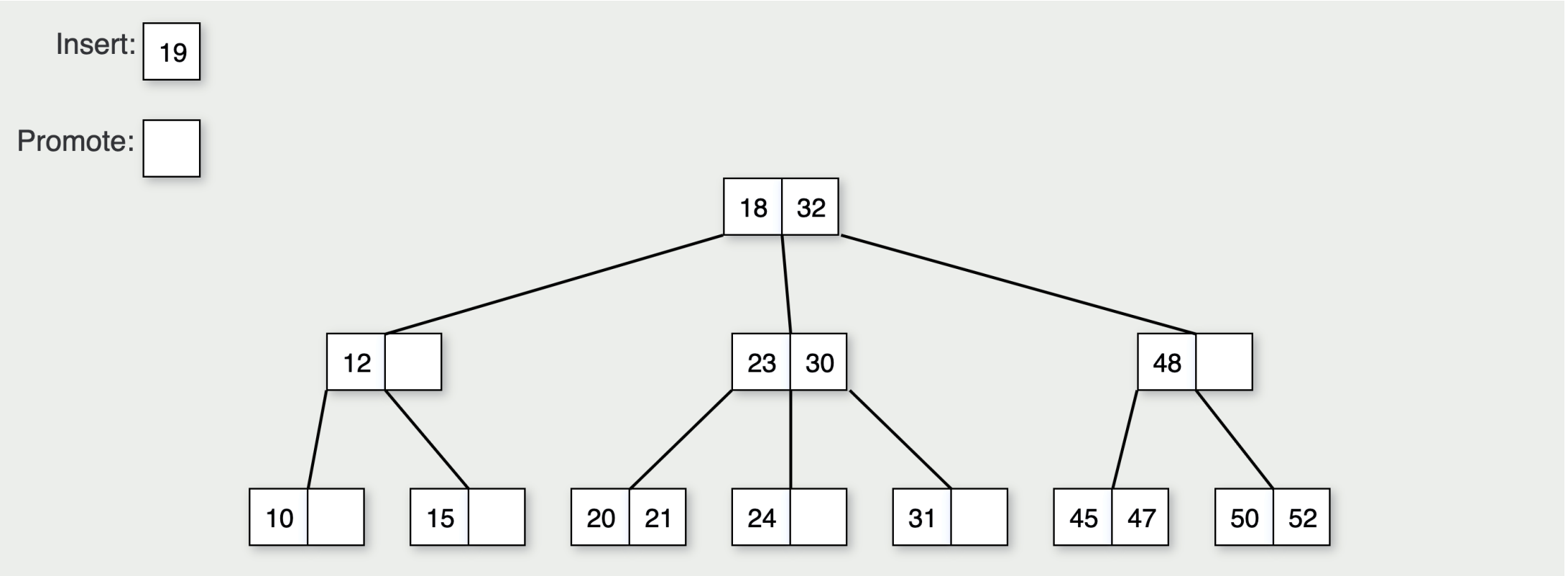
2-3 trees – εισαγωγή με προαγωγή κόμβου



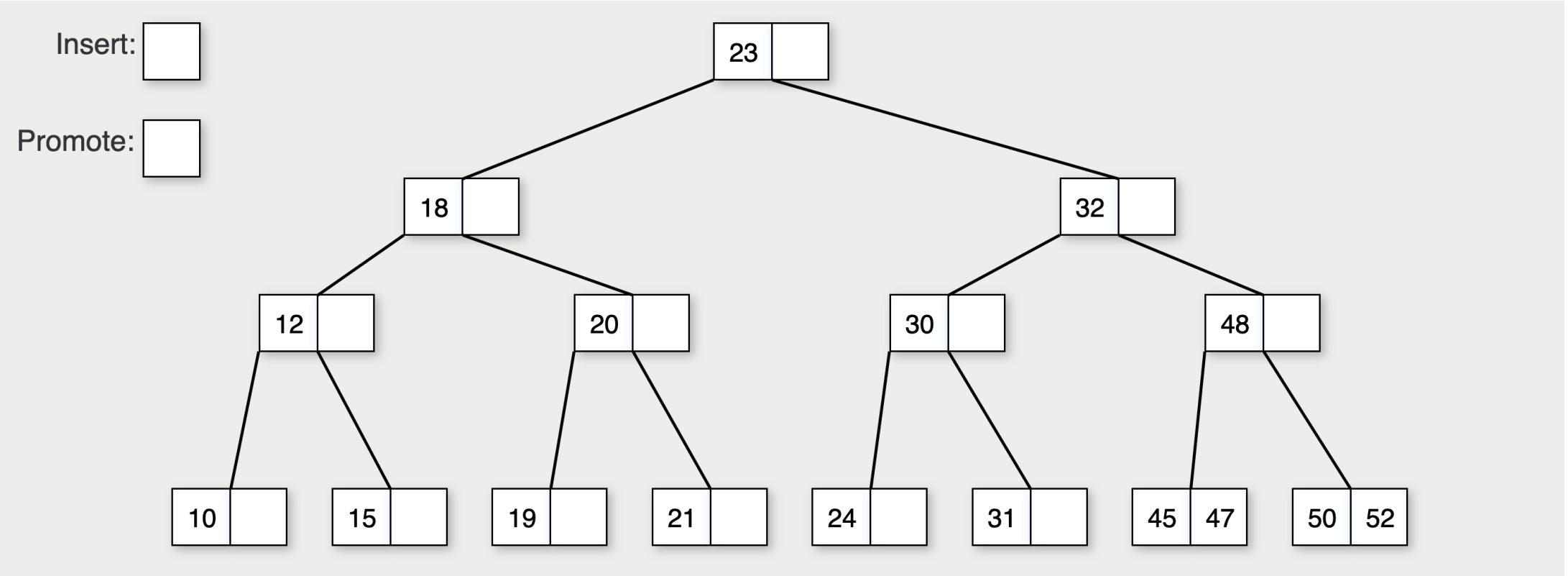
2-3 trees – εισαγωγή με προαγωγή κόμβου



2-3 trees – εισαγωγή με προαγωγή κόμβου και αύξηση ύψους



2-3 trees – εισαγωγή με προαγωγή κόμβου και αύξηση ύψους



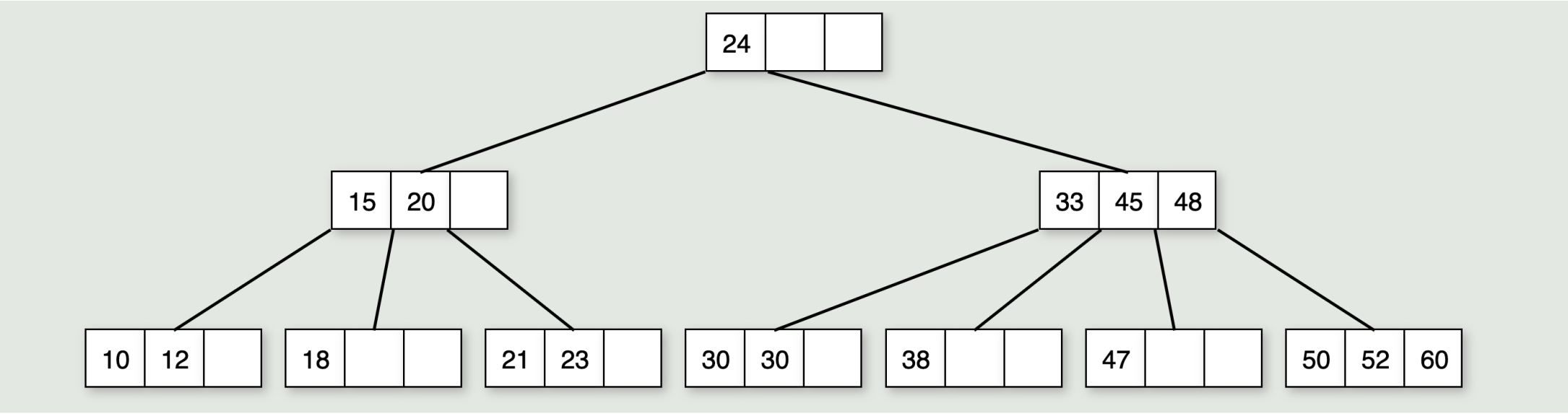
B-δένδρα

Ένα B-δένδρο *τάξης M είναι μια γενίκευση των 2-3 trees: ένα 2-3 δένδρο είναι ένα B-δένδρο τάξης 3.*

Ορίζεται ως είτε ένα κενό δένδρο ή ως ένα δένδρο αναζήτησης *m*-οδεύσεων *T* με τις παρακάτω ιδιότητες *T* :

1. Η ρίζα του δένδρου *T* έχει τουλάχιστον δύο υποδένδρα και το πολύ *M* υποδένδρα.
2. Όλοι οι εσωτερικοί κόμβοι του δένδρου *T* (εκτός της ρίζας) έχουν μεταξύ $\lceil M/2 \rceil$ και *M* υποδένδρα.
3. Όλα τα φύλλα του δένδρου *T* βρίσκονται στο ίδιο επίπεδο.

B-δένδρα



Αριθμός κλειδιών σε ένα Β-δένδρο

- Θεώρημα:
