Διάλεξη 12: Δέντρα ΙΙ - Δυαδικά Δέντρα

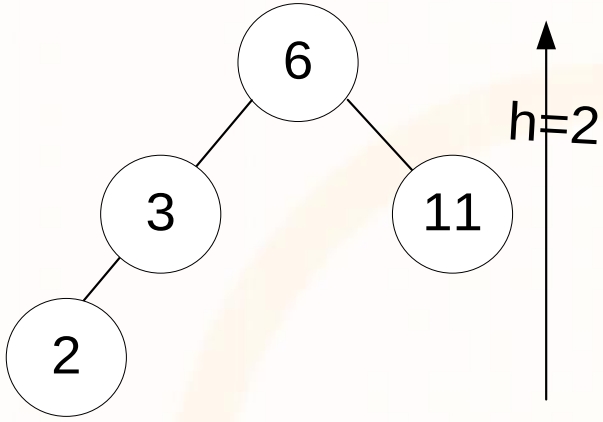
Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Δυαδικά Δένδρα
- Δυαδικά Δένδρα Αναζήτησης (ΔΔΑ)
  - Εύρεση Τυχαίου, Μέγιστου, Μικρότερου στοιχείου
  - Εισαγωγή στοιχείου
  - Διαγραφή Μικρότερου και Τυχαίου στοιχείου
  - Σύγκριση 2 ΔΔΑ, Διάσχιση

Δυαδικά Δένδρα

- Ένα δένδρο είναι **δυαδικό** αν όλοι οι κόμβοι του έχουν **βαθμό** ≤ 2.

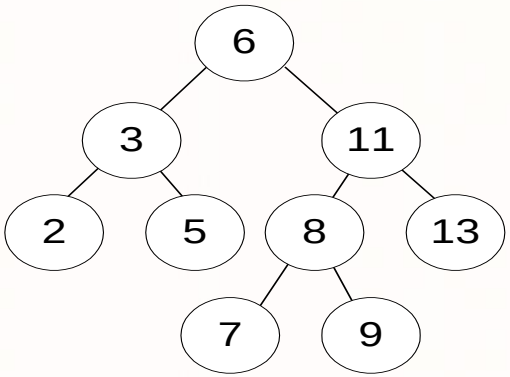
  Ορισμός: Δυαδικό δένδρο λέγεται ένα δένδρο το οποίο :
  - είτε είναι κενό,
  - ή αποτελείται από μια **ρίζα** και **δύο δυαδικά υπόδενδρα**. Αναφερόμαστε στα δύο υποδένδρα ως το **αριστερό** και το **δεξιό υπόδενδρο**.
- Το ύψος ενός δυαδικού δένδρου με **n** κόμβους μπορεί να είναι το πολύ : **n-1 (συνδεδεμένη λίστα)** και το λιγότερο $\lfloor \mathbf{lg\ n} \rfloor$.



floor(lg 4)= 2

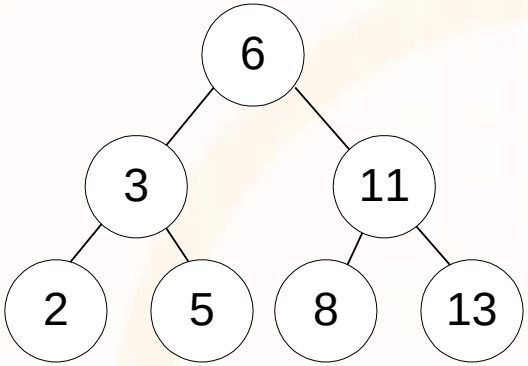
Δυαδικά Δένδρα (συν.)

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



FULL

- Ένα δυαδικό δένδρο είναι **τέλειο (perfect),** αν είναι γεμάτο και όλα τα φύλλα έχουν το ίδιο βάθος.



PERFECT

Δυαδικά Δένδρα (συν.)

