# Ενότητες 3 & 4: Δένδρα, Σύνολα & Λεξικά

# Ασκήσεις και Λύσεις

**Άσκηση 1**

Γράψτε μία αναδρομική συνάρτηση που θα παίρνει ως παράμετρο ένα δείκτη στη ρίζα ενός δυαδικού δένδρου και θα επιστρέφει το βαθμό του διατεταγμένου δένδρου (δηλαδή το μέγιστο πλήθος παιδιών μεταξύ των κόμβων του δένδρου). Ποιά είναι η χρονική πολυπλοκότητα του αλγορίθμου σας και γιατί;

**Λύση**

```
int TreeDegree (struct node * r) {    // εκτελεί μεταδιατεταγμένη διάσχιση του δένδρου
      int ld , rd , deg = 0;

      if (r == NULL) return 0;
      ld = TreeDegree (r→lc );    // υπολόγισε το μέγιστο βαθμό στο αριστερό υποδένδρο
      rd = TreeDegree (r→rc );    // υπολόγισε το μέγιστο βαθμό στο δεξιό υποδένδρο
Visit if (r→lc != NULL) deg++; // υπολόγισε το βαθμό του r, ο οποίος θα είναι 0, 1 ή 2
      if (r→rc != NULL) deg++;
      return max{deg , ld , rd };  // επέστρεψε το μέγιστο μεταξύ του βαθμού του r και
                                   // των βαθμών ld και rd του αριστερού και δεξιού υποδένδρου,
                                   // αντίστοιχα
}
```

**Χρονική Πολυπλοκότητα:** Η TreeDegree() εκτελεί διάσχιση στο δένδρο και για κάθε κόμβο υπολογίζει πόσα παιδιά έχει ο κόμβος. Άρα, η χρονική πολυπλοκότητα είναι O(n), όπου n είναι το πλήθος των κόμβων του δένδρου.

**Χωρική Πολυπλοκότητα:** Για κάθε αναδρομική κλήση, δημιουργείται στη μνήμη ένα activation record για την αποθήκευση των παραμέτρων και των τοπικών μεταβλητών του στιγμιότυπου της TreeDegree() που ενεργοποιείται με την κλήση. Κάθε χρονική στιγμή έχουμε O(h) κλήσεις της TreeDegree() ταυτόχρονα ενεργές, όπου h είναι το ύψος του δένδρου. Επομένως, η μνήμη που απαιτείται για να τρέξει η TreeDegree() (επιπρόσθετα της μνήμης για την αποθήκευση του δένδρου) είναι O(h).

**Άσκηση 2**

Γράψτε μία αναδρομική συνάρτηση, η οποία θα έχει, ως παράμετρο, ένα δείκτη στη ρίζα ενός δυαδικού δένδρου **που αναπαριστά ένα (όχι απαραίτητα δυαδικό) διατεταγμένο δένδρο** και θα επιστρέφει το βαθμό του διατεταγμένου δένδρου (δηλαδή το μέγιστο πλήθος παιδιών μεταξύ των κόμβων του δένδρου). Αν ο βαθμός του διατεταγμένου δένδρου είναι k, η συνάρτηση

πρέπει να εκτελείται σε O(nk), όπου n είναι το πλήθος των κόμβων του δένδρου. Εξηγήστε ποιες είναι οι διαφορές του αλγορίθμου που σχεδιάσατε συγκριτικά με τον αλγόριθμο που σχεδιάσατε στην Άσκηση 1, καθώς και γιατί αυτές οι αλλαγές είναι απαραίτητες. Επιχειρηματολογήστε για την χρονική πολυπλοκότητα του αλγορίθμου σας.

**Λύση**

```
int TreeDegree(struct node * r) {
        int ld, rd, deg = 0;

        if (r == NULL) return 0;
        ld = TreeDegree(r→lc);  // αναδρομικές κλήσεις για διάσχιση του δένδρου
        rd = TreeDegree(r→rc);
        p = r→lc;               // ξεκινώντας από το αριστερότερο παιδί του κόμβου r
        while (p != NULL) {     // διασχίζουμε όλα τα παιδιά του r
Visit       deg++;              // και μετράμε πόσα είναι στην μεταβλητή deg
            p = p→rs;           // έτσι η deg στο τέλος της while περιέχει το βαθμό του r
        }
        return max{deg, ld, rd}; // επιστρέφουμε το μέγιστο μεταξύ του βαθμού του r και
                                 // των βαθμών του αριστερού και του δεξιού του υποδένδρου
}
```

Το μόνο που αλλάζει είναι ηVisit(). Η προσπέλαση των παιδιών ενός κόμβου, για να μπορέσουμε να τα μετρήσουμε και να υπολογίσουμε το βαθμό του, γίνεται προσπελάζοντας πρώτα το αριστερότερο παιδί του κόμβου και στη συνέχεια τη λίστα των αδελφικών κόμβων αυτού του παιδιού προκειμένου να βρω τα υπόλοιπα παιδιά του κόμβου.

