Δομές Δεδομένων σε C

Μάθημα 9:

Δένδρα AVL

Δημήτρης Ψούνης

Περιεχόμενα Μαθήματος

Α. Θεωρία

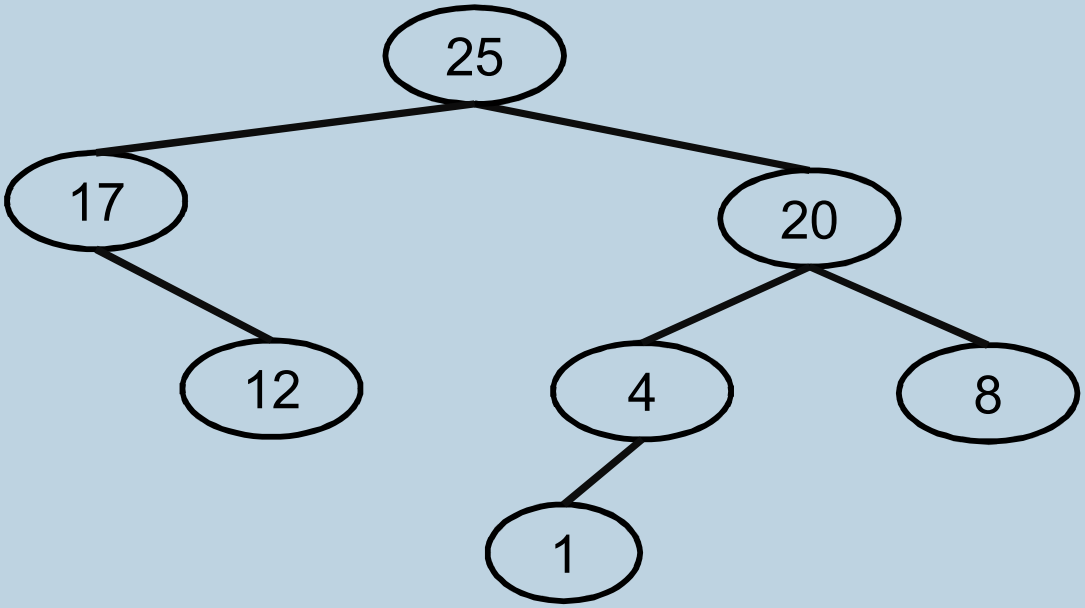
1. Δένδρο AVL

1. Ορισμός Δένδρου AVL

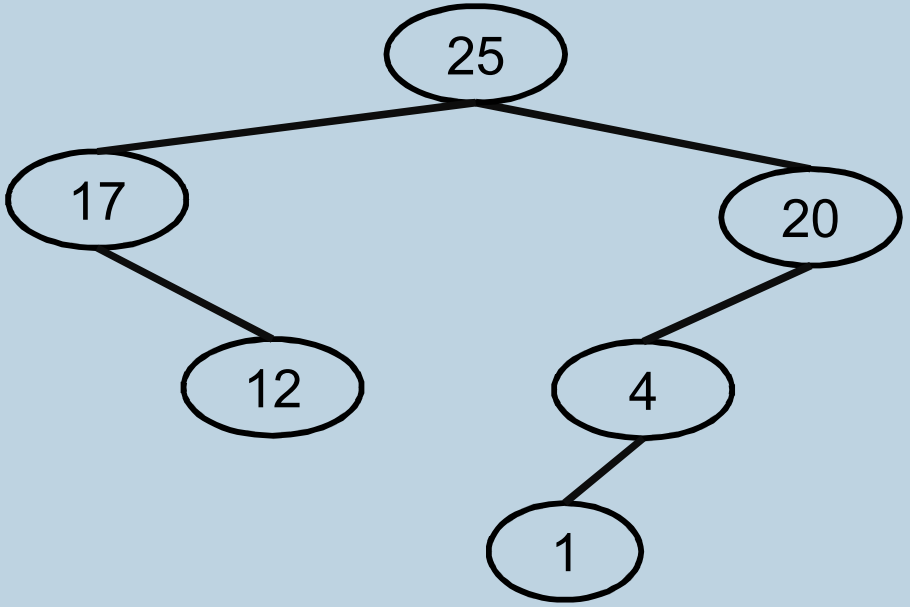
Το **«Δένδρο AVL» (Adelson-Velskii-Landis)** είναι ένα δυαδικό δένδρο στο οποίο:

- Σε κάθε κόμβο, το ύψος του αριστερού και του δεξιού υποδένδρου διαφέρει το πολύ κατά 1

Παράδειγμα AVL δένδρου:



Το ακόλουθο δεν είναι AVL δένδρο:



Παρατήρηση: Τα δένδρα AVL χρησιμοποιούνται:

- ώστε να ισοζυγίζουν ΔΔΑ και να μην επιτρέπουν να εκτρέπονται σε εκφυλισμένες μορφές.

Α. Θεωρία

1. Δένδρο AVL

2. Μαθηματικές Ιδιότητες

Το ελάχιστο ύψος του δένδρου AVL είναι $h = O(\log n)$

Έστω $N(h)$: Το ελάχιστο πλήθος κόμβων δένδρου ύψους h. Ισχύει ότι:

$$N(h) = 1 + N(h-1) + N(h-2)$$

Η επίλυση αυτής της αναδρομικής σχέσης (βλ. ΠΛΗ30 μάθημα 1.4-1.5) είναι:

$$N(h) = O\left(1{,}61^{h}\right) \Rightarrow h = O(\log n)$$

Το μέγιστο ύψος του δένδρου AVL είναι $h = O(\log n)$

Έστω $N(h)$: Το μέγιστο πλήθος κόμβων δένδρου ύψους h. Ισχύει ότι:

$$N(h) = 1 + N(h-1) + N(h-1)$$

Η επίλυση αυτής της αναδρομικής σχέσης είναι:

$$N(h) = O\left(2^{h}\right) \Rightarrow h = O(\log n)$$

Συνεπώς το ύψος του δένδρου είναι το πολύ logn, άρα οι αναζητήσεις σε αυτό θα απαιτούν το πολύ logn βήματα

Σημείωση: Η βελτίωση είναι σημαντική σε σχέση με τα n βήματα που θα χρειαζόταν στην χειρότερη περίπτωση σε ένα ΔΔΑ

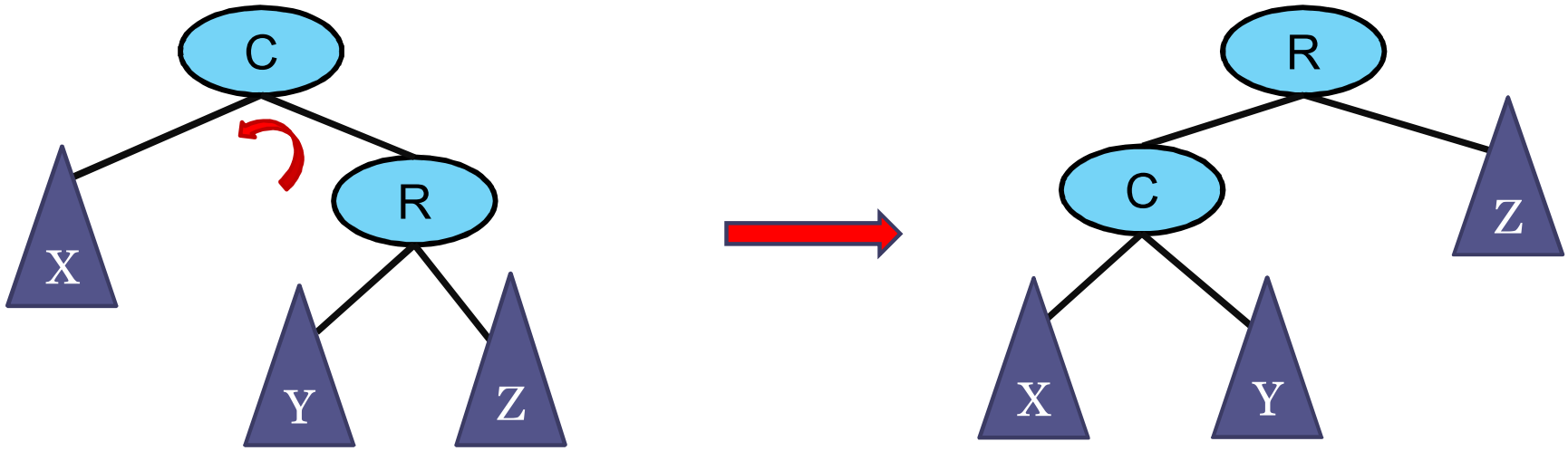
Α. Θεωρία

1. Δένδρο AVL

3. Περιστροφές

Βασική παρατήρηση: Μπορούμε να αναδιατάξουμε τους κόμβους ενός ΔΔΑ και να διατηρείται η διάταξη του

- **Αριστερή περιστροφή:** Ένας κόμβος μπορεί να δώσει τη θέση του στο δεξί παιδί του (και να γίνει αριστερό παιδί του), διορθώνοντας τη διάταξη των υποδένδρων τους:



$X < C < (Y < R < Z)$

$(X < C < Y) < R < Z$

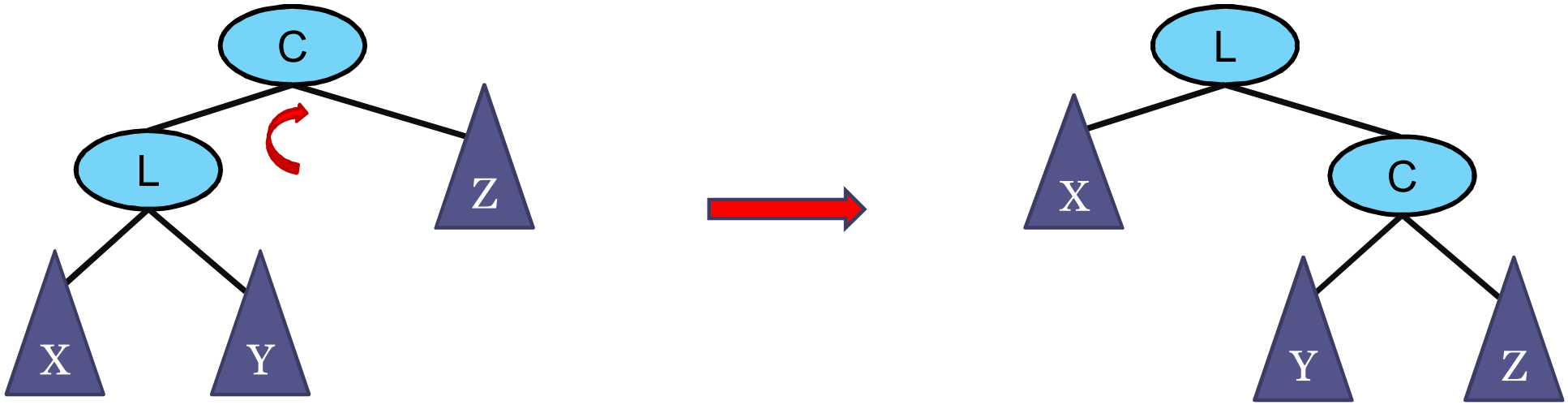
Αριστερή περιστροφή στο C

Α. Θεωρία

1. Δένδρο AVL

3. Περιστροφές

- **Δεξιά περιστροφή**: Ένας κόμβος μπορεί να δώσει τη θέση του στο αριστερό παιδί του (και να γίνει δεξί παιδί του), διορθώνοντας τη διάταξη των υποδένδρων τους:



$(X < L < Y) < C < Z$

$X < L < (Y < C < Z)$

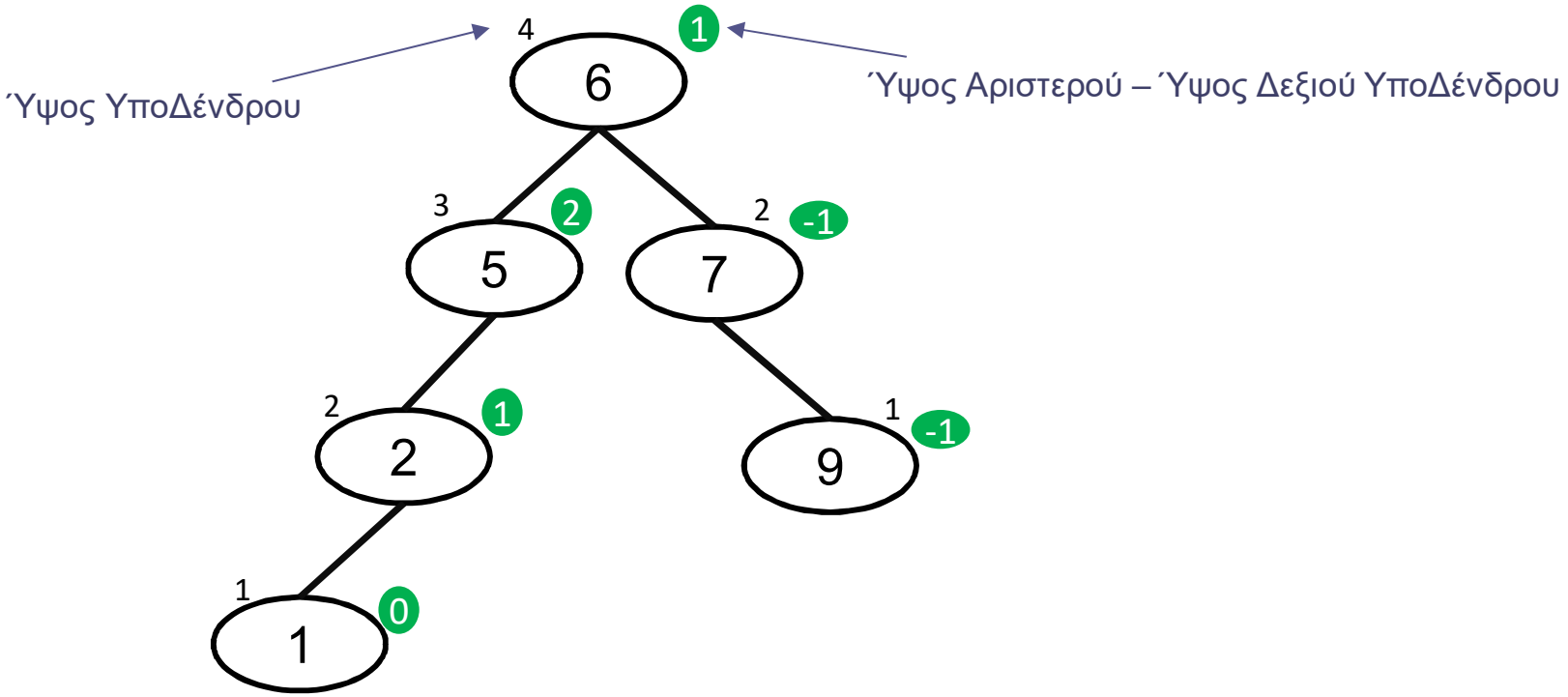
Δεξιά περιστροφή στο C

Α. Θεωρία

1. Δένδρο AVL

4. Παραβάσεις

- Κάθε κόμβος διατηρεί το ύψος του υποδένδρου στο οποίο είναι ρίζα.



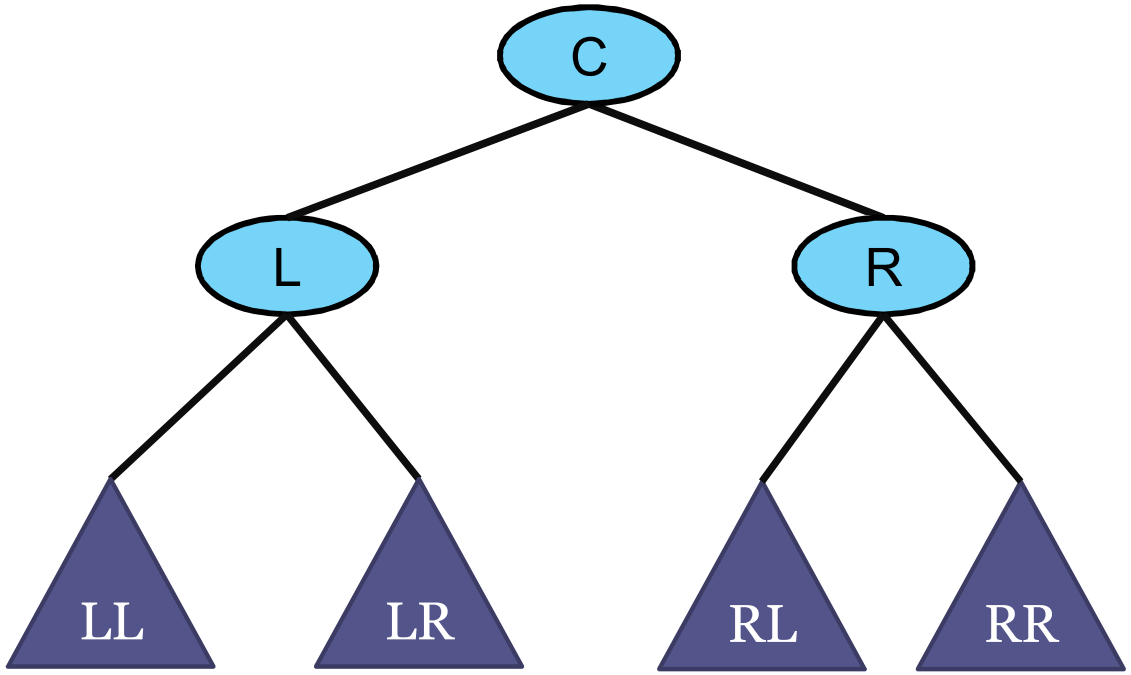
- Και υπολογίζει τη διαφορά ύψους αριστερού – δεξιού υποδένδρου.
- Με βάση τον ορισμό του AVL υπάρχει πρόβλημα, αν ένας κόμβος έχει διαφορά ίση με 2 ή -2.
  - Τότε με περιστροφές, γίνεται διόρθωση στην ανισοροπία.

Α. Θεωρία

1. Δένδρο AVL

4. Παραβάσεις

- Η κατασκευή του υποδένδρου γίνεται όπως στο ΔΔΑ.
- Όμως, όταν εντοπίζονται ανισορροπίες σε έναν κόμβο (εδώ στο C):
  - Διακρίνονται περιπτώσεις ανάλογα σε ποιο υποδένδρο από τα «εγγόνια» του κόμβου, έγινε η εισαγωγή που προκάλεσε την ανισορροπία:



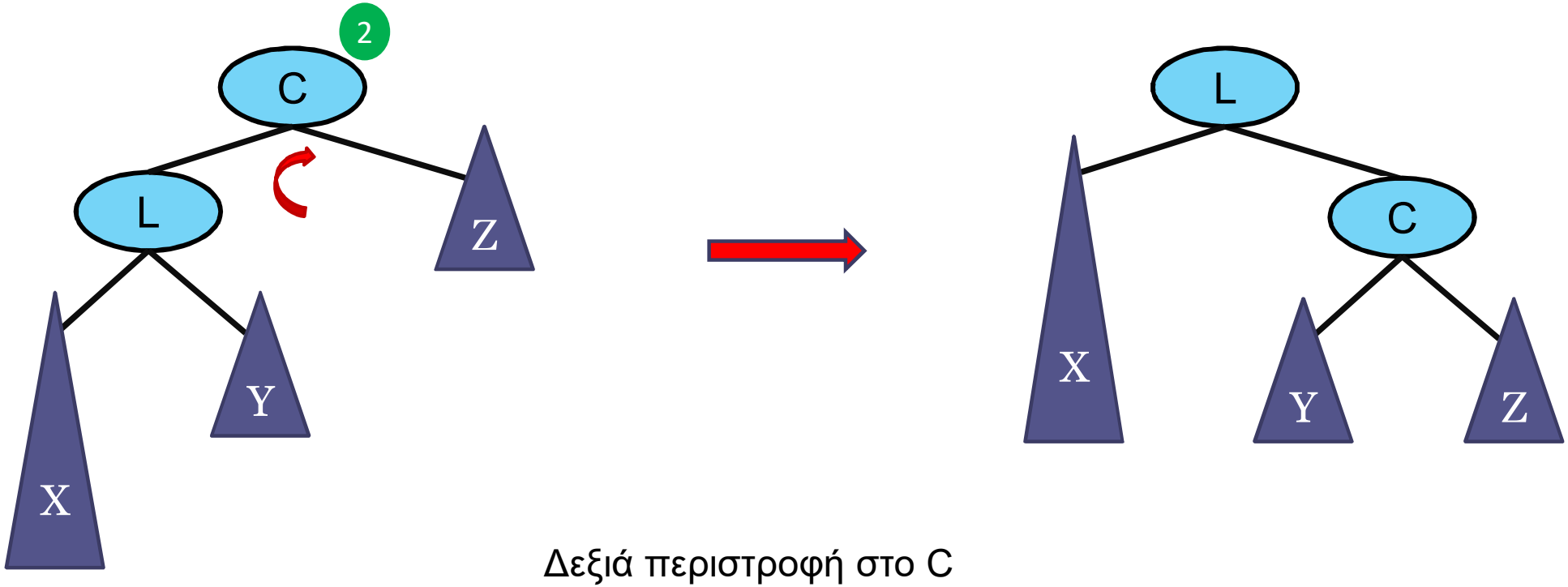
  - Οι περιπτώσεις είναι τέσσερις (LL, LR, RL, RR) όσα και τα εγγόνια του κόμβου.