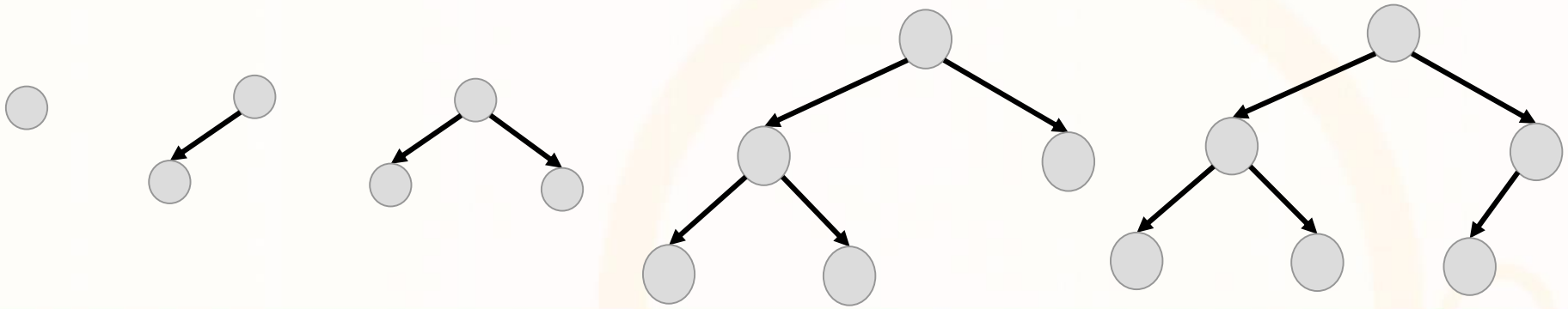
- Ένα δυαδικό δένδρο είναι **πλήρες** (complete) αν

  Αναδρομικός Ορισμός

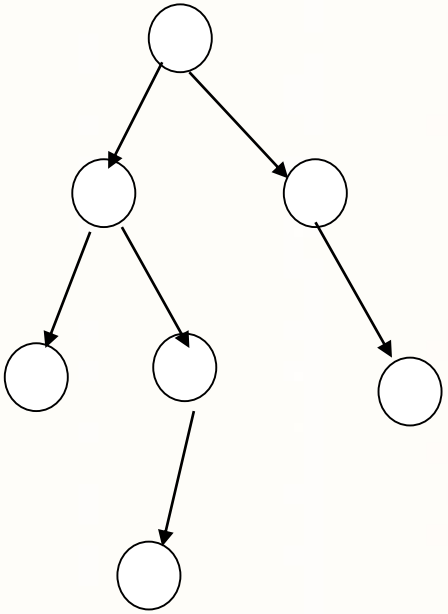
  1. έχει ύψος 0 και ένα κόμβο,
  2. έχει ύψος 1 και η ρίζα του έχει είτε δύο παιδιά είτε ένα αριστερό παιδί.
  3. έχει ύψος h και η ρίζα του έχει ένα τέλειο αριστερό υπόδενδρο ύψους h-1 και ένα πλήρες δεξιό υπόδενδρο ύψους h-1,