Η TreeDegree() εκτελεί μια μεταδιατεταγμένη διάσχιση των κόμβων. Για κάθε κόμβο v που επισκεπτόμαστε, το κόστος για να εκτελέσουμε των κώδικα Visit είναι O(k), όπου k είναι ο βαθμός του κόμβου του διατεταγμένου δένδρου που αναπαρίσταται από τον v. Άρα, η συνολική πολυπλοκότητα της TreeDegree() είναι O(nk).

## Άσκηση 3

Περιγράψτε αλγόριθμο, ο οποίος θα παίρνει, ως όρισμα, έναν δείκτη p σε έναν κόμβο ενός διπλά-συνδεδεμένου δυαδικού δένδρου και θα επιστρέφει τον προηγούμενο του κόμβου στην προδιατεταγμένη διάσχιση.

**Λύση:**

```
struct node *PreOrderPredecessor ( struct node * p) {
        struct node *q, *prevq = NULL;

        if (p→parent == NULL){
                print("There is only one node in the tree");
                return NULL;
        }
        if (p == p→parent→lc)  // αν είμαι αριστερό παιδί του πατέρα μου
```

```
            return p→parent;  // ο προηγούμενος στην προδιατεταγμένη είναι ο πατέρας μου

        // ακολουθεί κώδικας για την περίπτωση που ο κόμβος
        // στον οποίο δείχνει ο p είναι δεξί παιδί του πατέρα του
        q = p→parent→lc;        // ο προηγούμενος βρίσκεται στο αριστερό υποδένδρο του πατέρα
        prevq = p→parent;       // και είναι το δεξιότερο φύλλο σε αυτό το υποδένδρο
        while (q != NULL) {     // επαναληπτικά
                prevq = q;
                q→rc;           // κατεβαίνω προς τα δεξιά
                if (q == NULL) q= prevq→lc; // μέχρι να βγω σε NULL
                                            // αν ο τελευταίος κόμβος που διέσχισα
                                            // έχει αριστερό παιδί, συνεχίζω την ίδια
                                            // διαδικασία στο υποδένδρο του παιδιού αυτού
        }       // όταν βγαίνω από τη while, ο q είναι NULL και ο prevq δείχνει στο δεξιότερο φύλλο
                // του αριστερού υποδένδρου του πατέρα του p
        return prevq;           // ο prevq είναι επομένως ο προηγούμενος του p
                                // στην προδιατεταγμένη διάσχιση και τον επιστρέφω
}
```

**<u>Άσκηση 4 (Ταξινομημένα Δυαδικά Δένδρα)</u>**

1. Ας υποθέσουμε ότι ένα ταξινομημένο δυαδικό δένδρο αποτελείται από τα κλειδιά 1 έως 1000. Έστω ότι αναζητείται το κλειδί 363. Εξετάστε αν οι παρακάτω ακολουθίες κλειδιών είναι έγκυρες ακολουθίες κόμβων που θα μπορούσαν να προσπελάζονται από το γνωστό αλγόριθμο αναζήτησης σε ταξινομημένα δυαδικά δένδρα.

   α. 102, 352, 501, 498, 430, 444, 497, 463, 363

   β. 974, 270, 961 ,299, 948, 298, 412, 413, 363

   γ. 935, 278, 347, 621, 299, 392, 358, 363

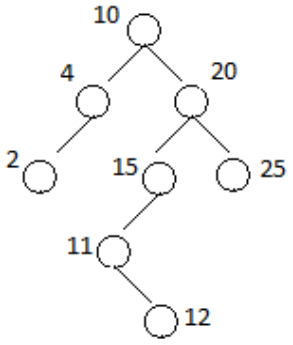
2. Ζωγραφίστε 10 ταξινομημένα δένδρα ύψους 5 με κλειδιά 3, 7, 9, 14, 19, 22, 26.

3. Εισάγετε τα κλειδιά 5, 1, 18 στο δένδρο του Σχήματος 1.

4. Παρουσιάστε τα δένδρα που προκύπτουν κατά τη διαγραφή των κλειδιών 10, 20, 1 και 15 από το δένδρο που προκύπτει μετά τις εισαγωγές του ερωτήματος 3.