Α. Θεωρία

1. Δένδρο AVL

4.1. Παράβαση LL

- Η **παράβαση LL** προκύπτει όταν προκληθεί ανισορροπία επειδή έγινε εισαγωγή στο αριστερό υποδένδρο του αριστερού παιδιού του κόμβου C:
  - Έχει αρχική διαφορά υποδένδρων ίση με 2
  - Επιλύεται με μία αριστερή περιστροφή στον κόμβο C



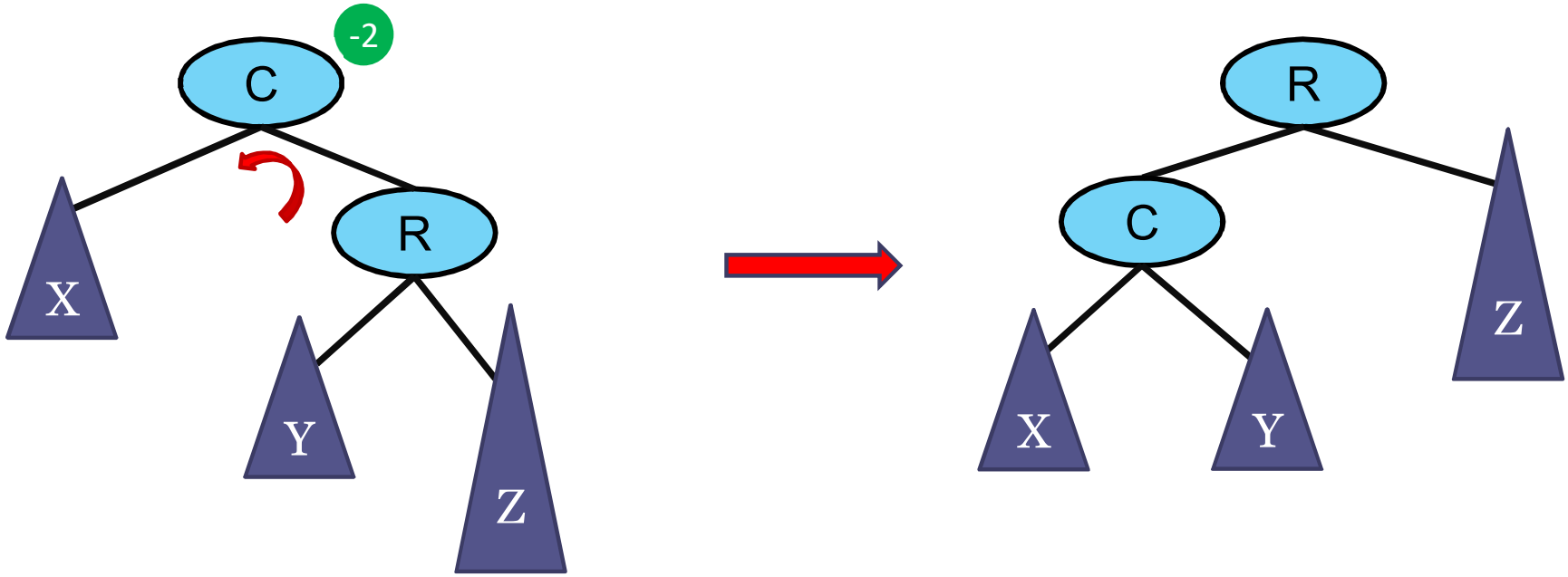
Δεξιά περιστροφή στο C

Α. Θεωρία

1. Δένδρο AVL

4.2. Παράβαση RR

- Η **παράβαση RR** προκύπτει όταν προκληθεί ανισορροπία επειδή έγινε εισαγωγή στο δεξί υποδένδρο του δεξιού παιδιού του κόμβου C:
  - Έχει αρχική διαφορά υποδένδρων ίση με -2
  - Επιλύεται με μία αριστερή περιστροφή στον κόμβο C

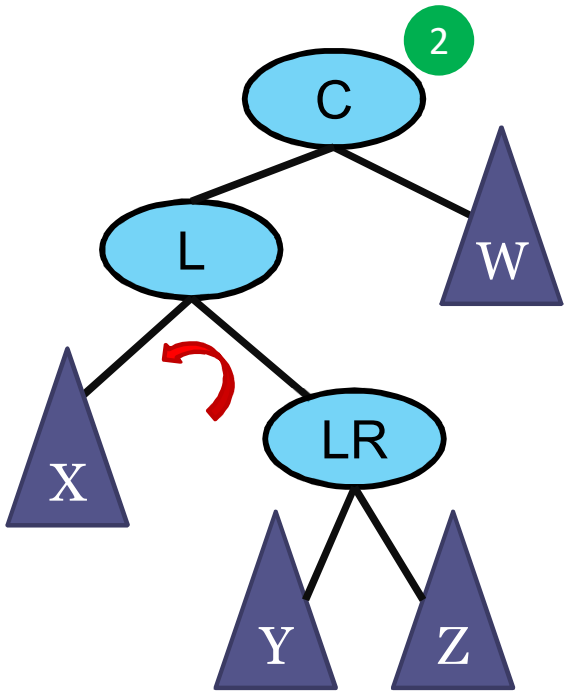


Α. Θεωρία

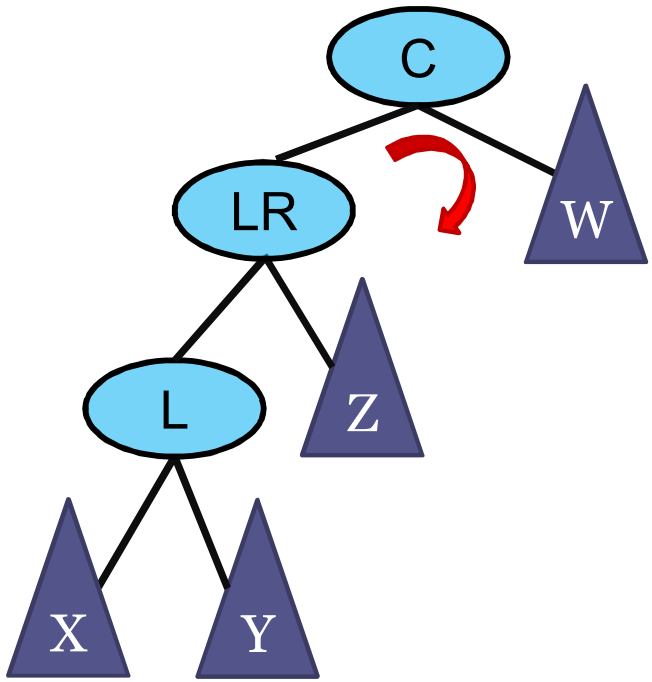
1. Δένδρο AVL

4.3. Παράβαση LR

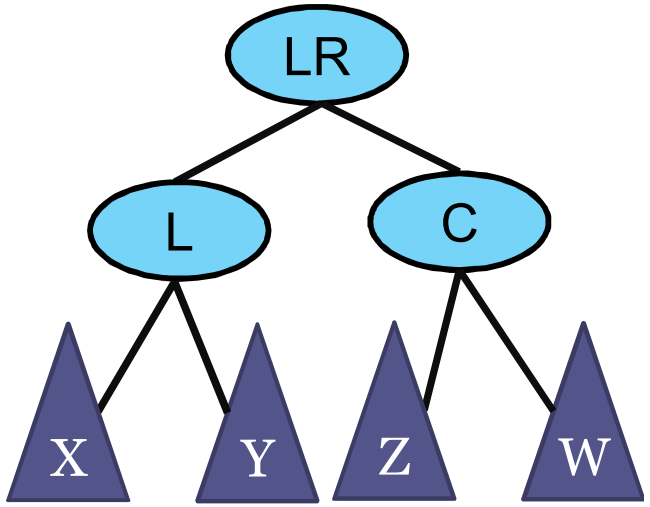
- Η **παράβαση LR** προκύπτει όταν προκληθεί ανισορροπία επειδή έγινε εισαγωγή στο δεξί υποδένδρο του αριστερού παιδιού του κόμβου C:
  - Έχει αρχική διαφορά υποδένδρων ίση με 2
  - Επιλύεται με μία αριστερή περιστροφή στον κόμβο L, και έπειτα δεξιά περιστροφή του C



1. Αριστερη περιστροφή στο L



2. Δεξιά περιστροφή στο C

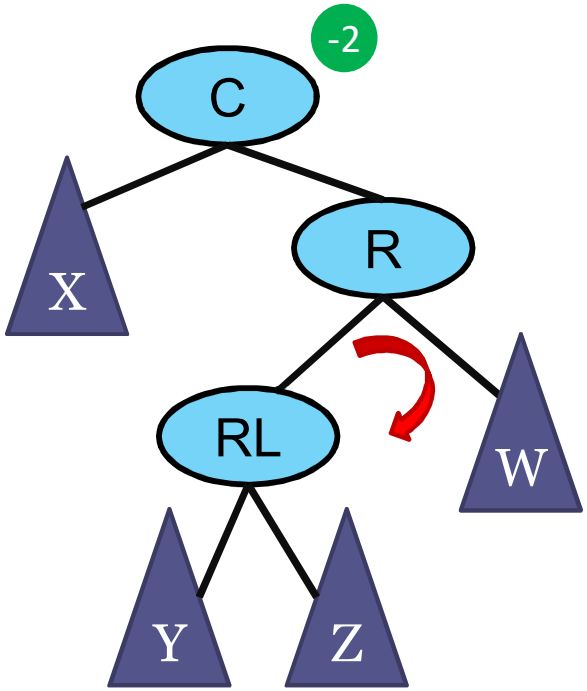


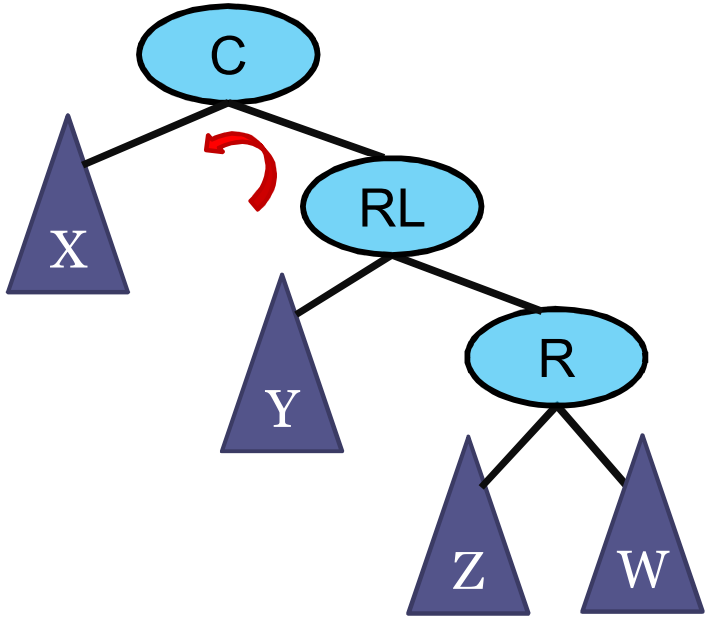
Α. Θεωρία

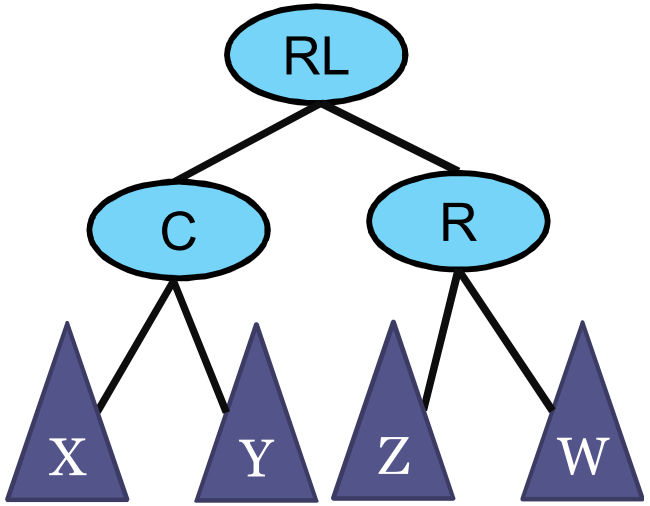
1. Δένδρο AVL

4.4. Παράβαση RL

- Η **παράβαση RL** προκύπτει όταν προκληθεί ανισορροπία επειδή έγινε εισαγωγή στο αριστερό υποδένδρο του δεξιού παιδιού του κόμβου C:
  - Έχει αρχική διαφορά υποδένδρων ίση με -2
  - Επιλύεται με μία δεξιά περιστροφή στον κόμβο R, και έπειτα αριστερή περιστροφή του C







