



Διάλεξη 22: Δυαδικά Δέντρα

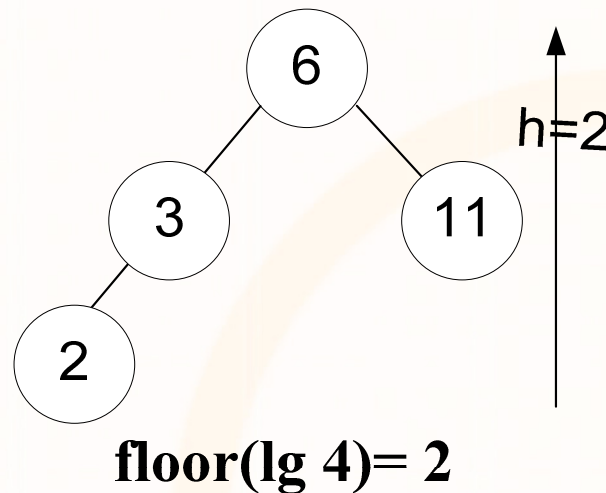
Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Δυαδικά Δένδρα
- Δυαδικά Δένδρα Αναζήτησης
- Πράξεις Εισαγωγής, Εύρεσης Στοιχείου, Διαγραφής Μικρότερου Στοιχείου

Διδάσκων: Παναγιώτης Ανδρέου

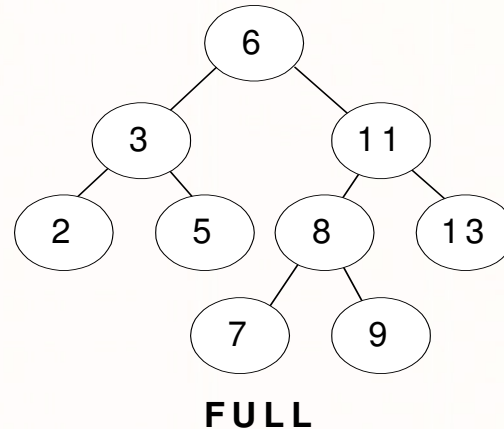
Δυαδικά Δένδρα

- Ένα δένδρο είναι **δυαδικό** αν όλοι οι κόμβοι του έχουν **βαθμό** ≤ 2 .
Ορισμός: Δυαδικό δένδρο λέγεται ένα δένδρο το οποίο :
 - είτε είναι κενό,
 - ή αποτελείται από μια **ρίζα** και **δύο δυαδικά υπόδενδρα**.
Αναφερόμαστε στα δύο υποδένδρα ως το **αριστερό** και το **δεξιό υπόδενδρο**.
- Το ύψος ενός δυαδικού δένδρου με n κόμβους μπορεί να είναι το πολύ : **$n-1$ (συνδεδεμένη λίστα)** και το λιγότερο $\lfloor \lg n \rfloor$.