**Λύση**

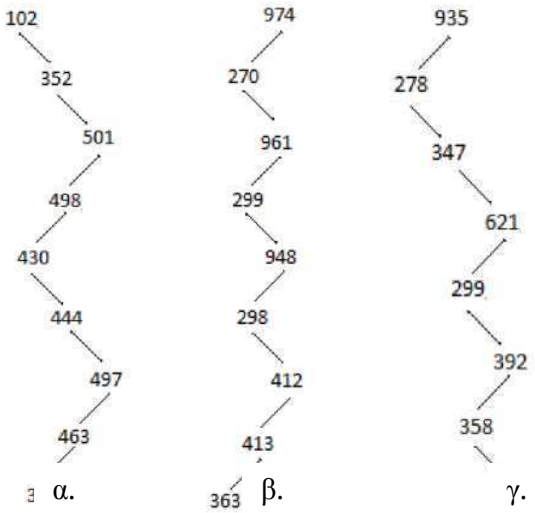
1. α. Το μονοπάτι που θα είχε ακολουθήσει ο αλγόριθμος αναζήτησης προκειμένου να επισκεφτεί τους κόμβους της ακολουθίας α. φαίνεται στο Σχήμα 2α. Παρατηρούμε πως από το κλειδί 430, η αναζήτηση θα έπρεπε να συνεχίσει σε μικρότερα κλειδιά και όχι σε μεγαλύτερα, όπως έγινε. Παρατηρήστε ότι στο Σχήμα 2α. το κλειδί 363 είναι στα δεξιά του 430 (που καταστρατηγεί την ταξινόμηση του δένδρου). Επομένως, η ακολουθία 1α. δεν είναι έγκυρη.



Σχήμα 1

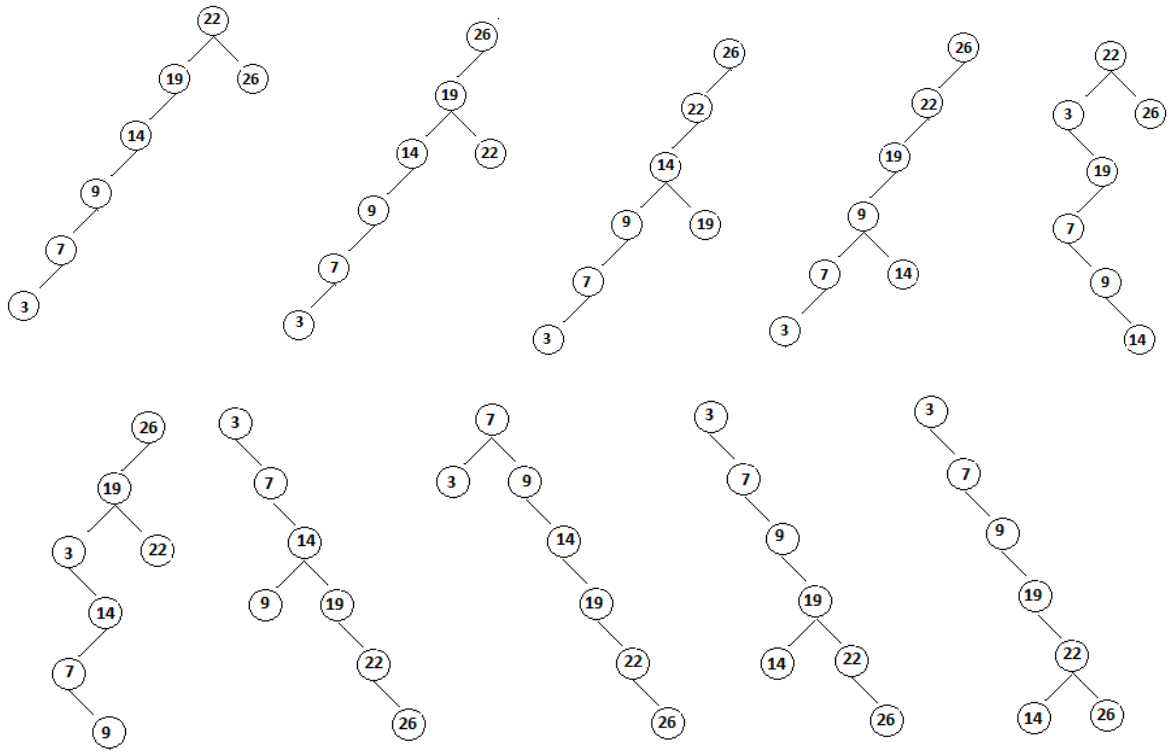
β. Το μονοπάτι που θα είχε ακολουθήσει ο αλγόριθμος αναζήτησης προκειμένου να επισκεφτεί τους κόμβους της ακολουθίας β. φαίνεται στο Σχήμα 1β. Παρατηρούμε πως το κλειδί 298 βρίσκεται στα δεξιά του 299. Αυτό καταστρατηγεί την ταξινόμηση του δένδρου). Επομένως, η ακολουθία 1β. επίσης δεν είναι έγκυρη.