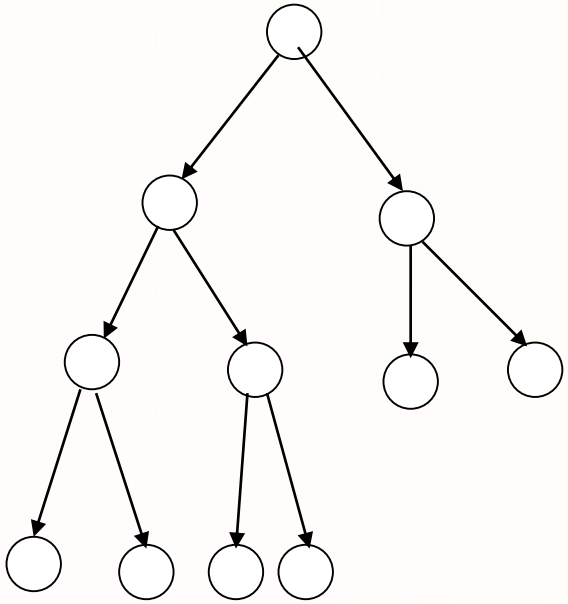
     ή

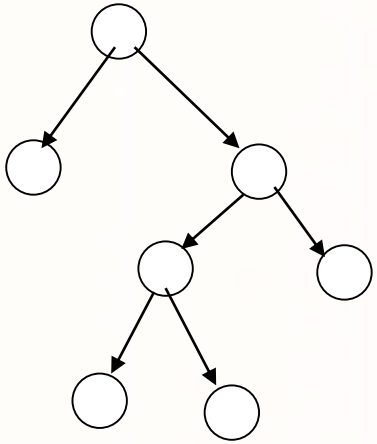
     ένα πλήρες αριστερό υπόδενδρο ύψους h-1 και ένα τέλειο δεξιό υπόδενδρο ύψους h-2.

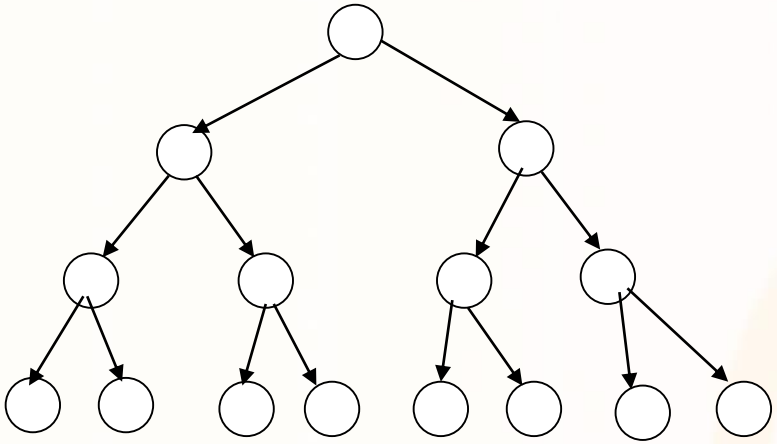


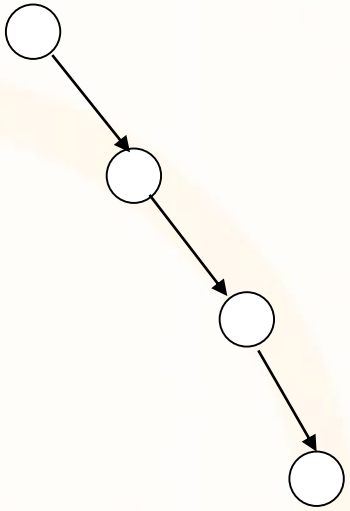
Παραδείγματα Δυαδικά Δένδρα











Θεώρημα

1. *Ένα γεμάτο δυαδικό (μη άδειο) δένδρο με n εσωτερικούς κόμβους, έχει n+1 φύλλα.*
2. *Κάθε δυαδικό δένδρο με n κόμβους έχει n+1 null δείκτες*

Απόδειξη (2) με τη μέθοδο της μαθηματικής επαγωγής:

Βάση της επαγωγής: n = 0

Το δένδρο αποτελείται από ένα NULL δείκτη και το ζητούμενο έπεται.

Υπόθεση της επαγωγής: Έστω ότι κάθε δυαδικό δένδρο με k κόμβους, k < m, έχει k+1 NULL δείκτες.

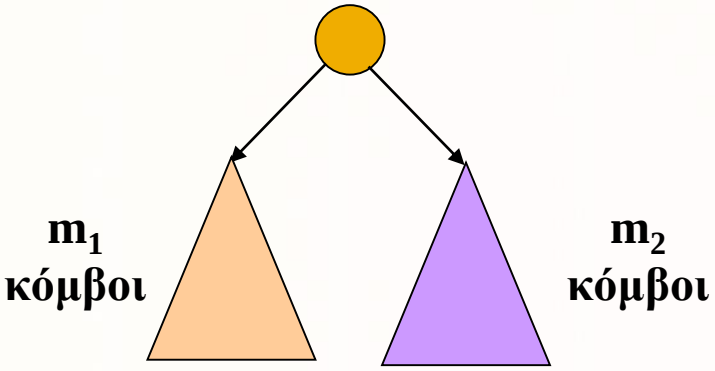
Βήμα της επαγωγής:

Έστω δυαδικό δένδρο με m κόμβους.