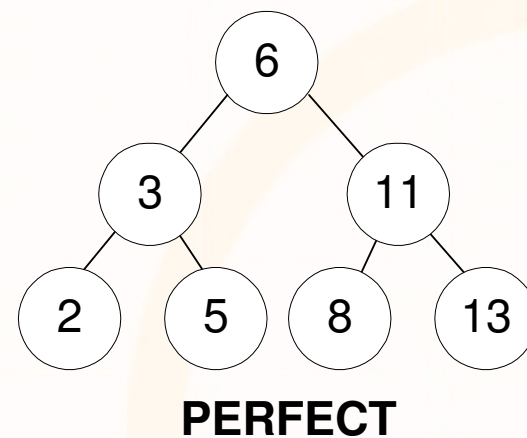


Δυαδικά Δένδρα

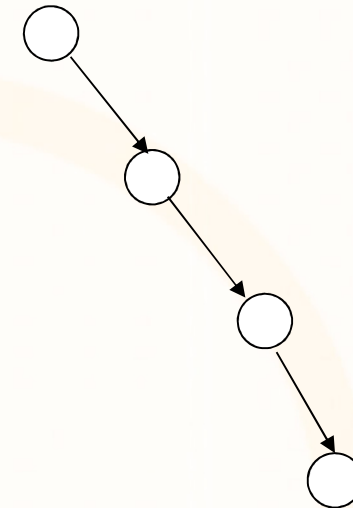
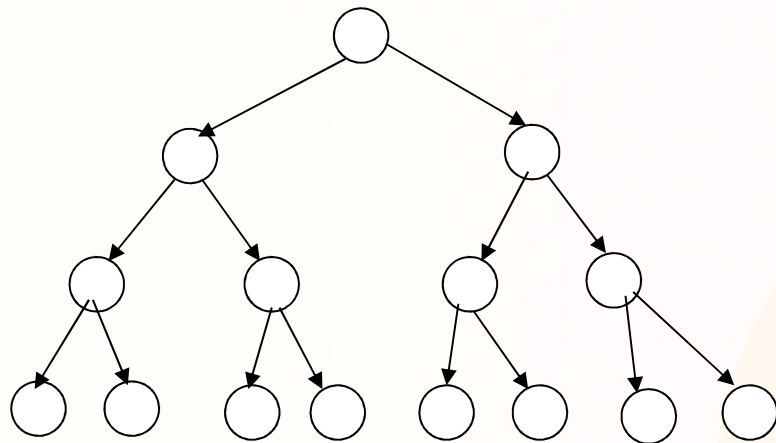
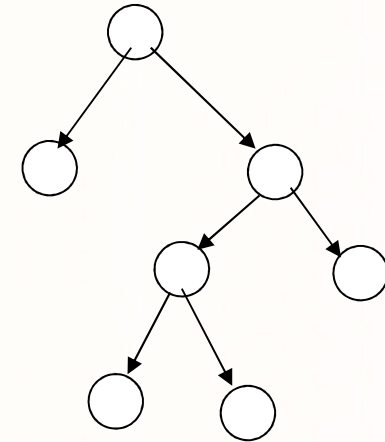
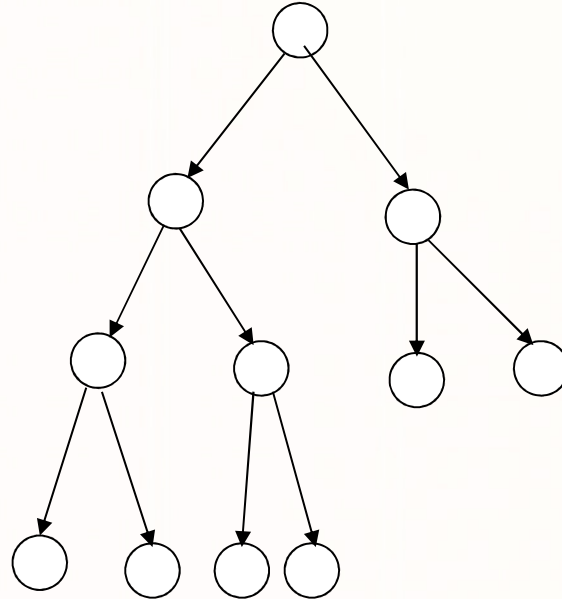
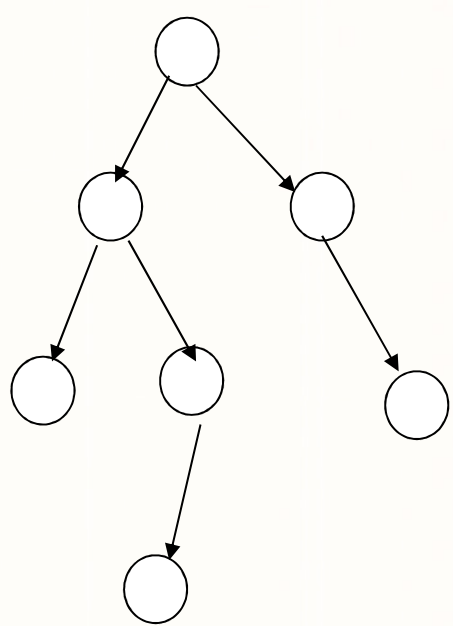
- Ένα δυαδικό δένδρο είναι **γεμάτο (full)**, αν κάθε εσωτερικός του κόμβος έχει δύο απογόνους.