γ. Το μονοπάτι που θα είχε ακολουθήσει ο αλγόριθμος αναζήτησης προκειμένου να επισκεφτεί τους κόμβους της ακολουθίας γ. φαίνεται στο Σχήμα 2γ. Παρατηρούμε ότι το κλειδί 299 βρίσκεται στα δεξιά του 347 το οποίο καταστρατηγεί την ιδιότητα ταξινόμησης του δένδρου. Επομένως, η ακολουθία 1γ. δεν είναι έγκυρη.

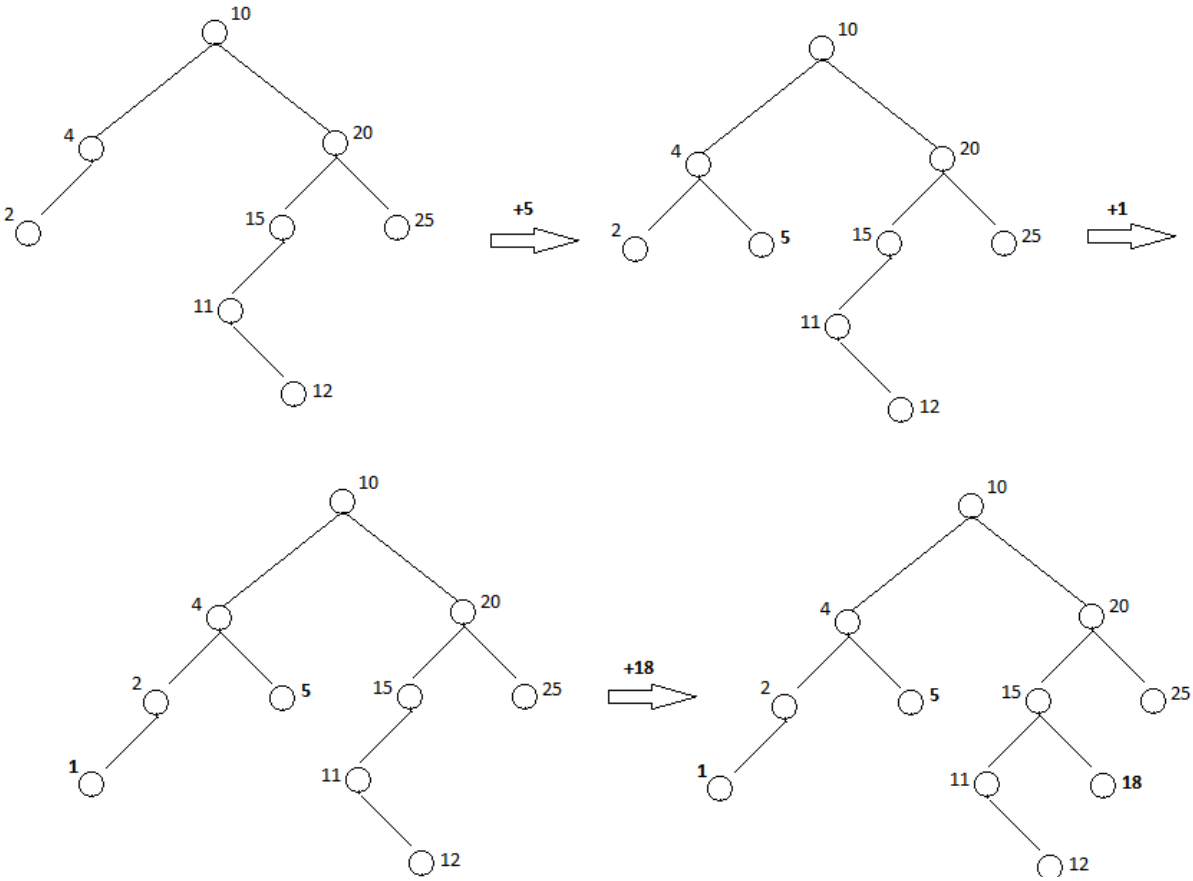



α. β. γ.