  Ο ελάχιστος αριθμός κλειδιών (τιμών των κόμβων) σε ένα Β-δένδρο τάξης $M \geq 2$ και ύψους $h \geq 0$ είναι
  $$n_h = 2\,[M / 2]^h - 1.$$
- Απόδειξη: με επαγωγή

Εισαγωγή στοιχείων σε Β-δένδρο

- Η «αναζήτηση» για το στοιχείο που πάμε να εισάγουμε καταλήγει σε φύλο
- Πάντα εισάγουμε ένα νέο στοιχείο σαν φύλλο
- Εάν το φύλλο περιέχει λιγότερα από $M - 1$ κλειδιά (τιμές) $[T_0, k_1, T_1, k_2, T_2, \ldots, k_{n-1}, T_{n-1}]$
  - Εισάγουμε το νέο στοιχείο (κλειδί) και φτιάχνουμε ένα κενό κόμβο όπως πρέπει για να κρατηθεί το σχήμα του δένδρου:

$$[T_0, k_1, T_1, k_2, T_2, \ldots, k_i, T_i, x, \varnothing, k_{i+1}, T_{i+1}, \ldots, k_{n-1}, T_{n-1}]$$

Εισαγωγή στοιχείων σε Β-δένδρο (συμπληρωμένα φύλλα)

- Χωρίζουμε τον κόμβο φύλλο στη μέση (σημειώστε ότι κάθε μισό εξακολουθεί να είναι μέρος ενός έγκυρου Β-δένδρου!)

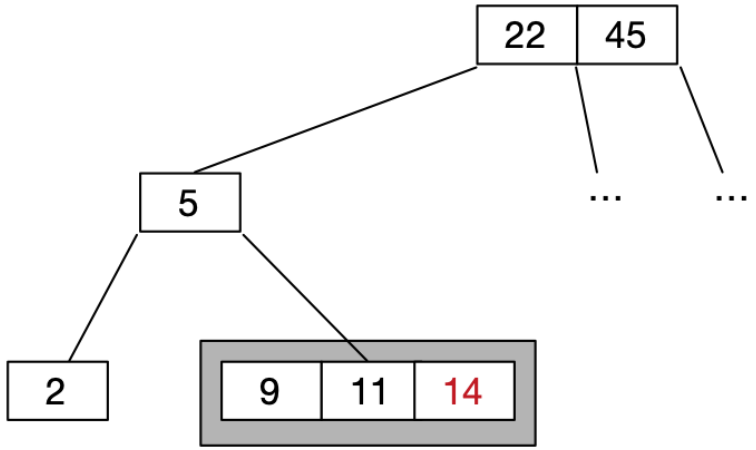
$$T'_L = [T_0, k_1, T_1, \ldots, k_{\lceil M/2 \rceil - 1}, T_{\lceil M/2 \rceil - 1}]$$