- Ένα δυαδικό δένδρο είναι **τέλειο (perfect)**, αν είναι γεμάτο και όλοι του οι κόμβοι που βρίσκονται στο ίδιο επίπεδο έχουν το ίδιο βάθος.



Παραδείγματα Δυαδικά Δένδρα



Αναπαράσταση Δένδρων στη Μνήμη

- Αφού κάθε κόμβος σε ένα δυαδικό δένδρο έχει το πολύ δύο παιδιά, μπορούμε να κρατούμε δείκτες στο καθένα από αυτά.
- Δηλαδή, ένας κόμβος μπορεί να υλοποιηθεί ως μια εγγραφή NODE με τρία πεδία (παρόμοια με κόμβο διπλά συνδεδεμένης λίστας).

```
typedef struct node {  
    int         val;  
    struct node *left;  
    struct node *right;  
} NODE;
```

1. **val**, όπου αποθηκεύουμε την πληροφορία του κόμβου,
 2. **left**, τύπου pointer, ο οποίος δείχνει το αριστερό, υπόδενδρο που ριζώνει στον συγκεκριμένο κόμβο, και
 3. **right**, τύπου pointer, ο οποίος δείχνει το δεξιό υπόδενδρο που ριζώνει στον συγκεκριμένο κόμβο.
- Έτσι, ένα δυαδικό δένδρο υλοποιείται ως ένας δείκτης στη ρίζα του δένδρου, δηλαδή δείκτης σε εγγραφή τύπου NODE → **NODE *root;**

Αναπαράσταση Δένδρων στη Μνήμη

- Επίσης μπορεί να υλοποιηθεί σαν μία ξεχωριστή δομή TREE η οποία να περιλαμβάνει και πληροφορίες όπως μέγεθος, ύψος, κ.τ.λ.
- Παρόμοια με Λίστες, Στοίβες, Ουρές, κτλ.
- Παράδειγμα:

```
typedef struct node {  
    int val;  
    struct node *left;  
    struct node *right;  
} NODE;
```