Θέλουμε να δείξουμε πως περιέχει m+1 NULL δείκτες.

Θεώρημα

Ένα οποιοδήποτε δυαδικό δένδρο έχει την πιο κάτω μορφή:



$$m_1 + m_2 + 1 = m$$

Ο αριθμός NULL δεικτών του δένδρου είναι

= ο αριθμός NULL δεικτών του αριστερού υποδένδρου

+

ο αριθμός NULL δεικτών του δεξιού υποδένδρου

= {από την υπόθεση της επαγωγής και αφού $m_1, m_2 < m$}

$(m_1+1) + (m_2+ 1) = (m_1+ m_2+1) + 1$

= $m + 1$

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 **TreeNode** με τρία πεδία (παρόμοια με κόμβο διπλά συνδεδεμένης λίστας).
  1. **key**, όπου αποθηκεύουμε το κλειδί 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.
  4. ... διάφορα άλλα χρήσιμα στοιχεία

```
private class TreeNode<E> {
  E key;
  BinaryTreeNode<E> left;
  BinaryTreeNode<E> right;

  //other useful fields
  boolean isLeaf, isInternal,
          isRoot;
  Object data;    // data

  BinaryTreeNode(E key){
    this.key = key;
    this.left = null;
    this.right = null;
  }
}
```

Αναπαράσταση Δένδρων στη Μνήμη (συν.)

- Έτσι, ένα δυαδικό δένδρο υλοποιείται ως ένα δείκτης προς τη ρίζα του δένδρου, δηλαδή μία αναφορά σε εγγραφή τύπου **TreeNode.**
- 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.

```
public class
Tree<E extends Comparable>{

  private class TreeNode<E>
  {...}

  private BinaryTreeNode<E> head;
  private int size;
  private int height;

  public void makeEmpty() {
    this.head=null;
    this.size=0;
  }

  public boolean isEmpty() {
    return this.size==0;
  }
  public int size() {
    return this.size;
  }
  …
```

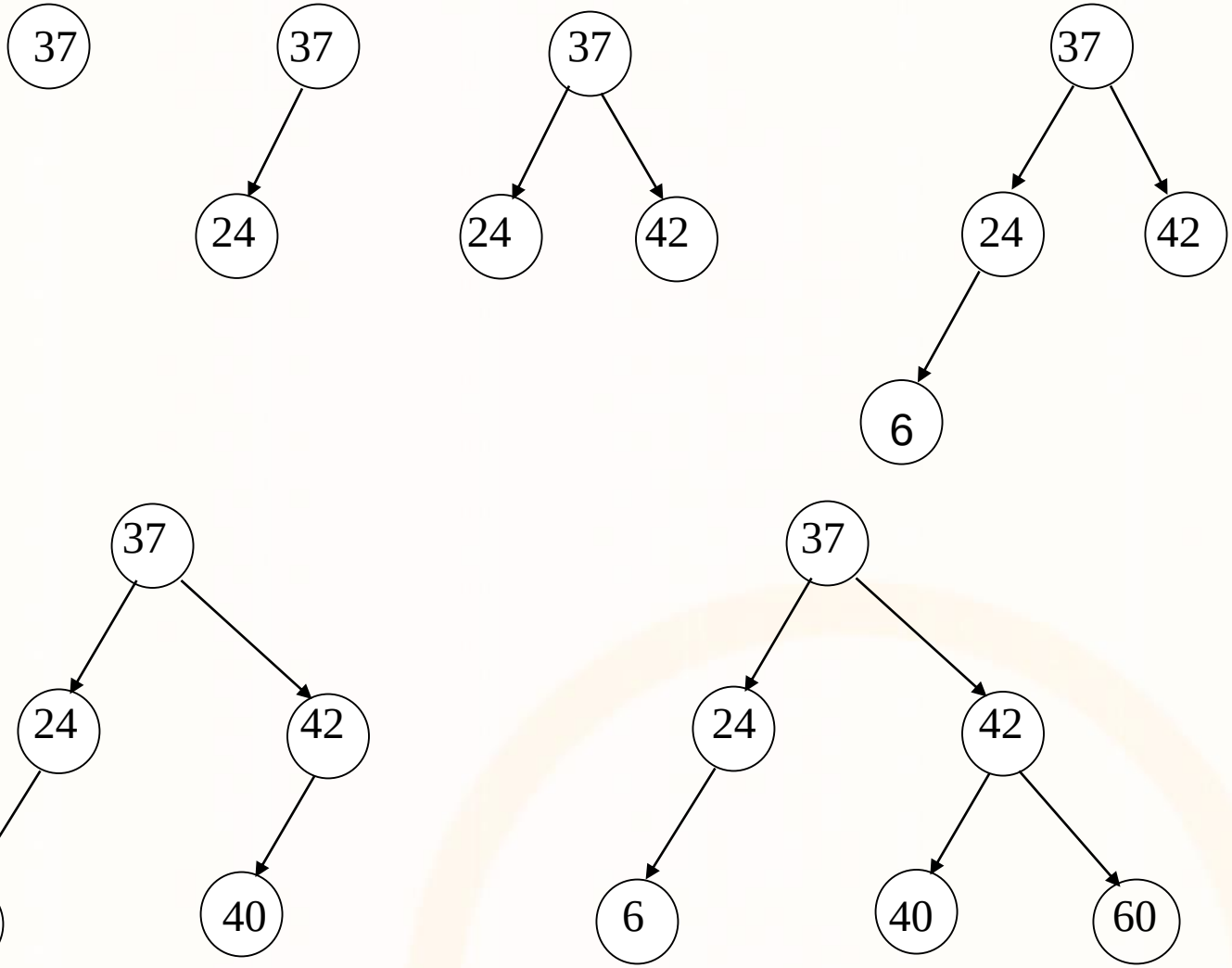
Δυαδικά Δένδρα Αναζήτησης ΔΔΑ (Binary Search Trees)