1. Δεξιά περιστροφή στο R

2. Αριστερή περιστροφή στο C

Α. Θεωρία

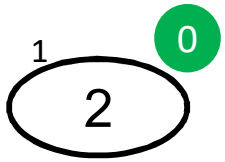
1. Δένδρο AVL

5. Εισαγωγή σε AVL – Παράδειγμα

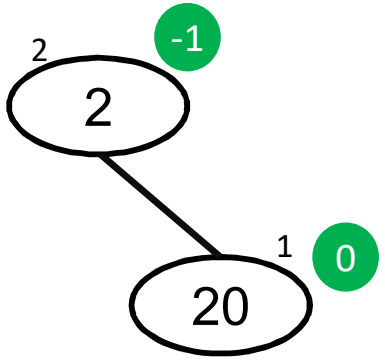
- Ο αλγόριθμος είναι ίδιος με αυτόν της εισαγωγής σε ΔΔΑ:
  - Μετά την εισαγωγή όμως διατρέχεται το μονοπάτι του νέου κόμβου μέχρι τη ρίζα αντίστροφα και γίνονται οι απαραίτητες διορθώσεις μέσω περιστροφών

- Θα μελετήσουμε αυτές τις εισαγωγές μέσω ενός παραδείγματος

Εισαγωγή «2»



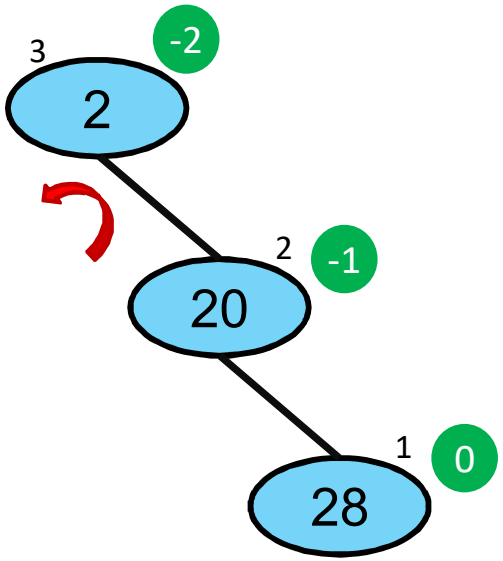
Εισαγωγή «20»



Α. Θεωρία

1. Δένδρο AVL

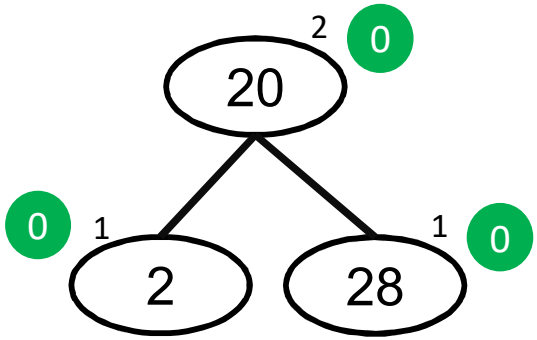
5. Εισαγωγή σε AVL – Παράδειγμα

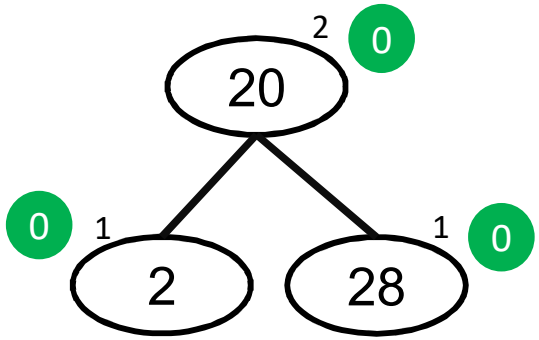


Εισαγωγή «28»

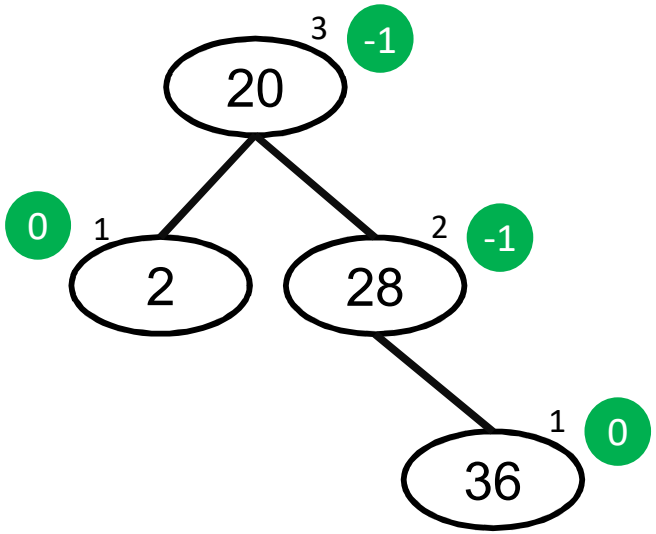
Παράβαση RR

Διόρθωση «L»





Εισαγωγή «36»

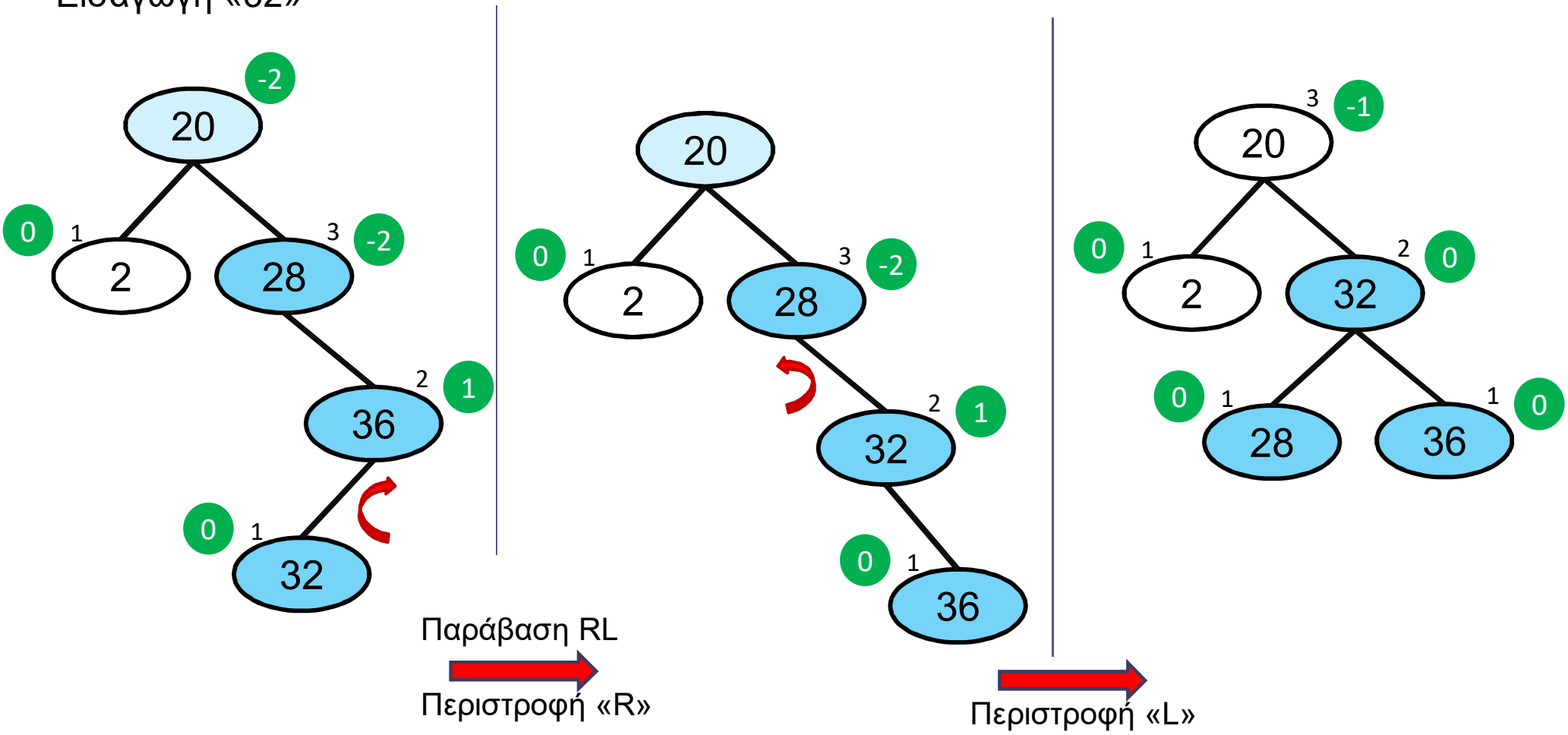


Α. Θεωρία

1. Δένδρο AVL

5. Εισαγωγή σε AVL – Παράδειγμα

Εισαγωγή «32»



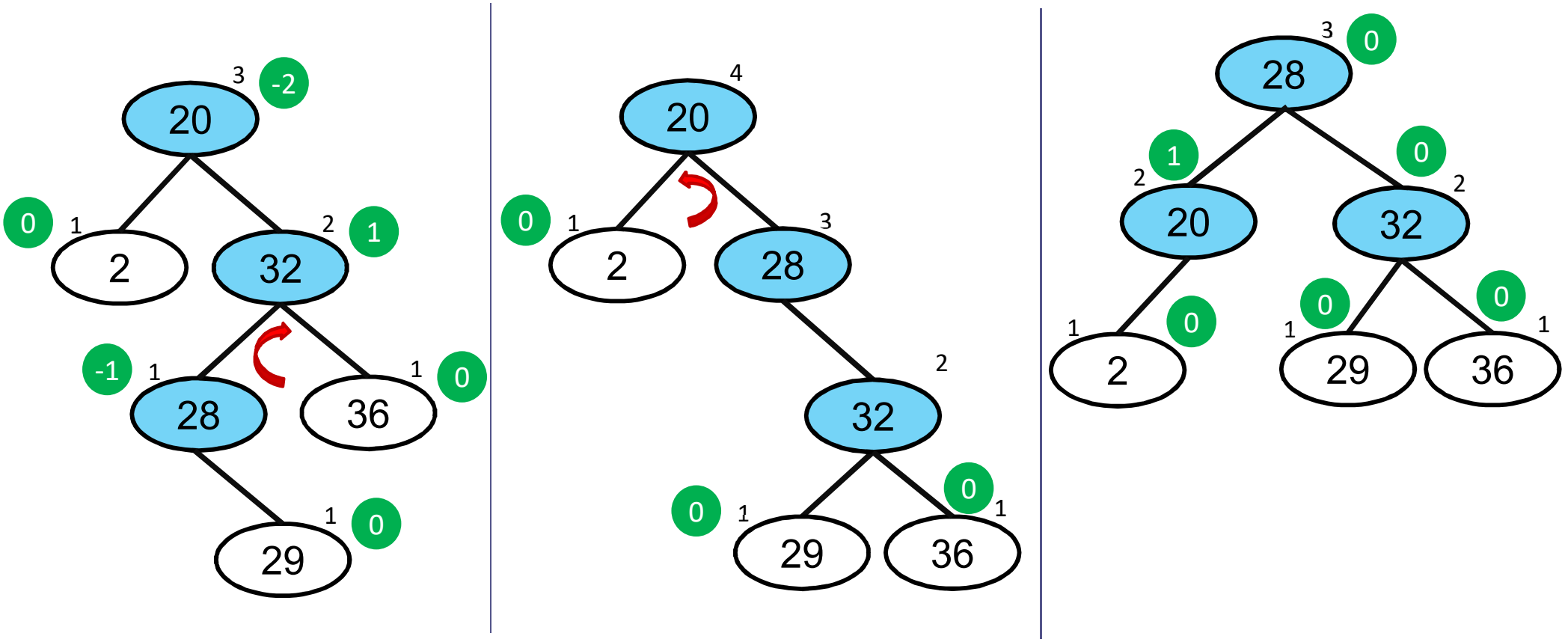
Σημείωση: Η διόρθωση θα γίνεται στον κόμβο του χαμηλοτέρου επιπέδου. Εδώ και το 20 είναι ανισόρροπο, αλλά η διόρθωση στο 28, διορθώνει την ισορροπία και του 20.

