Φροντιστήριο 6 – Σκελετοί Λύσεων

Άσκηση 1

Για υλοποίηση της επιπλέον διαδικασίας (υπολογισμού της σειράς στοιχείων του δένδρου) σε χρόνο γραμμικό ως προς το ύψος του δένδρου θα πρέπει ανά πάσα στιγμή να έχουμε για κάθε στοιχείο, x, πληροφορίες για το πόσα στοιχεία μικρότερα ή ίσα του x βρίσκονται μέσα στο δένδρο. Συγκεκριμένα σε κάθε κόμβο διατηρούμε μια μεταβλητή greater που μετρά τον αριθμό των στοιχείων του αριστερού υποδένδρου του κόμβου + 1. Ένας κόμβος υλοποιείται ως την πιο κάτω εγγραφή:

```
struct Node {
   int         element;
   struct Node *left;
   struct Node *right;
   int         greater;
}
```

Οι διαδικασίες ενός ΔΔΑ πρέπει να επεκταθούν έτσι ώστε να διατηρείται η προδιαγραφή για το πεδίο greater.

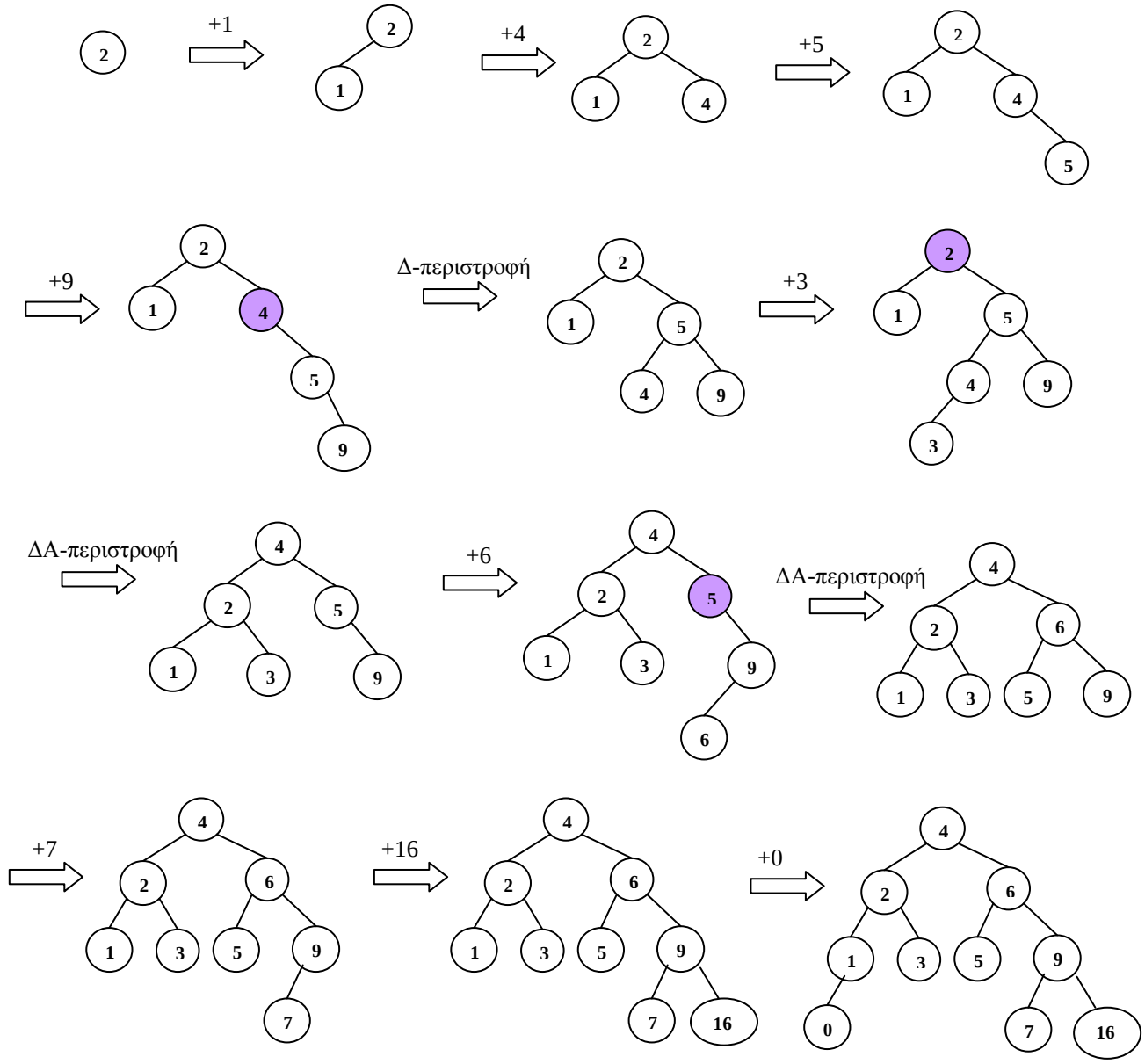
Ακολουθεί η διαδικασία Select(i,p) η οποία βρίσκει το i-οστό στοιχείο μέσα στο δένδρο p (όπου p είναι τύπου *Node, δηλαδή είναι δείκτης στη ρίζα του δένδρου).
Η κύρια ιδέα είναι η εξής , αν ο κόμβος x είναι μεγαλύτερος ή ίσος από i στοιχεία τότε είναι το i-οστό στοιχείο του δένδρου, άρα επιστρέφουμε το στοιχείο x.element. Διαφορετικά, (1) αν είναι μεγαλύτερος ή ίσος από λιγότερα των i στοιχείων, έστω j στοιχεία, τότε πρέπει να βρούμε το (i-j)-οστό μεγαλύτερο στοιχείο του δεξιού υποδένδρου του κόμβου x, και, τέλος, (2) αν είναι μεγαλύτερος ή ίσος από περισσότερα από i (έστω j), τότε πρέπει να βρούμε το i-οστό στοιχείο του αριστερού υποδένδρου.

```
int Select(int i, Node *p){

        Node *q = p;

        while (i != q->greater && q!= null){
             if (q->greater < i)
                  i = i - q->greater;
                  q = q->right
             if (q->greater > i)
                  q = q->left
        }
        if (q==null)
             return error
        else
             return q->element;
}
```