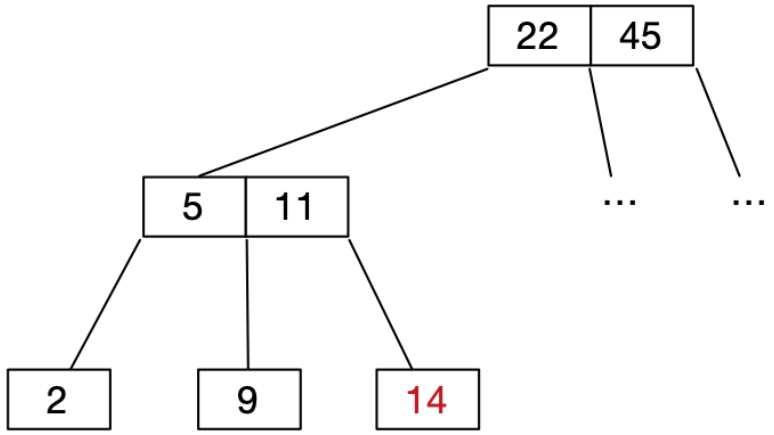
$$T'_R = [T_{\lceil M/2 \rceil}, k_{\lceil M/2 \rceil + 1}, T_{\lceil M/2 \rceil + 1}, \ldots, k_M, T_M]$$

Αντίστοιχα με την εισαγωγή σε δένδρα 2-3

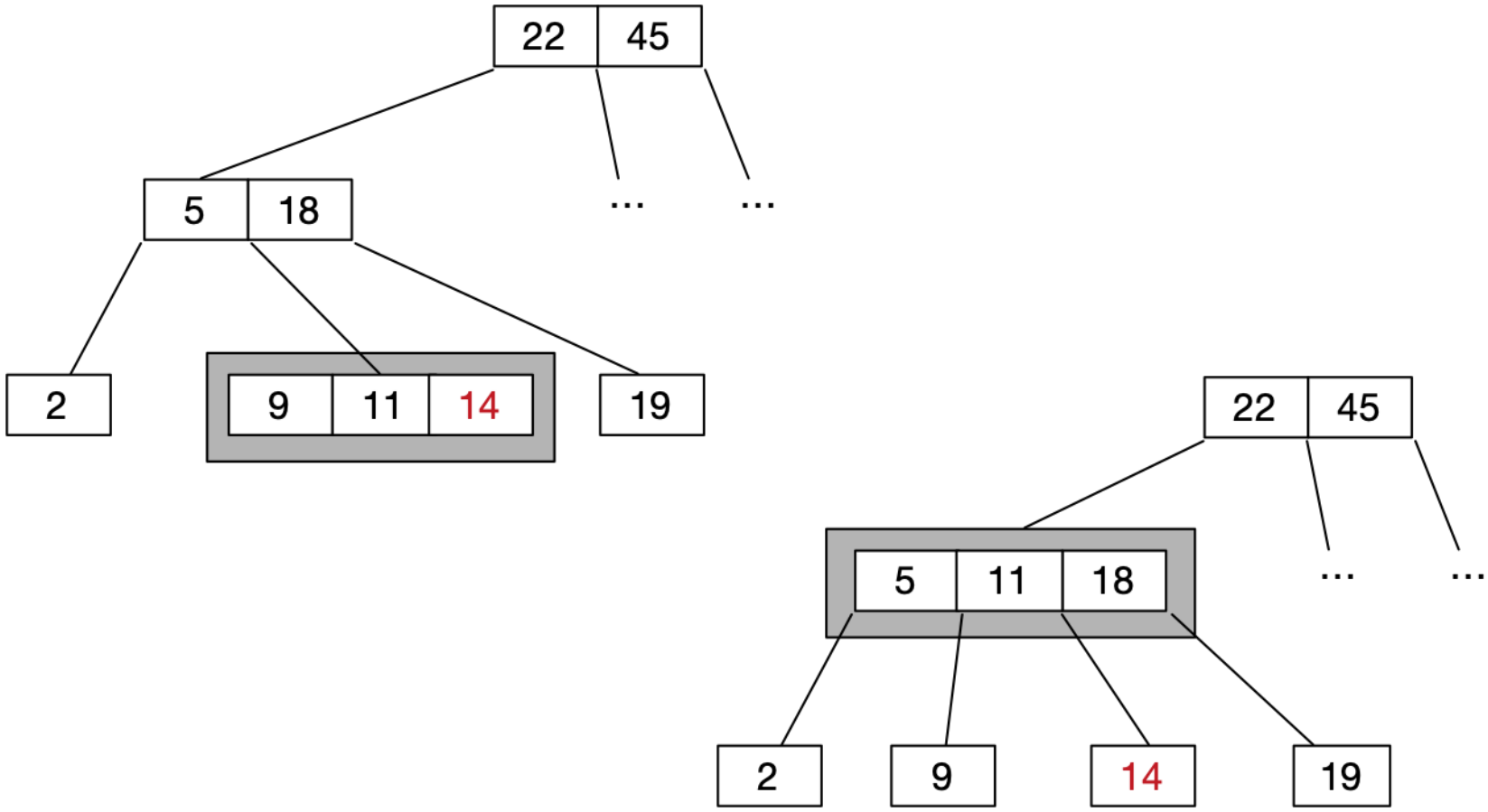
- Έστω ότι ο κόμβος *T* δεν είναι η ρίζα
  - **Έχουμε δύο κόμβους, T'_L and T'_R, και ένα κλειδί που περισσεύει, $k_{\lceil M/2 \rceil}$**
  - **T'_L αντικαθιστά τον κόμβο *T* στον γονέα του *T***
  - **Το ζευγάρι ($k_{\lceil M/2 \rceil}$, T'_R) εισάγεται αναδρομικά στον γονέα του T**
- Εάν ο κόμβος *T* είναι ρίζα, κατασκευάζουμε ένα νέο κόμβο: $[T'_L, k_{\lceil M/2 \rceil}, T'_R]$