- Το πιο σημαντικό πλεονέκτημα της χρήσης δυαδικών δένδρων η αποδοτική αναζήτηση σε ένα σύνολο στοιχείων

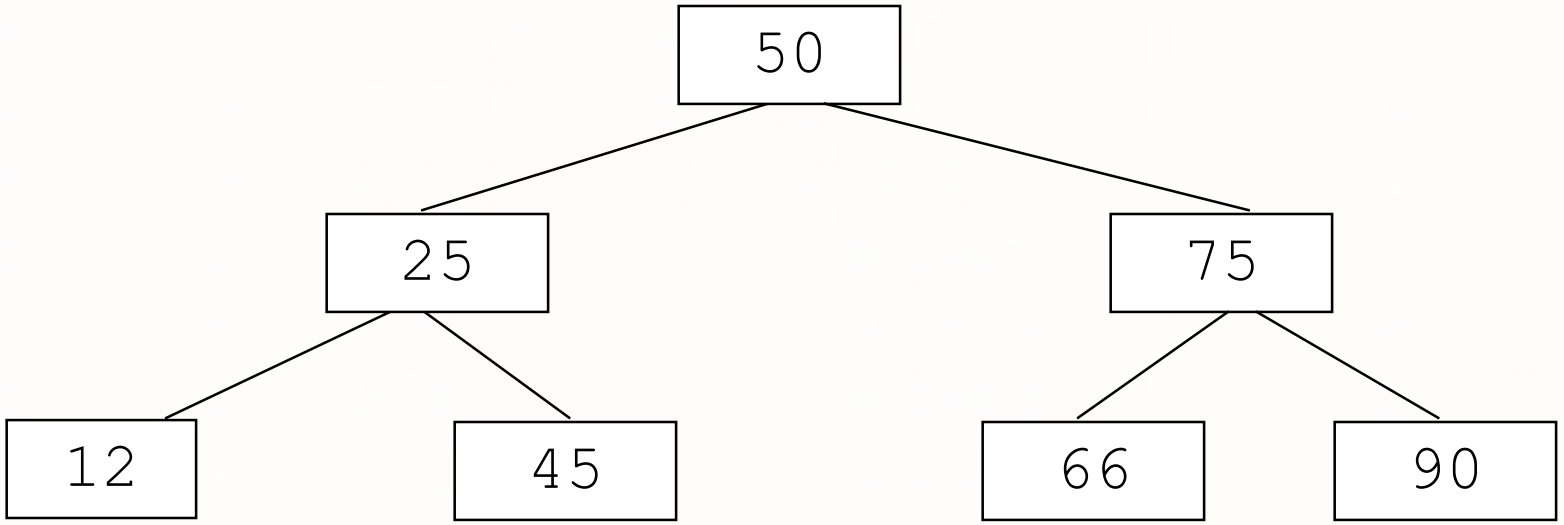
- Υποθέτουμε την ύπαρξη μιας σχέσης στο σύνολο των στοιχείων που επεξεργαζόμαστε, έστω τη σχέση < πάνω στο σύνολο των ακεραίων.