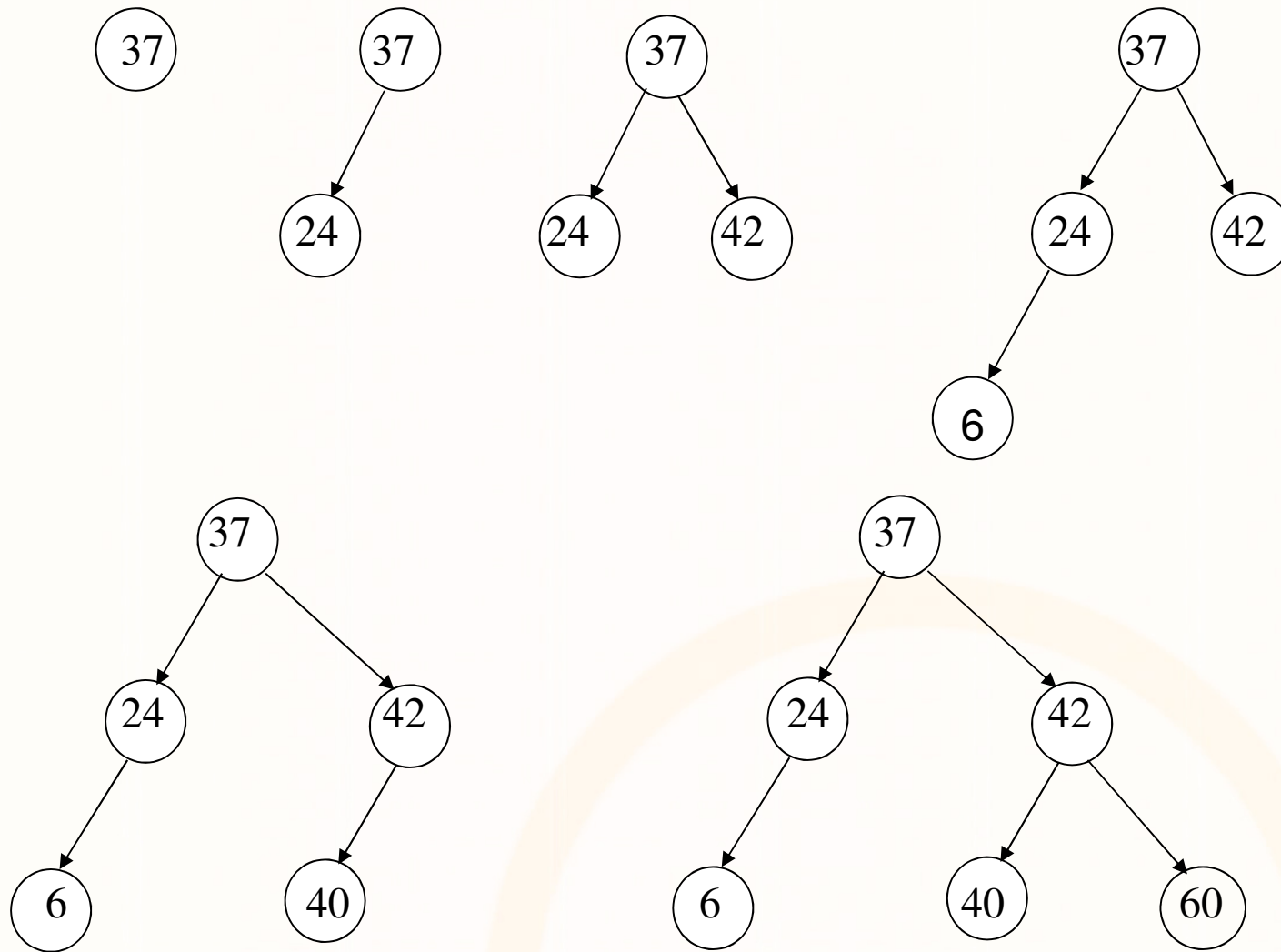
```
typedef struct stack {  
    NODE *head;  
    int size;  
} TREE;
```

- Σε αυτή την περίπτωση, ένα δυαδικό δένδρο υλοποιείται σαν μία μεταβλητή τύπου TREE που ο δείκτης head δείχνει στη ρίζα του δένδρου.

Δυαδικά Δένδρα Αναζήτησης ΔΔΑ (Binary Search Trees)