Εισαγωγή στοιχείων σε Β-δένδρο – Παράδειγμα

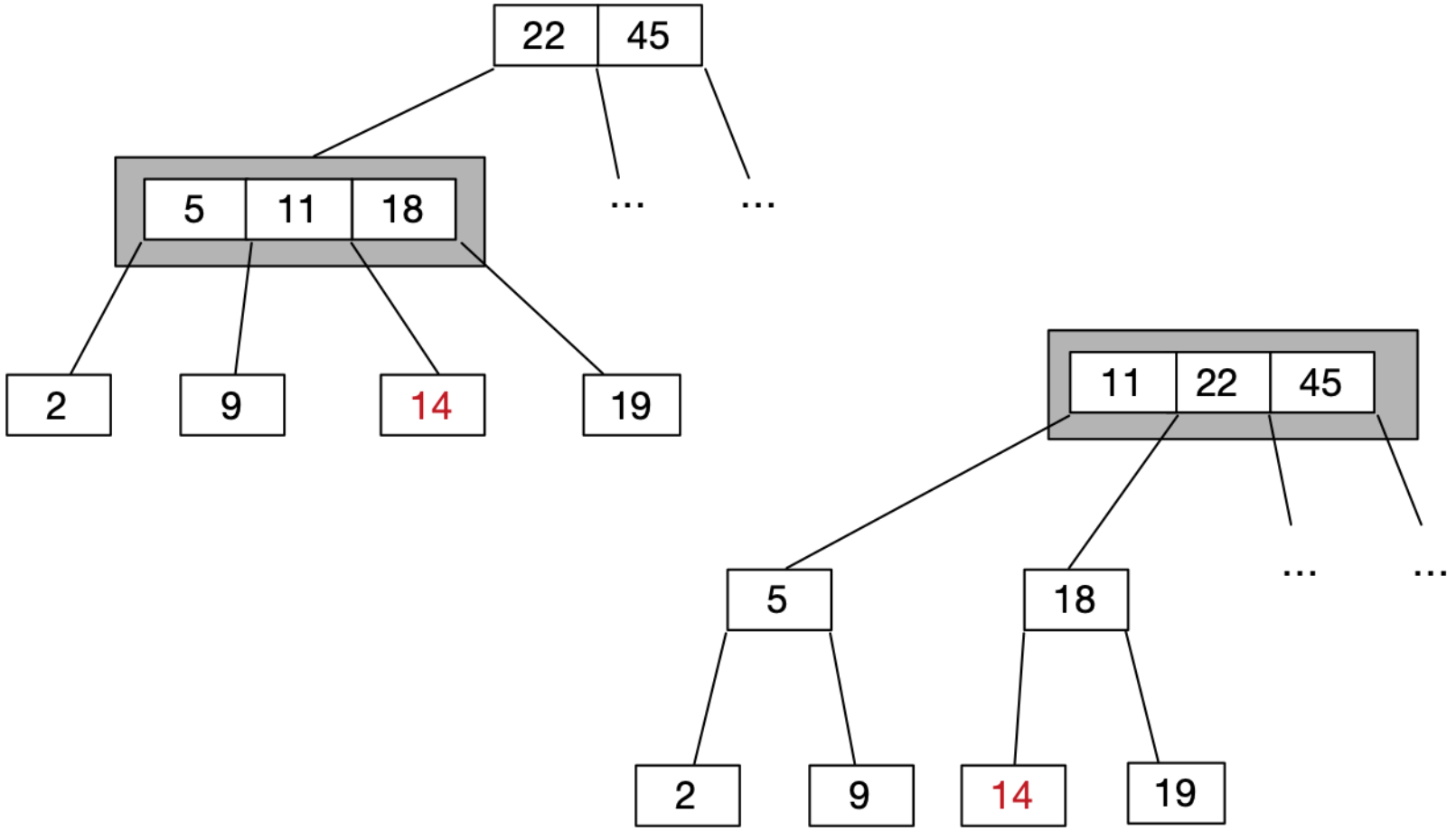




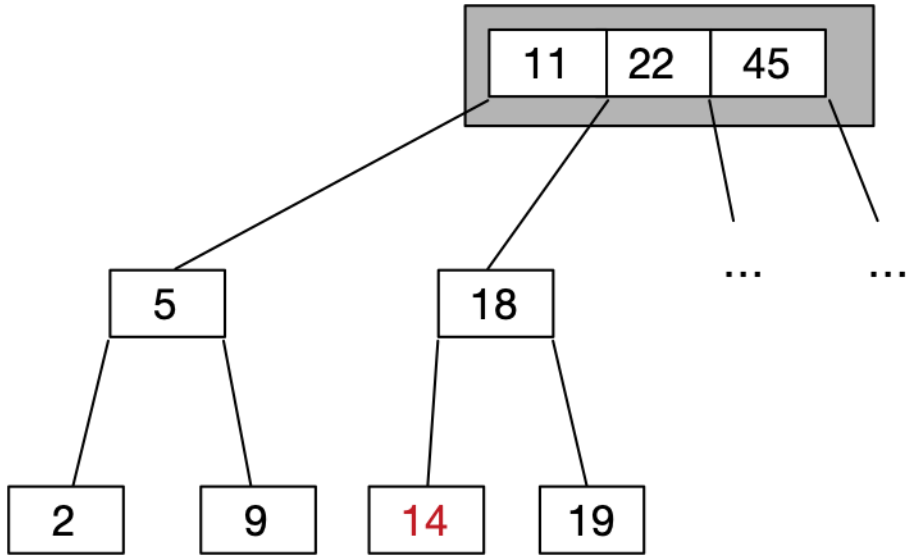
Εισαγωγή στοιχείων σε Β-δένδρο – Παράδειγμα