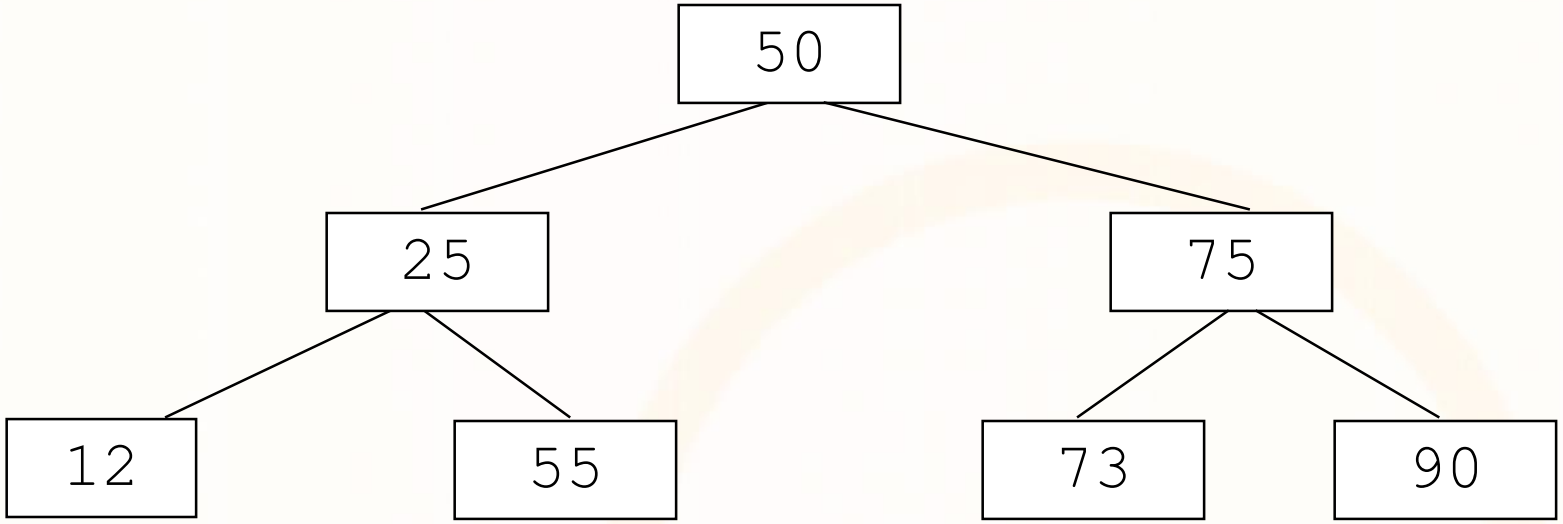
- **Ένα δυαδικό δένδρο αναζήτησης (ΔΔΑ) είναι ένα δυαδικό δένδρο κάθε κόμβος u του οποίου ικανοποιεί τα εξής:**
  1. τα κλειδιά του αριστερού υποδένδρου του u είναι μικρότερα από το κλειδί του u
  2. τα κλειδιά του δεξιού υποδένδρου του u είναι μεγαλύτερα (ή ίσο) από το κλειδί του u.