Α. Θεωρία

1. Δένδρο AVL

5. Εισαγωγή σε AVL – Παράδειγμα

Εισαγωγή «29»

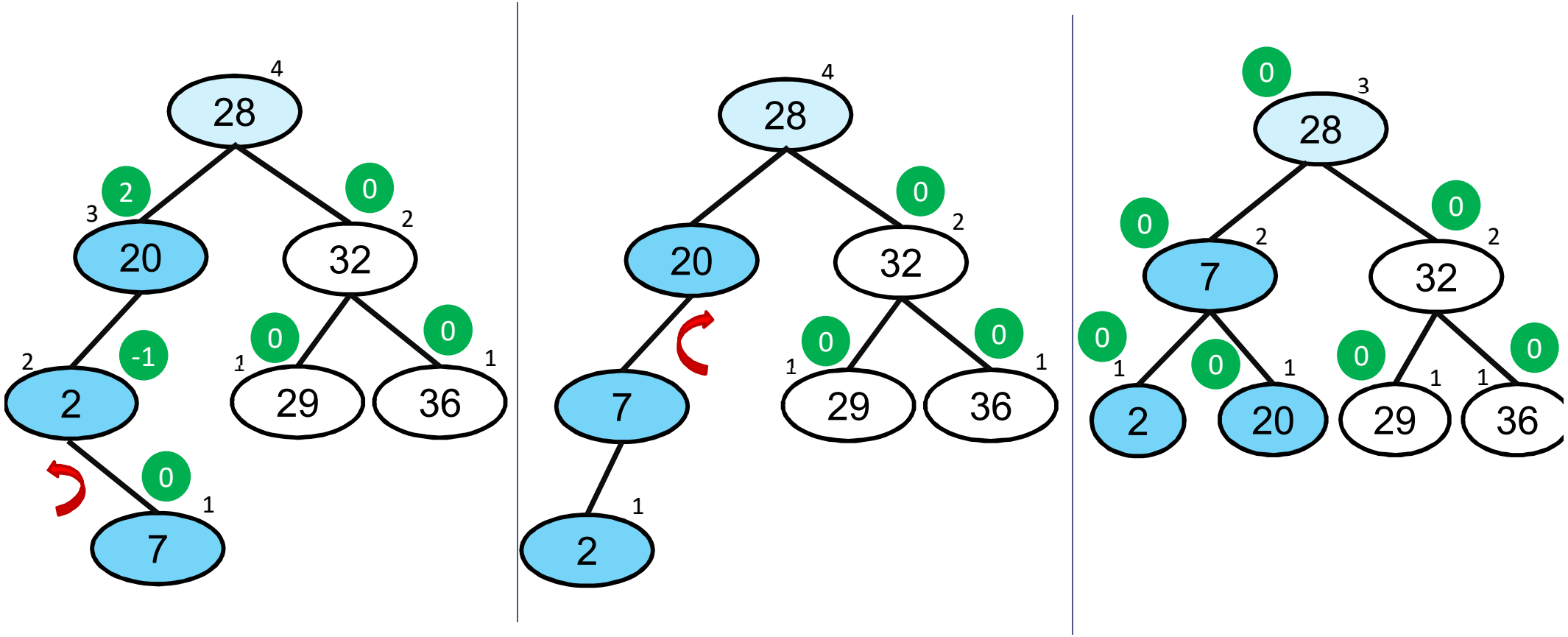


Α. Θεωρία

1. Δένδρο AVL

5. Εισαγωγή σε AVL – Παράδειγμα

Εισαγωγή «7»

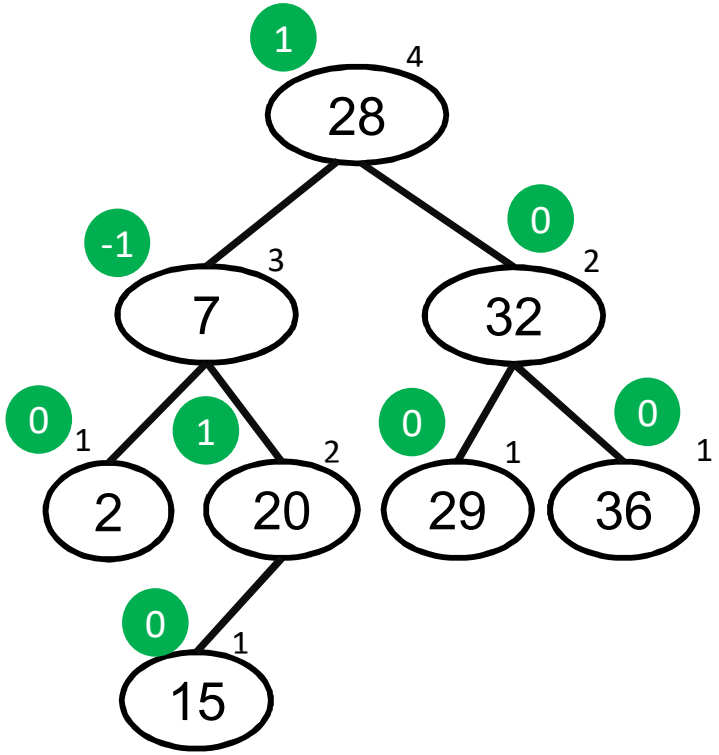


Α. Θεωρία

1. Δένδρο AVL

5. Εισαγωγή σε AVL – Παράδειγμα

Εισαγωγή «15»

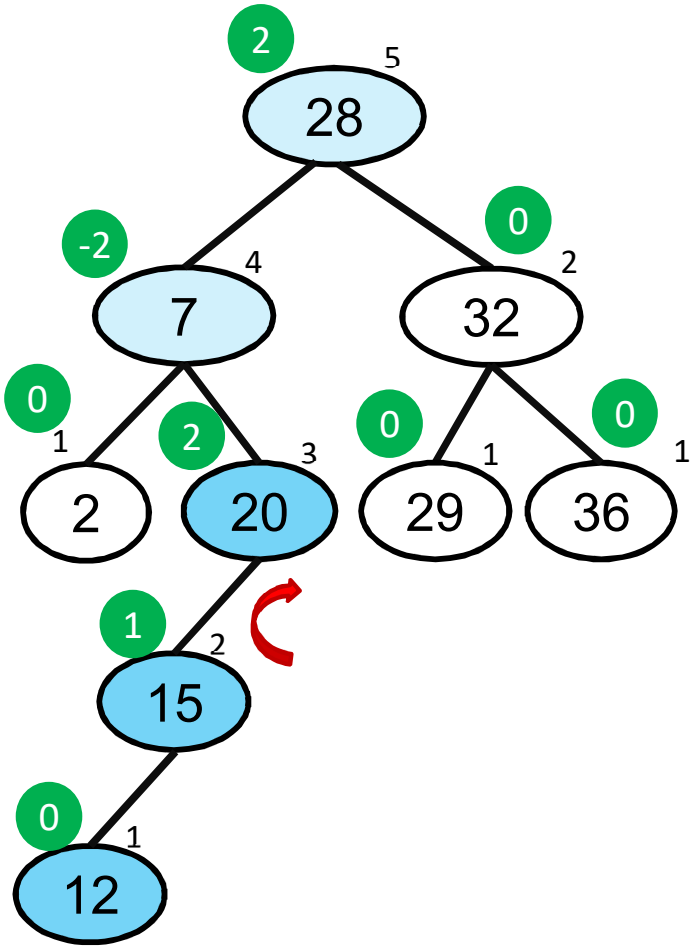


Α. Θεωρία

1. Δένδρο AVL

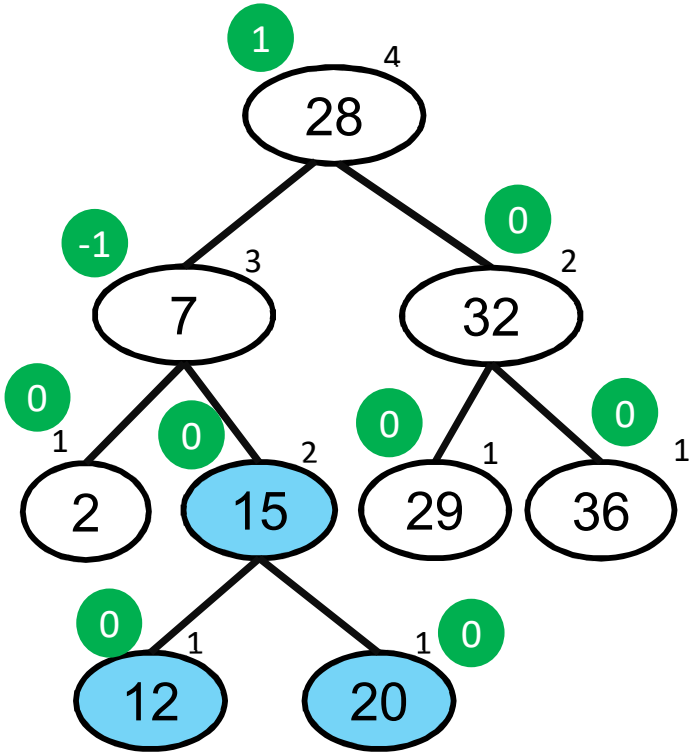
5. Εισαγωγή σε AVL – Παράδειγμα

Εισαγωγή «12»



Παράβαση LL

Περιστροφή «R»



Α. Θεωρία

1. Δένδρο AVL

6. Διαγραφή σε AVL

- Σημαντική Υπενθύμιση:
  - Το AVL δένδρο είναι ένα δένδρο ΔΔΑ
- Συνεπώς **η διαγραφή θα γίνει όπως στο ΔΔΑ.**
  - Αλλά θα εξεταστεί αν η διαγραφή οδήγησε σε ανισορροπία, οπότε και θα διορθωθεί με τις τεχνικές που μάθαμε.