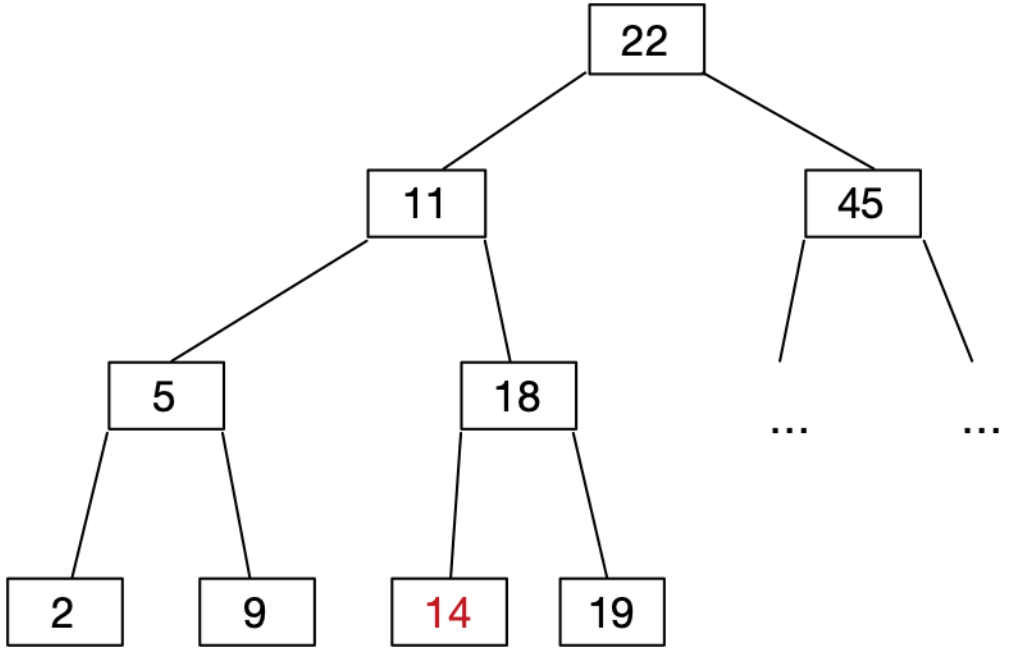


Εισαγωγή στοιχείων σε Β-δένδρο – Παράδειγμα



Εισαγωγή στοιχείων σε Β-δένδρο – Παράδειγμα





Διαγραφή στοιχείων από Β-δένδρο

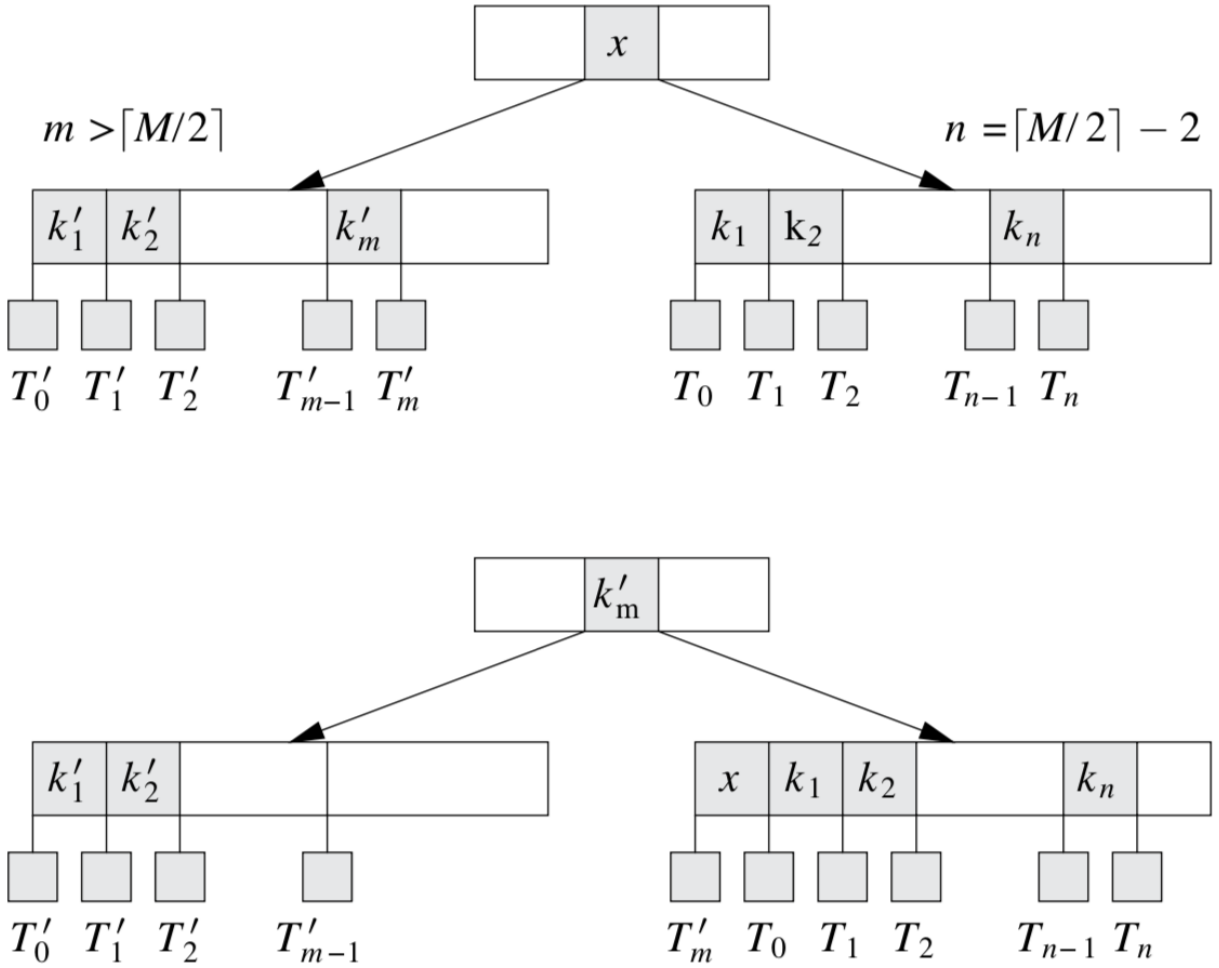
- Η «αναζήτηση» για το στοιχείο που πάμε να διαγράψουμε μπορεί να καταλήξει σε οποιοδήποτε κόμβο
- Αν ο κόμβος δεν είναι φύλλο διαγράφουμε το στοιχείο, αντικαθιστώντας το με προηγούμενο (δεξιότερο στοιχείο του αριστερού του υποδένδρου) ή το επόμενο (αριστερότερο στοιχείο του δεξιότερου υποδένδρου), όπως και στην περίπτωση των BST
- Διαγράφουμε το στοιχείο το οποίο χρησιμοποιήσαμε στην αντικατάσταση (αυτό είναι σε φύλλο - *γιατί;*)
- Η διαγραφή αυτή μπορεί να δημιουργήσει πρόβλημα (αν δεν είναι το φύλλο αυτό ρίζα - *γιατί αν είναι στη ρίζα δεν υπάρχει πρόβλημα;*).