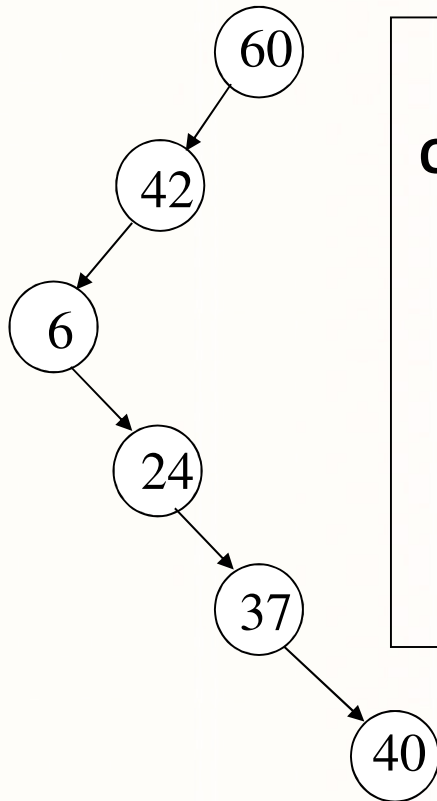
- Το πιο σημαντικό πλεονέκτημα της χρήσης δυαδικών δένδρων η αποδοτική αναζήτηση σε ένα σύνολο στοιχείων
- Υποθέτουμε την ύπαρξη μιας σχέσης στο σύνολο των στοιχείων που επεξεργαζόμαστε, έστω τη σχέση $<$ πάνω στο σύνολο των ακεραίων.
- Ένα **δυαδικό δένδρο αναζήτησης (ΔΔΑ)** είναι ένα δυαδικό δένδρο κάθε κόμβος u του οποίου ικανοποιεί τα εξής:
 1. τα κλειδιά του αριστερού υποδένδρου του u είναι μικρότερα από το κλειδί του u
 2. τα κλειδιά του δεξιού υποδένδρου του u είναι μεγαλύτερα (ή ίσο) από το κλειδί του u .

Παράδειγμα Κτισίματος ενός ΔΔΑ



Κτίζοντας ένα ΔΔΑ

Δένδρο με τα ίδια στοιχεία τοποθετημένα με διαφορετική σειρά: 60, 42, 6, 24, 37, 40.



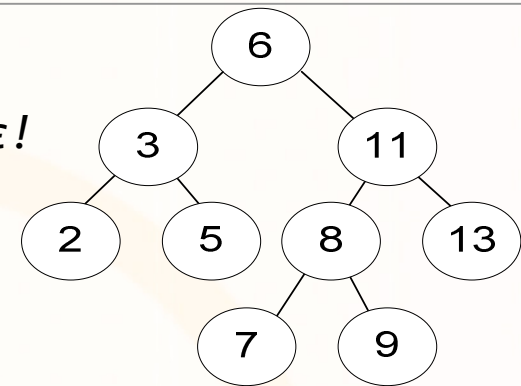
Ποιο είναι το χειρότερο σενάριο?
Οι τιμές εισόδου φθάνουν με την ακόλουθη σειρά
60,42,40,37,24,6 (φθίνουσα σειρά)
ή
6,24,37,40,42,60 (αύξουσα σειρά)
Αυτές οι δυο τιμές εισόδου παράγουν μια
συνδεμένη λίστα

Διαδικασία Εύρεσης Στοιχείου – FindNode()

- Απλή αναδρομική στρατηγική: συγκρίνουμε το στοιχείο που μας ενδιαφέρει α με το στοιχείο της ρίζας του δένδρου β (αν υπάρχει) και
 1. αν $\alpha = \beta$, σταματούμε,
 2. αν $\alpha < \beta$, προχωρούμε στο αριστερό υπόδενδρο,
 3. αν $\alpha > \beta$ προχωρούμε στο δεξιό υπόδενδρο.

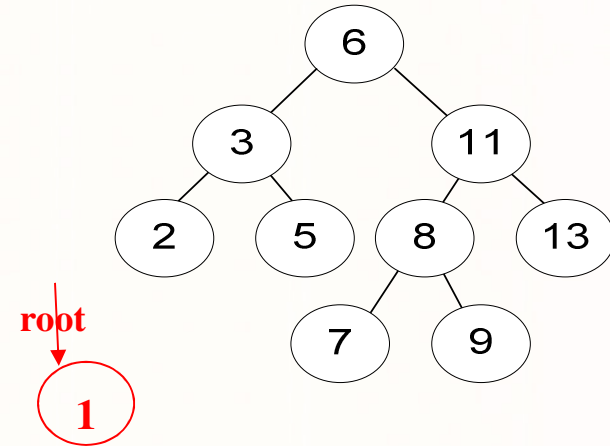
```
NODE *FindNode(NODE *root, int val) {  
    if (root == NULL)  
        return NULL; // το στοιχείο δεν βρέθηκε  
    else if (root->val == val) // το στοιχείο βρέθηκε!  
        return root;  
    else { // ακόμα δεν βρέθηκε το στοιχείο  
        if (val < root->val)  
            return FindNode(root->left, val);  
        else  
            return FindNode(root->right, val);  
    }  
}
```