- Οι περιπτώσεις που μπορεί να προκύψουν είναι ίδιες με τις περιπτώσης της διαγραφή σε ΔΔΑ:
  - Ο κόμβος δεν έχει παιδιά
  - Ο κόμβος έχει ένα παιδί
  - Ο κόμβος έχει δύο παιδιά. Βρίσκουμε τον επόμενο στην ενδοδιατεταγμένη διαδρομή:
    - Αν έχει μόνο δεξί παιδί
    - Αν έχει αριστερό παιδί

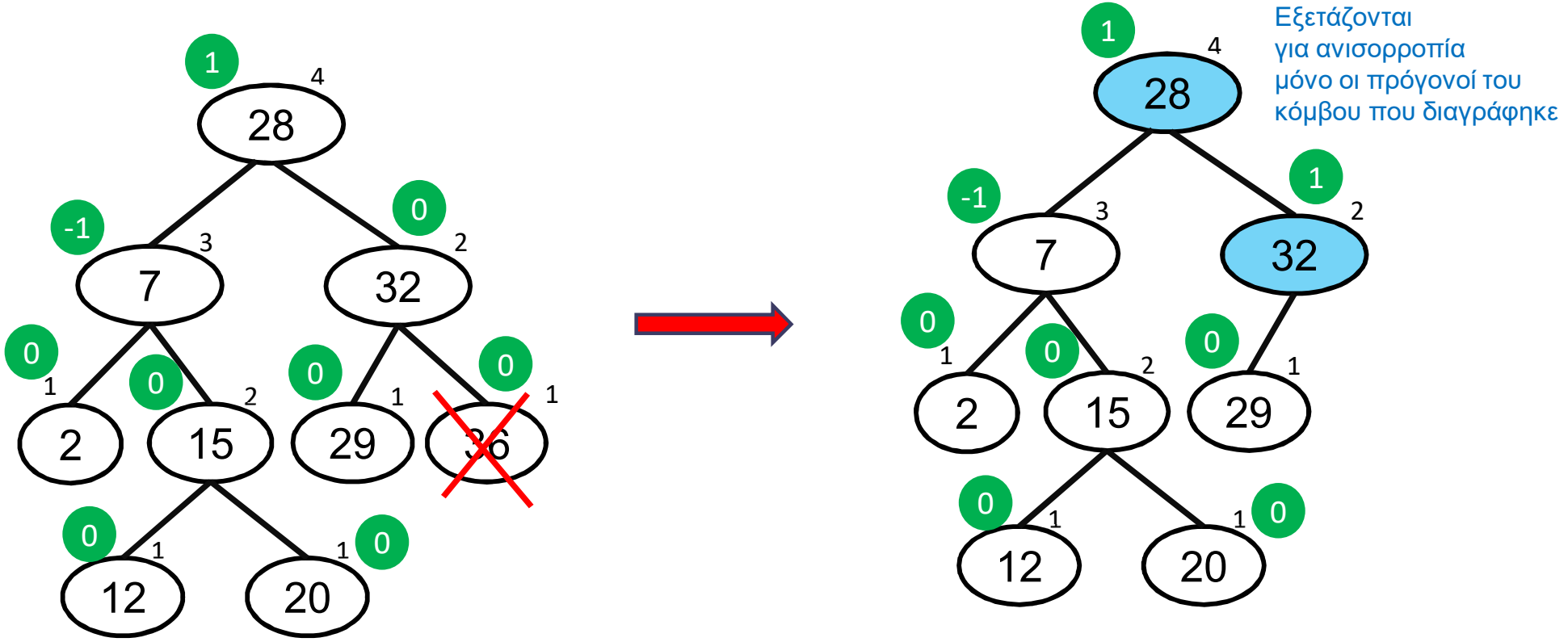
Α. Θεωρία

1. Δένδρο AVL

6.1. Διαγραφή σε AVL – Περίπτωση 1: Χωρίς Παιδιά

- **Αν ο κόμβος που διαγράφεται δεν έχει παιδιά**:
  - Διαγράφουμε τον κόμβο.
  - Διορθώνουμε το ύψος των κόμβων από τον πατέρα του μέχρι τη ρίζα.
  - Αν προκύψουν ανισορροπίες τις διορθώνουμε με τις γνωστές περιστροφές

- Παράδειγμα: Διαγραφή του 36



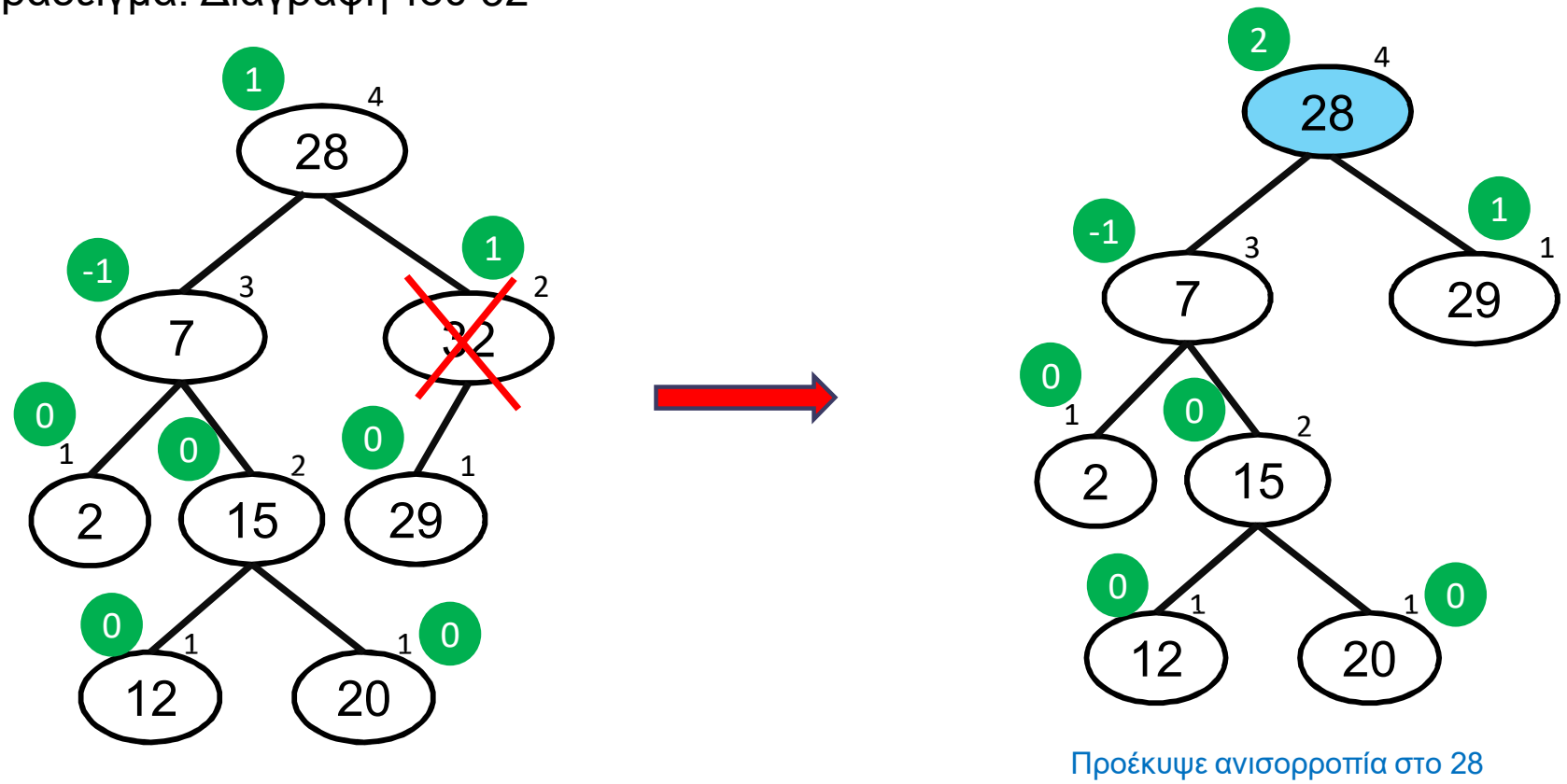
Α. Θεωρία

1. Δένδρο AVL

6.2. Διαγραφή σε AVL – Περίπτωση 2: Ένα Παιδί

- **Αν ο κόμβος που διαγράφεται έχει ένα παιδί**:
  - Διαγράφουμε τον κόμβο. Τον αντικαθιστούμε με το παιδί του.
  - Διορθώνουμε το ύψος των κόμβων από τον πατέρα του μέχρι τη ρίζα.
  - Αν προκύψουν ανισορροπίες τις διορθώνουμε με τις γνωστές περιστροφές.

- Παράδειγμα: Διαγραφή του 32



Προέκυψε ανισορροπία στο 28

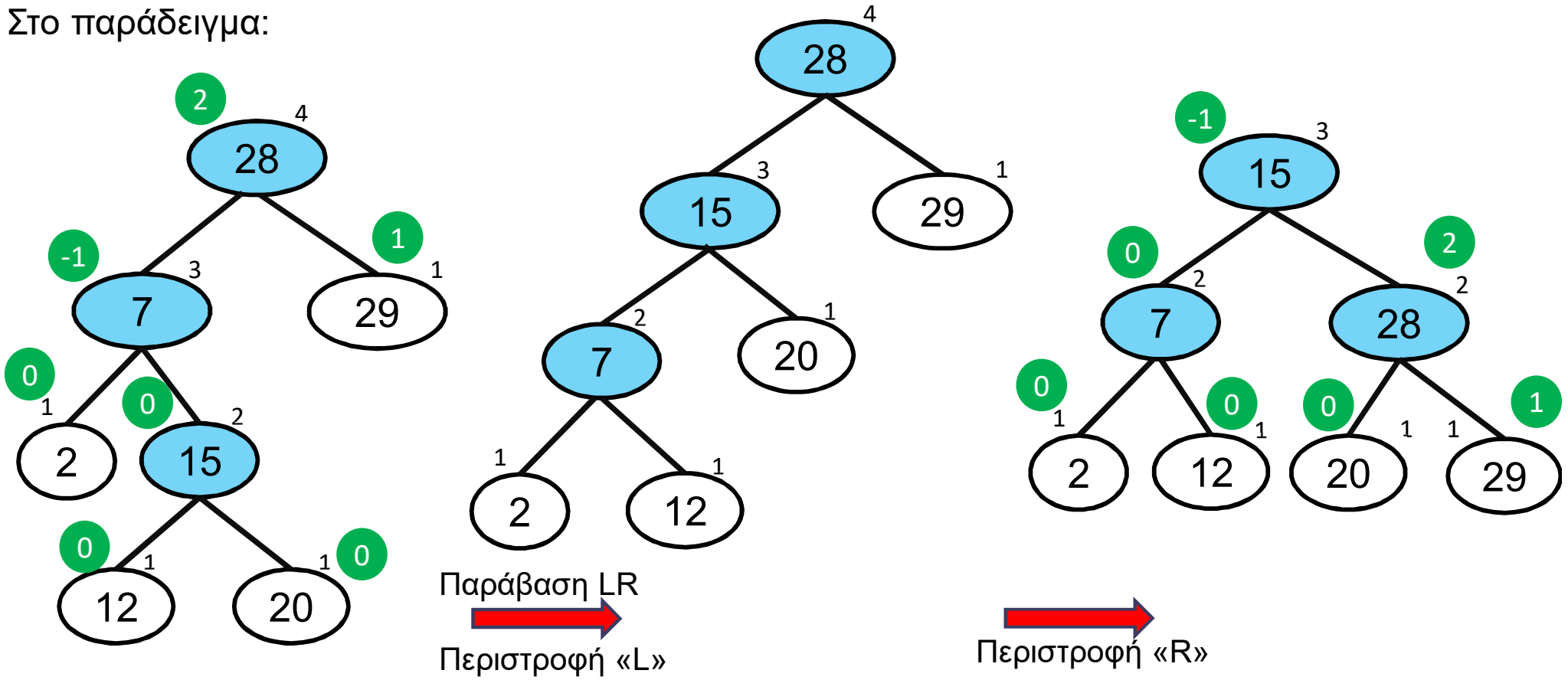
Α. Θεωρία

1. Δένδρο AVL

6.2. Διαγραφή σε AVL – Περίπτωση 2: Ένα Παιδί

- Η διόρθωση της ανισορροπίας γίνεται εξετάζοντας τα ύψη. Π.χ.:
  - Η διαφορά είναι 2 στον κόμβο 28, άρα η διόρθωση πρέπει να γίνει αριστερά (L)
  - Η διαφορά είναι -1 στον κόμβο 7, άρα η διόρθωση πρέπει να γίνει δεξιά (R)
  - Συνεπώς πρόκειται για ανισορροπία LR

- Στο παράδειγμα:



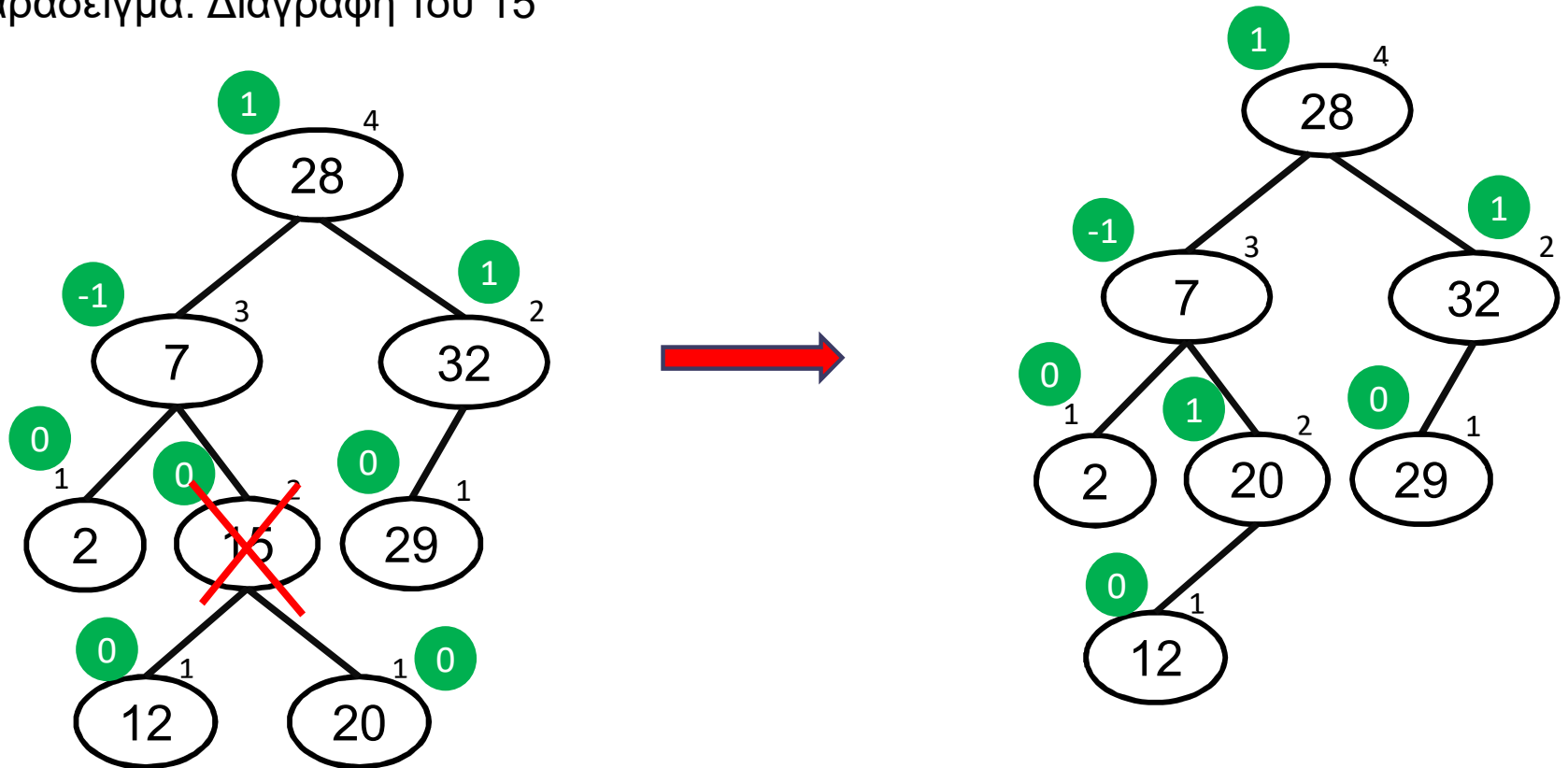
Α. Θεωρία

1. Δένδρο AVL

6.3. Διαγραφή σε AVL – Περίπτωση 3: Δύο παιδιά

- **Αν ο κόμβος που διαγράφεται έχει δυο παιδιά**:
- Υποπερίπτωση 1:
  - **Το δεξί παιδί του δεν έχει αριστερό παιδί**
    - Αντικαθιστούμε τον κόμβο με το δεξί του παιδί
- Αν προκύψουν ανισορροπίες τις διορθώνουμε με τις γνωστές περιστροφές

- Παράδειγμα: Διαγραφή του 15



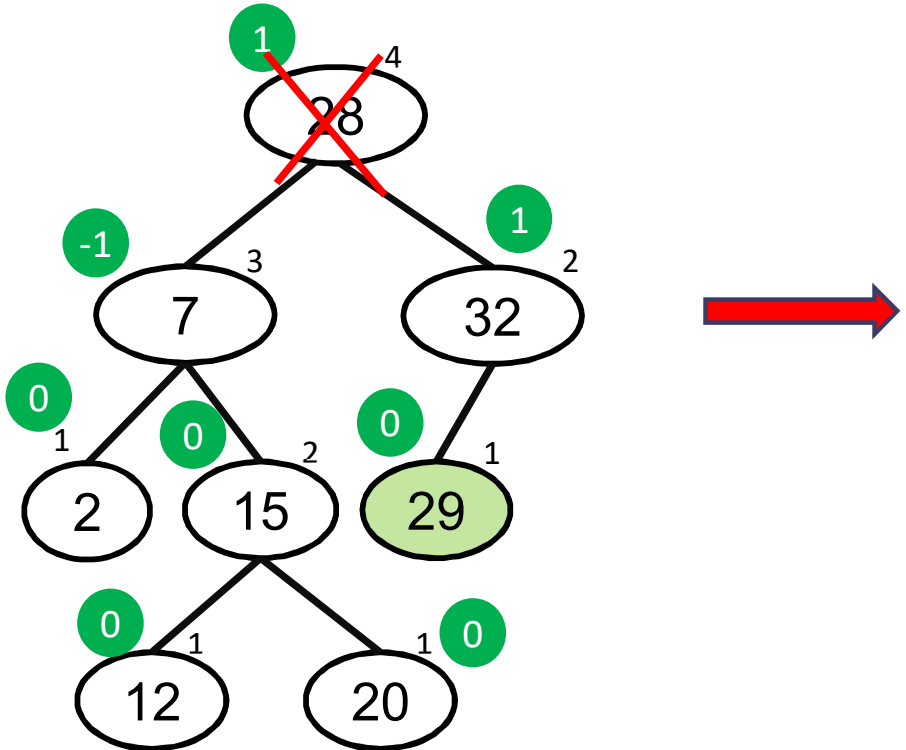
Α. Θεωρία

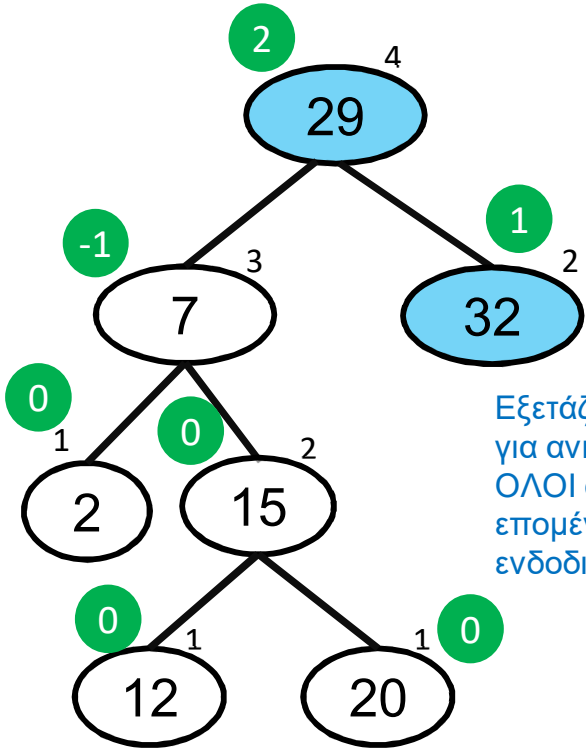
1. Δένδρο AVL

6.3. Διαγραφή σε AVL – Περίπτωση 3: Δύο παιδιά

- **Αν ο κόμβος που διαγράφεται έχει δυο παιδιά**:
- Υποπερίπτωση 2:
  - **Το δεξί παιδί του έχει αριστερό παιδί**
    - Αντικαθιστούμε τον κόμβο με τον επόμενο του στην ενδοδιαταγμένη διαδρομή
- Αν προκύψουν ανισορροπίες τις διορθώνουμε με τις γνωστές περιστροφές

- Παράδειγμα: Διαγραφή του 28





Εξετάζονται για ανισορροπία ΟΛΟΙ οι πρόγονοί του επομένου στην ενδοδιατεγμένη διαδρομή

Προέκυψε ανισορροπία στο 29, θα γίνει LR διόρθωση

Α. Θεωρία

2. Υλοποίηση σε C

1. Δηλώσεις

Οι **δηλώσεις** σε C είναι οι ακόλουθες:

- Το δένδρο AVL είναι μία δομή (struct) με τα εξής στοιχεία:
  - Ο κόμβος έχει τα ίδια στοιχεία με τον κόμβο του απλού δένδρου
  - Επαυξημένο με το ύψος του υποδένδρου στο οποίο είναι ρίζα

```
/* tree.h : Dilwseis dendrou AVL */

#define TRUE 1
#define FALSE 0

typedef int elem;              /* typos dedomenwn dendrou*/

struct node{                   /* Typos komvou listas */
   elem data;                  /* dedomena */
   struct node *left;          /* aristero paidi */
   struct node *right;         /* deksi paidi */
   int height;                 /* Ypsos toy ypodendroy */
};
typedef struct node TREE_NODE; /* Sinwnimo tou komvou dendrou */
typedef struct node *TREE_PTR; /* Sinwnimo tou deikti komvou */
```

Παρατήρηση: Ο ορισμός του κόμβου έχει επαυξηθεί, έτσι ώστε να ενσωματώνει το ύψος του υποδένδρου του οποίου ρίζα είναι ο κόμβος

Α. Θεωρία

2. Υλοποίηση σε C

2. Βασικές Πράξεις

Οι **βασικές πράξεις** σε ένα δένδρο AVL (επεκτείνοντας το δυαδικό δένδρο αναζήτησης) είναι:

- **Εισαγωγή** ενός στοιχείου στο δένδρο (**insert_AVL**)
- **Αναζήτηση** ενός στοιχείου στο δένδρο (**search_AVL**)
- **Διαγραφή** ενός στοιχείου από το δένδρο (**delete_AVL**)

Παρατήρηση: Επεκτείνουμε τον ορισμό του Δυαδικού Δένδρου Αναζήτησης (Μάθημα 7):

- Η αναζήτηση θα είναι η ίδια (search_BST)
- αλλά θα τροποποιηθεί ο κώδικας για την εισαγωγή και τη διαγραφή.