Διαγραφή στοιχείων από Β-δένδρο

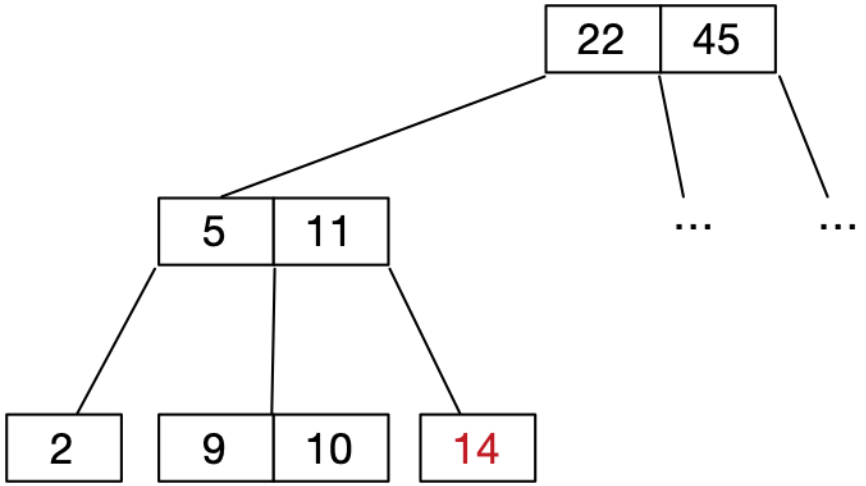
- Το πρόβλημα μπορεί να δημιουργηθεί από την παραβίαση της συνθήκης 2 του ορισμού του Β-δένδρου (*όλοι οι εσωτερικοί κόμβοι του δένδρου Τ (εκτός της ρίζας) έχουν μεταξύ* $\lceil M / 2 \rceil$ *και Μ υποδένδρα.*)
- Η διαχείριση του προβλήματος αυτού γίνεται με τις εξής τεχνικές:
  - **Περιστροφή**, αν υπάρχει αριστερός ή δεξιός γείτονας με περισσότερα από $\lceil M / 2 \rceil$ παιδιά (βλ. Σχήμα επόμενης διαφάνειας)
  - **Συγχώνευση**, αν και οι δύο γείτονές του (αριστερός και δεξιός) έχουν ακριβώς $\lceil M / 2 \rceil$ παιδιά (βλ. Σχήμα μεθεπόμενης διαφάνειας)