2. Τα δένδρα παρουσιάζονται στο Σχήμα 3.



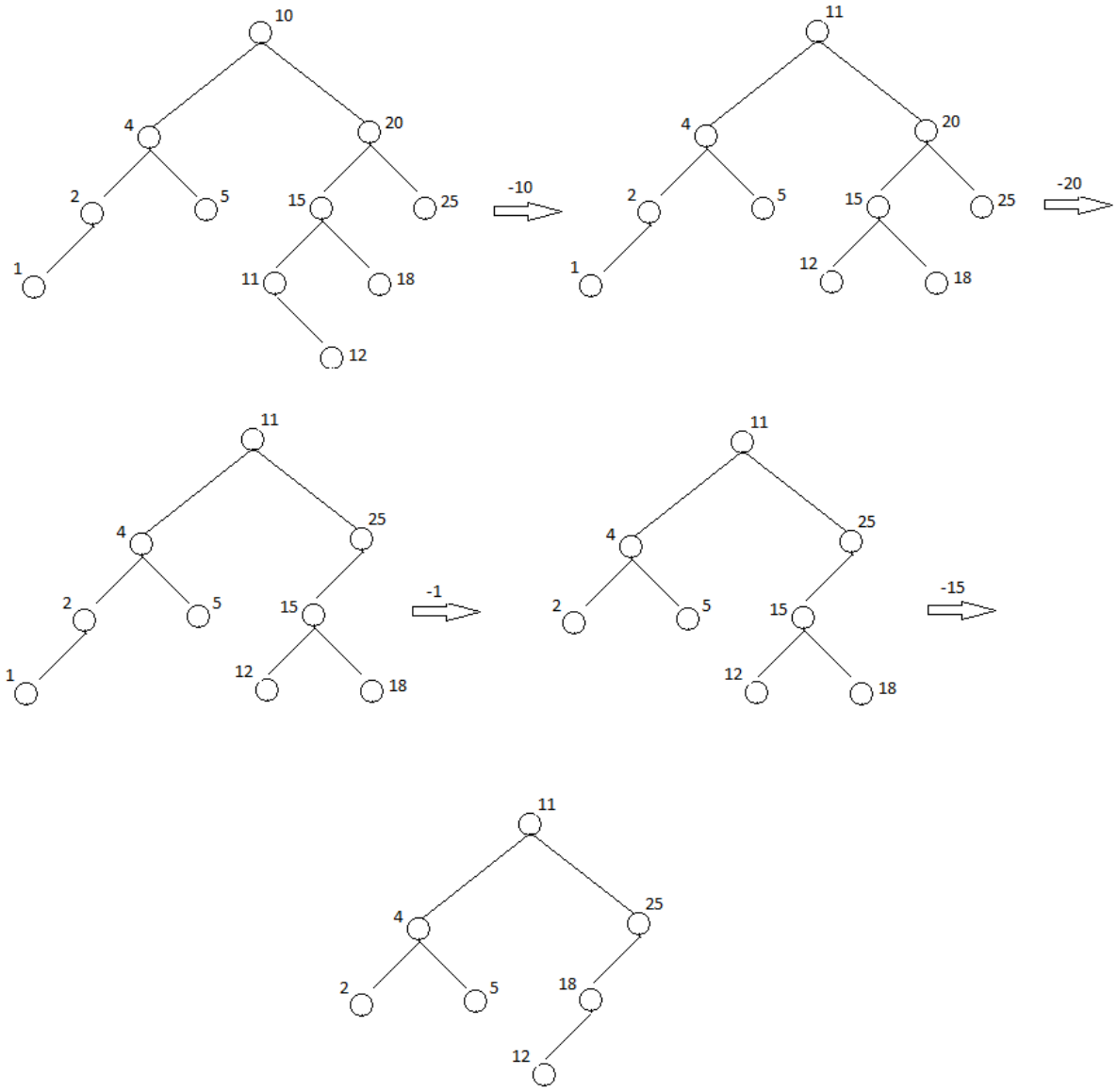
Σχήμα 3




3. Η διαδοχική εισαγωγή των κλειδιών παρουσιάζεται στο Σχήμα 4.

Σχήμα 4

4. Η διαδοχική διαγραφή των κλειδιών 10, 20, 1 και 15 αναπαρίσταται στο σχήμα 5.


10
4
20
2
5
15
25
1
11
18
12
-10
11
4
20
2
5
15
25
1
12
18
-20
11
4
25
2
5
15
1
12
18
-1
11
4
25
2
5
15
12
18
-15
11
4
25
2
5
18
12

Σχήμα 5