Α. Θεωρία

2. Υλοποίηση σε C

3. Εισαγωγή σε AVL

Ο αναδρομικός αλγόριθμος «Εισαγωγής σε AVL» δεδομένου ενός ΔΔΑ Τ και ενός δεδομένου x:

- **Εισάγει τον κόμβο σύμφωνα με τον αλγόριθμο εισαγωγής σε ΔΔΑ**
- **Μέσω των αναδρομικών κλήσεων ελέγχει για ανισορροπίες και τις διορθώνει**

Σκιαγράφηση αναδρομικού αλγορίθμου (ορίσματα: τρέχων κόμβος node, δεδομένα που θα εισαχθούν: data)

Αν node=NULL
τοποθέτησε το node ως ρίζα. Επέστρεψε το node
Αλλιώς αν data < node
Θέσε node->left = Αναδρομική Εισαγωγή των data στο node->left
Αν ο node είναι ανισόρροπος, διόρθωσέ τον.
Αλλιώς αν data > node
Θέσε node->right = Αναδρομική Εισαγωγή των data στο node->right
Αν ο node είναι ανισόρροπος, διόρθωσέ τον.
// αν data==node δεν κάνουμε τίποτα (δεν γίνεται εισαγωγή)

Α. Θεωρία

2. Υλοποίηση σε C

3. Εισαγωγή σε AVL

```
TREE_PTR AVL_rotate_R(TREE_PTR C)
{
        TREE_PTR L = C->left;
        C->left = L->right;
        L->right = C;
        C->height = max(height(C->left), height(C->right))+1;
        L->height = max(height(L->left), height(L->right))+1;
        return L;
}
```

Α. Θεωρία

2. Υλοποίηση σε C

3. Εισαγωγή σε AVL

```
TREE_PTR AVL_rotate_L(TREE_PTR C)
{
  TREE_PTR R = C->right;
  C->right = R->left;
  R->left = C;
  C->height = max(height(C->left), height(C->right))+1;
  R->height = max(height(R->left), height(R->right))+1;
  return R;
}
```

Α. Θεωρία

2. Υλοποίηση σε C

3. Εισαγωγή σε AVL

```
TREE_PTR AVL_rotate_RL(TREE_PTR C)
{
  C->right = AVL_rotate_R(C->right);

  return AVL_rotate_L(C);
}
```

Α. Θεωρία

2. Υλοποίηση σε C

3. Εισαγωγή σε AVL

```
TREE_PTR TR_insert_AVL(TREE_PTR *root, elem x)
{
        if (*root==NULL)
        {
                TR_insert_root(root, x);
                (*root)->height=1;
                return *root;
        }
        else if (x < (*root)->data)
        {
                (*root)->left=TR_insert_AVL(&((*root)->left), x);
                if (height((*root)->left)-height((*root)->right)==2)
                {
                        if(x < (*root)->left->data)
                                (*root)=AVL_rotate_R(*root);
                        else
                                (*root)=AVL_rotate_LR(*root);
                }
        }
```

Α. Θεωρία

2. Υλοποίηση σε C

3. Εισαγωγή σε AVL

```
        else if (x > (*root)->data)
        {
                (*root)->right=TR_insert_AVL(&((*root)->right), x);
                if (height((*root)->left)-height((*root)->right)==-2)
                {
                        if(x > (*root)->right->data)
                                (*root)=AVL_rotate_L(*root);
                        else
                                (*root)=AVL_rotate_RL(*root);
                }
        }
        // else x == root->x : den kanoyme tipota

        (*root)->height = max(height((*root)->left), height((*root)->right))+1;
        return (*root);
}
```

Α. Θεωρία

2. Υλοποίηση σε C

4. Διαγραφή από AVL

Ο αλγόριθμος «Διαγραφής από AVL» δεδομένου ενός ΔΔΑ Τ και ενός δεδομένου x:

- **Αναζητά το στοιχείο στο AVL**
- **Διαγράφει το στοιχείο και κάνει τις απαραίτητες διορθώσεις στο AVL**

Σκιαγράφηση επαναληπτικού αλγορίθμου (ορίσματα: ΔΔΑ Τ, δεδομένα που θα εισαχθούν: data)

Βήμα 1:
Βρες το μονοπάτι που οδηγεί μέχρι το κόμβο (πατέρας και κατεύθυνση παιδιού (left ή right)
Αν ο κόμβος έχει δύο παιδιά, κράτα στο μονοπάτι και την ακολουθία κόμβων μέχρι τον επόμενο στην ενδοδιατεταγμενη διαδρομη

Βήμα 2:
Διέγραψε τον κόμβο (με την ρουτίνα διαγραφής από ΔΔΑ)

Βήμα 3:
Αντιστρόφως, για κάθε προηγούμενο στο μονοπάτι:
Δες αν υπάρχει ανισορροπία,
Διόρθωσε την ανισορροπία με περιστροφές

Α. Θεωρία

2. Υλοποίηση σε C

4. Διαγραφή από AVL

```
int TR_delete_AVL(TREE_PTR *root, elem x)
{
        TREE_PTR pred[DEPTH], new_root;
        int orient[DEPTH]; //0-left, 1-right
        TREE_PTR current;
        int i=0,j;


        if (*root==NULL) //adeio dentro
                return FALSE;

        /* Vres olous tous progonous */
        /* Apo ti riza mexri ton komvo */
        current=*root;
```

Α. Θεωρία

2. Υλοποίηση σε C

4. Διαγραφή από AVL

```
while(current!=NULL)
{
        if (x == current->data)
                break;
        else if (x < current->data)
        {
                orient[i]=0;
                pred[i++]=current;
                current=current->left;
        }
        else // x > current->data
        {
                orient[i]=1;
                pred[i++]=current;
                current=current->right;
        }
}
```

Α. Θεωρία

2. Υλοποίηση σε C

4. Διαγραφή από AVL

```
if (current==NULL) //den vrethike
        return FALSE;

if (current->left!=NULL && current->right!=NULL)
{
        orient[i]=1;
        pred[i++]=current;
        current=current->right;

        while (current->left!=NULL)
        {
                orient[i]=0;
                pred[i++]=current;
                current=current->left;
        }
}

TR_delete_BST(root, x);
```

Α. Θεωρία

2. Υλοποίηση σε C

4. Διαγραφή από AVL

```
for (j=i-1; j>=0; j--)
{
        if (height(pred[j]->left)-height(pred[j]->right)==2)
        {
                if(height(pred[j]->left->left)>height(pred[j]->left->right))
                        new_root=AVL_rotate_R(pred[j]);
                else
                        new_root=AVL_rotate_LR(pred[j]);

        }
        else if (height(pred[j]->left)-height(pred[j]->right)==-2)
        {
                if(height(pred[j]->right->right)>height(pred[j]->right->left))
                        new_root=AVL_rotate_L(pred[j]);
                else
                        new_root=AVL_rotate_RL(pred[j]);
        }
        else
                continue;
```

Α. Θεωρία

2. Υλοποίηση σε C

4. Διαγραφή από AVL

```
                        if (j==0)
                                *root = new_root;
                        else
                                if (orient[i-1]==0)
                                        pred[i-1]->left=new_root;
                                else
                                        pred[i-1]->right=new_root;
                }
}
```

Α. Θεωρία

2. Υλοποίηση σε C

5. Αναζήτηση σε AVL

Η συνάρτηση «Αναζήτηση» ψάχνει για το στοιχείο Χ στο δυαδικό δένδρο αναζήτησης και επιστρέφει ΝΑΙ/ΟΧΙ ανάλογα με το αν το στοιχείο υπάρχει στο δένδρο:

Σημείωση: Είναι ίδια με την αναζήτηση σε ΔΔΑ

Σκιαγράφηση αλγορίθμου:

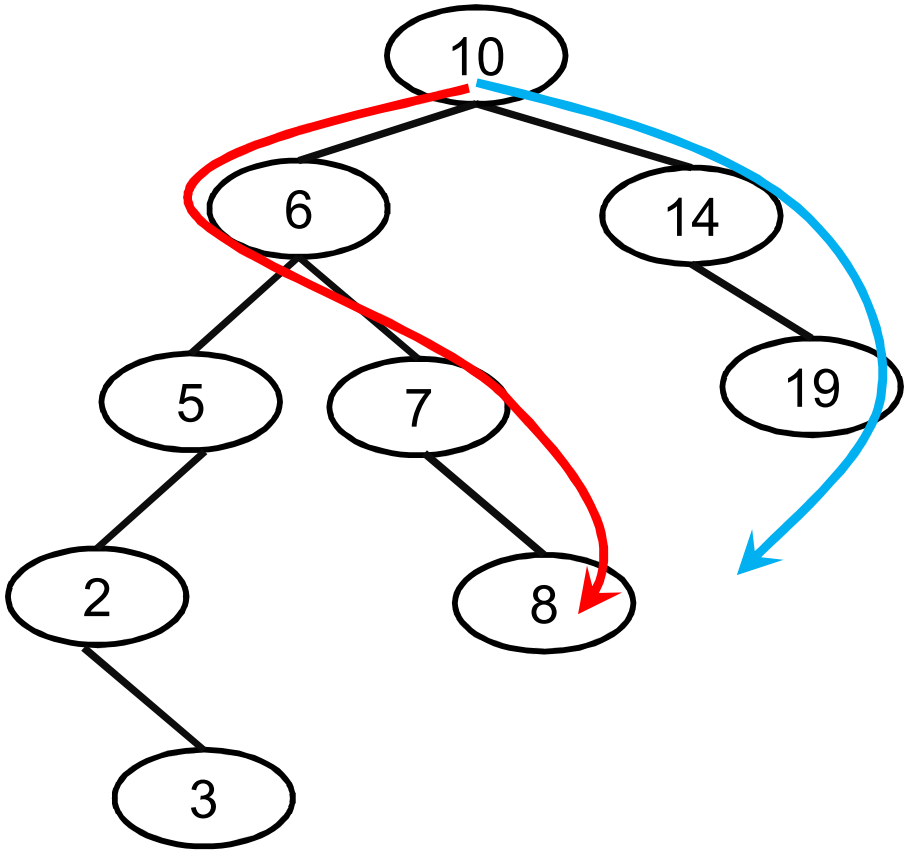
```
Θέτει Κ = ρίζα του δένδρου
Επανέλαβε όσο  Κ ≠ ΚΕΝΟ
    Αν (Χ = Κ)
          Επέστρεψε ΝΑΙ
    Αλλιώς αν (Χ > Κ)
          Θέσε Κ=δεξί παιδί της Κ.
    Αλλιώς αν (Χ < Κ)
          Θέσε Κ=αριστερό παιδί της Κ.
Τέλος-Επανάληψης
Επέστρεψε ΟΧΙ
```

Α. Θεωρία

2. Υλοποίηση σε C

5. Αναζήτηση σε AVL

- Παράδειγμα Αναζήτησης του δεδομένου 8 (κόκκινο χρώμα) και του 17 (μπλέ χρώμα)



- Αναζήτηση του 8: 10(αριστερά), 6(δεξιά), 7(δεξιά), 8 (βρέθηκε). Απάντηση: ΝΑΙ
- Αναζήτηση του 17: 10(δεξιά), 14 (δεξιά), 19(αριστερά). ΚΕΝΟ. Απάντηση: ΟΧΙ

Α. Θεωρία

2. Υλοποίηση σε C

5. Αναζήτηση σε AVL

```
/* TR_search_AVL(): anazitisi tou x sto
         DDA me riza root   */
int TR_search_AVL(TREE_PTR root, elem x)
{
   TREE_PTR current;

   current=root;

   while (current!=NULL)
   {
      if (x == current->data)
         return TRUE;
      else if (x < current->data)
         current=current->left;
      else // x > current->data
         current=current->right;
   }
   return FALSE;
}
```

Β. Ασκήσεις

Εφαρμογή 1: Μελέτη Προγράμματος

- Μελετήστε το project tree.dev στο οποίο υλοποιούνται οι βασικές πράξεις των δένδρων AVL που μελετήσαμε στο μάθημα.

Β. Ασκήσεις

Εφαρμογή 2: Εκτέλεση «με το χέρι»

1. Εισάγετε τα ακόλουθα δεδομένα σε ένα AVL: «10 8 15 7 3 9 2 11».

2. Διαγράψτε τον κόμβο 10

3. Εισάγετε τους κόμβους 22 και 29

Β. Ασκήσεις

Εφαρμογή 2: Εκτέλεση «με το χέρι»

4. Δώστε την ενδοδιατεταγμένη διαπέραση του δένδρου

5. Δώστε την προδιατεταγμένη διαπέραση του δένδρου

6. Δώστε τη μεταδιατεταγμένη διαπέραση του δένδρου

7. Κατασκεύαστε πλήρες δυαδικό δένδρο αναζήτησης που να περιέχει τα περιεχόμενα του δένδρου .