Προφανώς όλες οι διαδικασίες μπορούν να πραγματοποιηθούν με μια μόνο διέλευση από τη ρίζα μέχρι, το πολύ, το κατάλληλο φύλλο. Άρα η πολυπλοκότητα τους είναι της τάξης O(h).

Οι διαδικασίες εισαγωγής και εξαγωγής στοιχείου αφήνονται ως ασκήσεις για τον αναγνώστη.

Άσκηση 2



Άσκηση 3

Χρησιμοποιούμε την εγγραφή

```
typedef struct AVL-Node{
      int height;
      int key;
      struct AVL-Node *left;
      struct AVL-Node *right;
} avl-node;
```

και υποθέτουμε πως ένα AVL δένδρο είναι υλοποιημένο ως δείκτης στη ρίζα του δένδρου, δηλαδή, έχει τύπο *avl-node.

(i) Για να υπολογίσουμε το πλήθος των φύλλων ενός AVL δένδρου που δείχνεται από κάποιο δείκτη p ξεχωρίζουμε τρεις περιπτώσεις: Αν το δένδρο είναι κενό, το πλήθος των φύλλων του είναι 0. Αν το δένδρο αποτελείται μόνο από μια ρίζα, το πλήθος των φύλλων του είναι 1. Διαφορετικά, το πλήθος των φύλλων του δένδρου είναι ίσο με το άθροισμα των φύλλων του αριστερού υποδένδρου και των φύλλων του δεξιού υποδένδρου

Αυτή η ιδέα μεταφράζεται στην πιο κάτω αναδρομική διαδικασία:

```
int RecCountLeaves(avl-node *p){
     if (p == NULL)
        return 0;
     if (p->left == NULL AND p->right == NULL)
          return 1;
     else
          x = RecCountLeaves(p->left)
               + RecCountLeaves(p->right);
                  return x;
}
```

Η αναδρομική διαδικασία καλείται μια φορά σε κάθε κόμβο, επομένως ο χρόνος εκτέλεσής της είναι O(n) όπου n είναι ο αριθμός των κόμβων του δένδρου.

(ii) Η μη-αναδρομική διαδικασία απαιτεί τη χρήση στοίβας και έχει ως εξής:

I. Θέτουμε counter = 0 και p ίσο με το δείκτη στη ρίζα του δένδρου.
II. Εφόσον ο δείκτης δεν είναι NULL προχωρούμε στο βήμα ΙΙ.
III Αν κόμβος είναι φύλλο αυξάνουμε την τιμή του counter κατά 1 και
- Αν η στοίβα δεν είναι κενή συνεχίζουμε με τον κόμβο κορυφής της στοίβας και επαναλαμβάνουμε το βήμα.
- Διαφορετικά επιστρέφουμε την τιμή του counter.

IV. Αν ο κόμβος δεν είναι φύλλο τότε προχωρούμε σε ένα από τα παιδιά του κόμβου, και αν υπάρχει και δεύτερο παιδί το τοποθετούμε στη στοίβα, επαναλαμβάνοντας από το βήμα ΙΙΙ.

```
int CountLeaves(avl-node *p){

stack S;
counter = 0;

MakeEmpty(S);

while (p != NULL){
        if (p->left == NULL AND p->right == NULL)
             counter++;
             if (!IsEmpty(S))
                p = Pop(S);
             else
                return counter;
        else
             if (p->right != NULL)
                  Push(p->right, S);
             if (p->left != NULL)
                p = p->left;
            else
                p = Pop(S);
  }
 return counter;
}
```