Χείριστη περίπτωση $O(n)$
(εάν έχουμε λίστα)



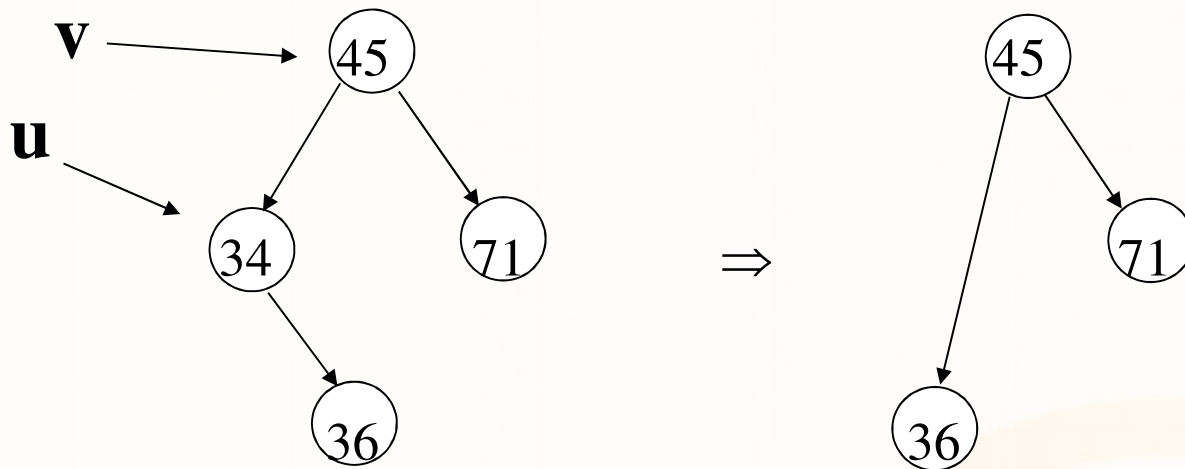
Διαδικασία Εισαγωγής Κόμβου – InsertNode()

```
NODE *InsertNode(NODE *root, int val){
    if (root == NULL) { // εισαγωγή κόμβου
        root = (NODE *) malloc(sizeof(NODE));
        if (root==NULL) {
            printf("Out of Memory!");
            exit(1);
        }
        root->val = val;
        root->left = NULL;
        root->right = NULL;
    }
    else {
        if (val < root->val) // εισαγωγή αριστερά
            root->left = InsertNode(root->left, val);
        else // εισαγωγή δεξιά
            root->right = InsertNode(root->right, val);
    }
    return root;
}
```



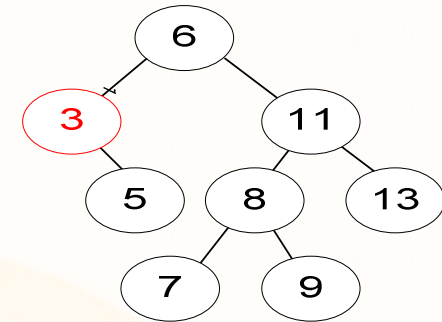
Εξαγωγή του μικρότερου κόμβου – DeleteMin()

1. Ακολουθούμε τους αριστερούς δείκτες όσο μπορούμε, φθάνοντας στον κόμβο με το μικρότερο στοιχείο, **u**.
2. Βρίσκουμε τον πατέρα **v** του **u** και αλλάζουμε τον αριστερό δείκτη του **v** ώστε να δείχνει στο δεξιό παιδί του **u**.