Διαγραφή στοιχείων σε Β-δένδρο

Περιστροφή

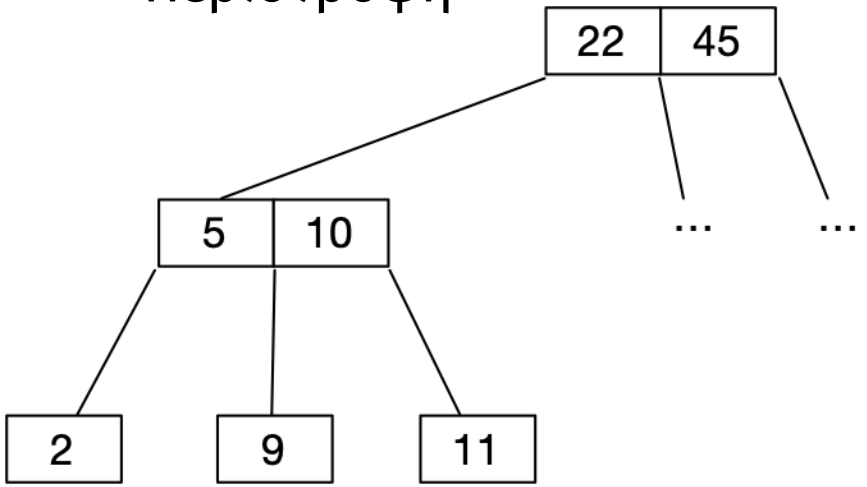


Διαγραφή στοιχείων σε Β-δένδρο



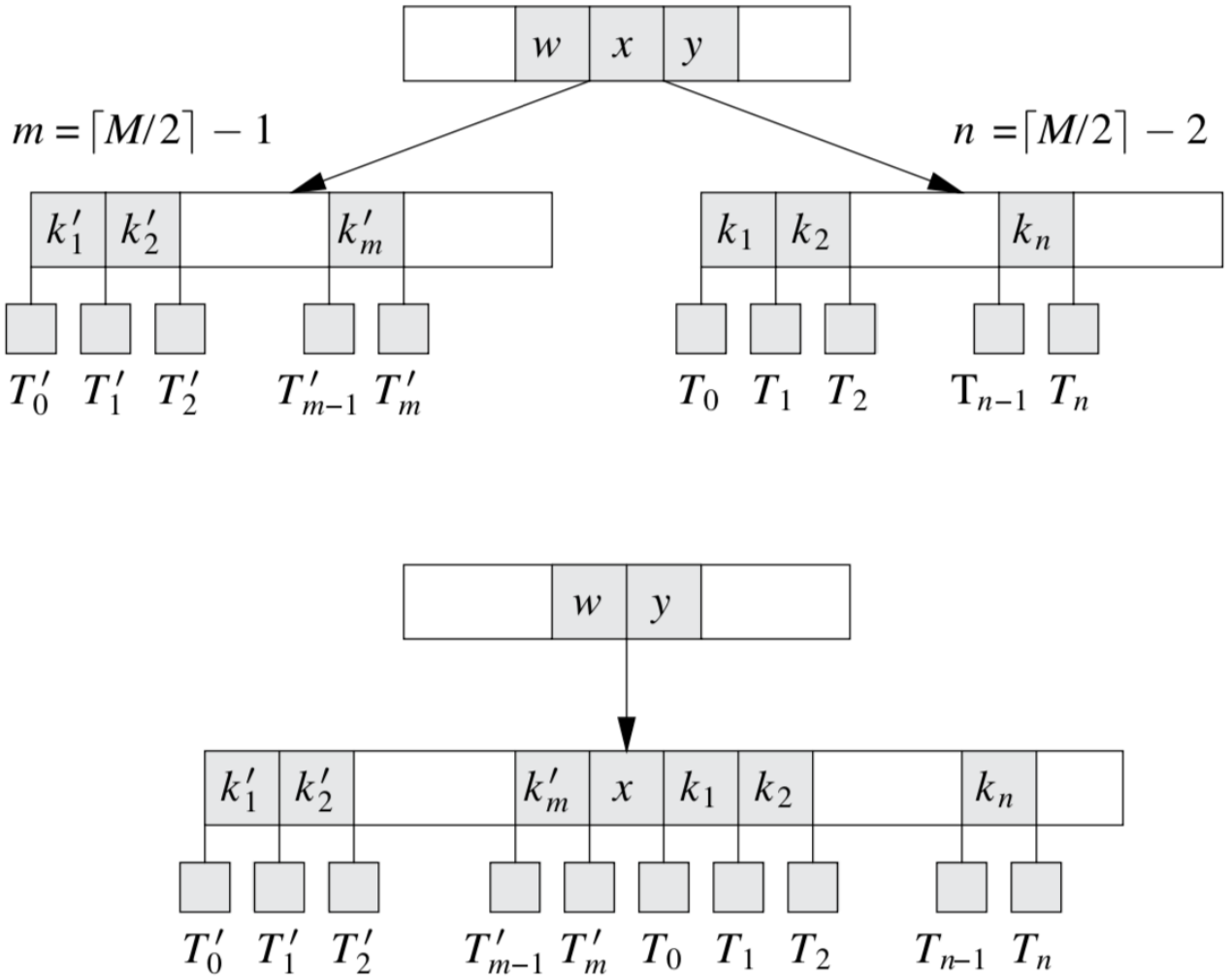
Διαγραφή 14

Περιστροφή

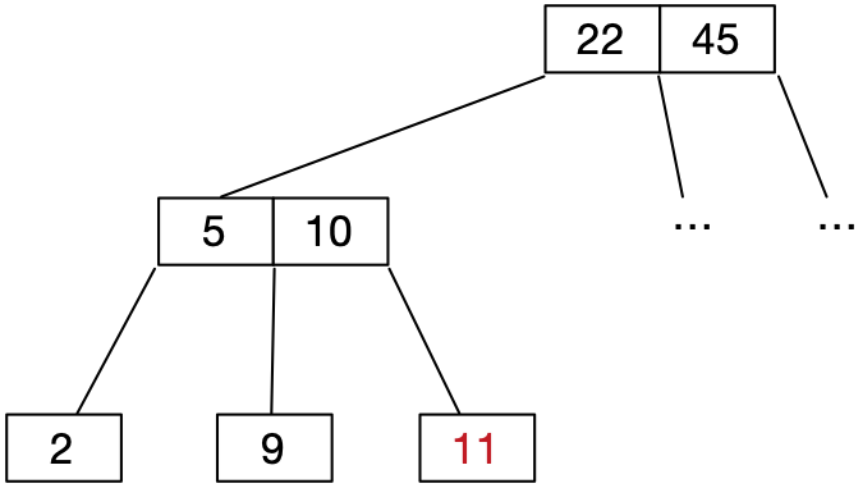


Διαγραφή στοιχείων σε Β-δένδρο

Συγχώνευση

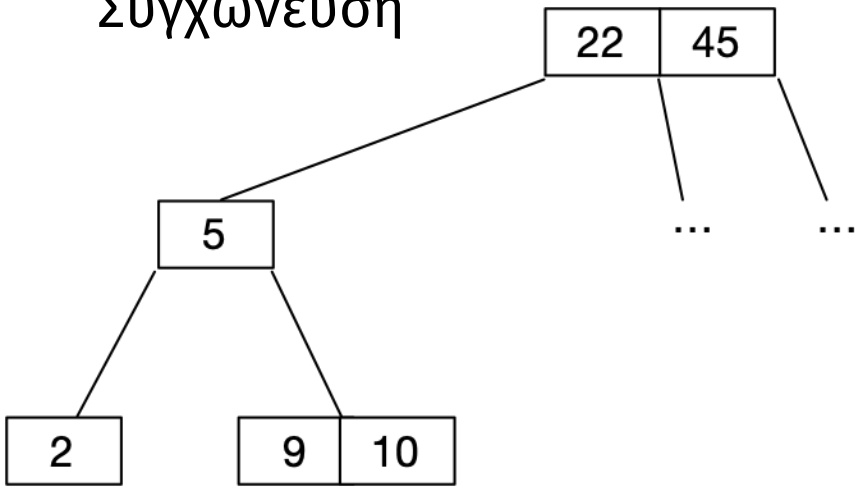


Διαγραφή στοιχείων σε Β-δένδρο



Διαγραφή 11

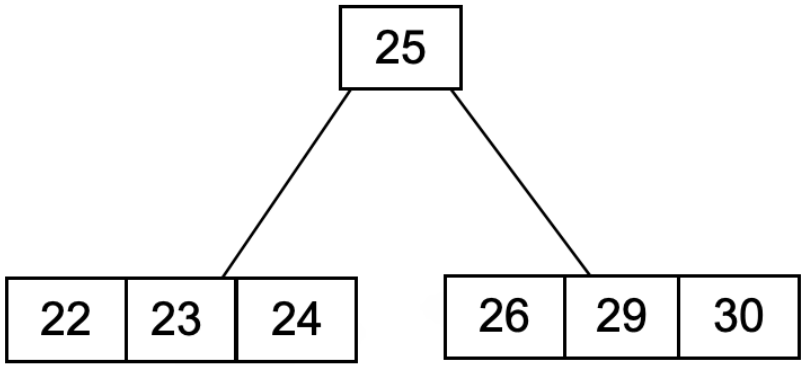
Συγχώνευση



Ασκήσεις

Δίνεται το m-way δένδρο 4 οδεύσεων του παρακάτω σχήματος, στο οποίο αποθηκεύουμε ως κλειδιά φυσικούς αριθμούς. Ποια από τις παρακάτω προτάσεις που το αφορούν είναι σωστή;

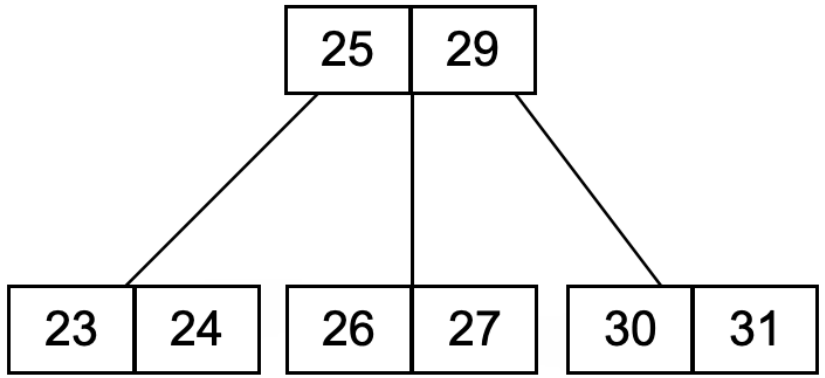
Α. Το δένδρο αυτό δεν είναι Β-δένδρο, διότι η ρίζα έχει λιγότερα από το επιτρεπτό στοιχεία.
Β. Το δένδρο αυτό δεν είναι Β-δένδρο, λόγω των τιμών των κλειδιών του.
Γ. Το δένδρο αυτο είναι Β-δένδρο.
Δ. Καμία από τις άλλες προτάσεις δεν είναι σωστή.



Ασκήσεις