Εξαγωγή του μικρότερου κόμβου

```
NODE *deleteMin(NODE *root) {  
    NODE *p;  
    if (root == NULL) {  
        // μικρότερο στοιχείο δεν βρέθηκε  
        return NULL;    // πχ αν είναι κενό το δένδρο  
    }  
    else if (root->left == NULL) {  
        // φτάσαμε στο μικρότερο  
        p = root;  
        root = root->right;  
        free(p);  
    }  
    else { // αναδρομική προχώρηση προς τα αριστερά  
        root->left = deleteMin(root->left);  
    }  
    return root;  
}
```

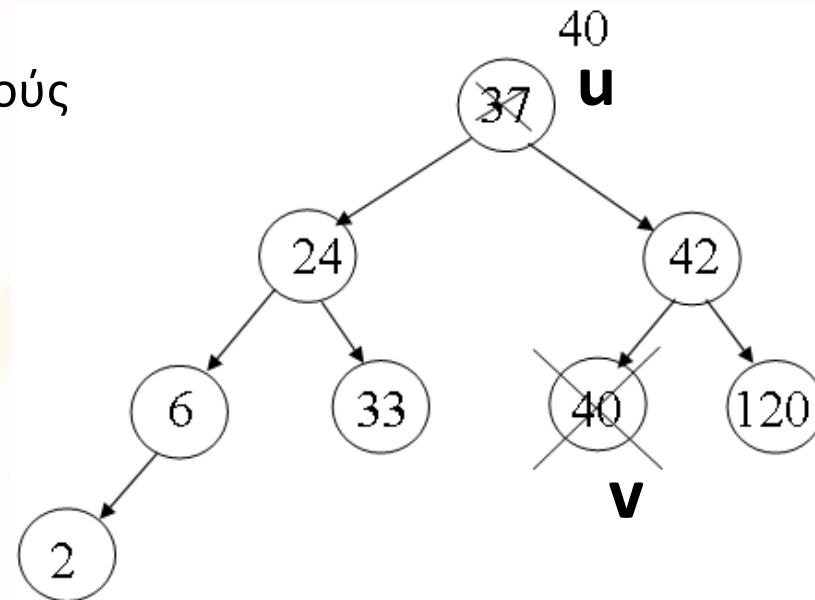


Διαγραφή κόμβου i (μόνο η ιδέα)

Για να αφαιρέσουμε ένα κλειδί X από ένα ΔΔΑ:

1. Βρίσκουμε τον κόμβο u που περιέχει το X .
2. Πριν διαγράψουμε τον u βρίσκουμε τον μικρότερο (από δεξιά) **απόγονο του** (τον οποίο ονομάζουμε v).
3. Η ιδέα είναι ότι ο v πρέπει να αντικαταστήσει τον u !

Η υλοποίηση γίνεται πιο απλή χρησιμοποιώντας επαναληπτικούς βρόγχους παρά αναδρομή.



Σύγκριση Δυαδικών Δένδρων

Να γραφεί μια συνάρτηση η οποία να επιστρέφει “1” αν δυο δυαδικά δένδρα είναι τα ίδια και “0” στην αντίθετη περίπτωση

```
int sameTree(NODE *a, NODE *b) {  
  
    // και τα δυο δένδρα είναι κενά => άρα επιστρέφουμε TRUE  
    if (a==NULL && b==NULL)  
        return 1;  
  
    // και τα δυο δένδρα δεν είναι κενά - συγκρίνουμε τις ρίζες τους  
    else if (a!=NULL && b!=NULL) {  
        return(  
            (a->data == b->data)  
            &&  
            sameTree(a->left, b->left)  
            &&  
            sameTree(a->right, b->right)  
        );  
    }  
  
    // το ένα εκ των δυο υπό-δένδρων είναι κενό => επιστρέφουμε FALSE  
    else return 0;  
}
```