Δίνεται το Β-δένδρο 3 οδεύσεων του παρακάτω σχήματος, στο οποίο αποθηκεύουμε ως κλειδιά φυσικούς αριθμούς. Για την εισαγωγή στοιχείων χρησιμοποιούμε το γνωστό αλγόριθμο εισαγωγής στοιχείων σε Β-δένδρα. Ποια από τις παρακάτω προτάσεις είναι σωστή;

Α. Δεν μπορει να εισαχθεί κανένα στοιχείο, χωρίς να αλλάξει ύψος το δένδρο αυτό.
Β. Όποιο στοιχείο και να εισαχθεί το δένδρο δεν θα αλλάξει ύψος.
Γ. Το δένδρο μπορεί να αλλάξει ύψος, μπορεί και όχι, εξαρτάται από την τιμή του στοιχείου που θα εισαχθεί.
Δ. Καμία από τις άλλες προτάσεις δεν είναι σωστή.



Ασκήσεις

Δίνεται το Β-δένδρο 3 οδεύσεων του παρακάτω σχήματος, στο οποίο αποθηκεύουμε ως κλειδιά φυσικούς αριθμούς. Για τις εισαγωγές και διαγραφές στοιχείων χρησιμοποιούμε τους γνωστούς αλγόριθμους. Ποια από τις παρακάτω προτάσεις είναι σωστή;

Α. Δεν μπορεί να διαγραφεί κανένα στοιχείο, χωρίς να αλλάξει ύψος το δένδρο αυτό.
Β. Όποιο στοιχείο και να διαγραφεί το δένδρο δεν θα αλλάξει ύψος.
Γ. Το δένδρο μπορει να αλλάξει ύψος, μπορεί και όχι, εξαρτάται από την τιμή του στοιχείου που θα διαγραφεί.
Δ. Καμία από τις άλλες προτάσεις δεν είναι σωστή