Παράδειγμα Κτισίματος ενός ΔΔΑ



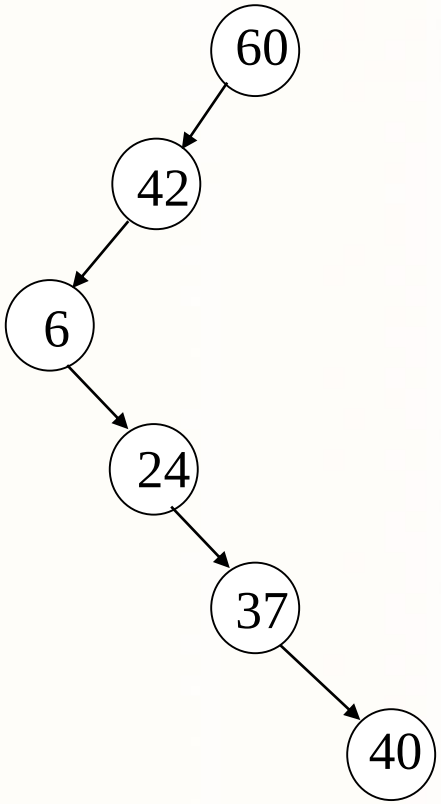
Είναι αυτά ΔΔΑ; (*μόνο τα κλειδιά εμφανίζονται)*





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)

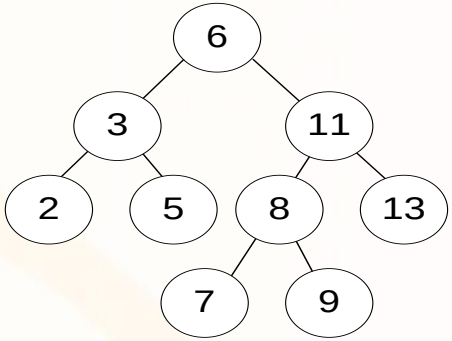
ή

6,24,37,40,42,60 (αύξουσα σειρά)

Αυτές οι δυο τιμές εισόδου παράγουν μια συνδεμένη λίστα

Διαδικασία Εύρεσης Τυχαίου Στοιχείου

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



- Χρόνος Εκτέλεσης;
  - **Χείριστη Περίπτωση:** O(h) όπου h το ύψος του δέντρου, O(n) αν το δέντρο είναι λίστα
  - **Μέση Περίπτωση:** $O(\log_2 n)$ αν το δέντρο είναι ισοζυγισμένο

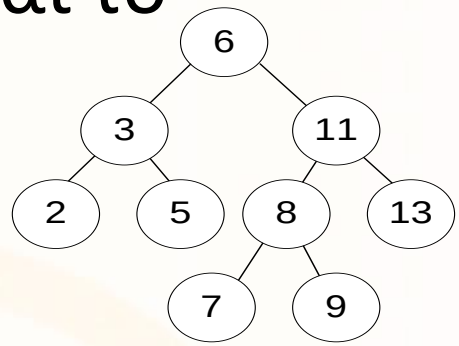
Διαδικασία Εύρεσης Τυχαίου Στοιχείου: Υλοποίηση

```
public BinaryTreeNode<E> find(E key) {
  return this.find(key, this.head);
}

private BinaryTreeNode<E> find(E key, BinaryTreeNode<E> node) {
  if(node==null){
    return null;
  }
  else if (key.compareTo(node.key) == 0) {
    return node;
  }
  else if (key.compareTo(node.key) < 0) {
    return find(key, node.left);
  }
  else  {
    return find(key, node.right);
  }
}
```

Διαδικασία Εύρεσης Ελαχίστου/Μεγίστου στοιχείου

- **Στόχος**: να επιστραφεί ο κόμβος που περιέχει το μικρότερο (μεγαλύτερο) κλειδί στο δένδρο.

- **Αλγόριθμος**: Ξεκινάμε από τη ρίζα και πηγαίνουμε αριστερά (δεξιά) όσο υπάρχει ένα αριστερό (δεξιό) παιδί. Το σημείο που σταματάμε είναι το μικρότερο (μεγαλύτερο) στοιχείο.



- Χρόνος Εκτέλεσης;
  - Παρόμοια με find

Διαδικασία Εύρεσης Ελαχ./Μεγίστ.: Υλοποίηση

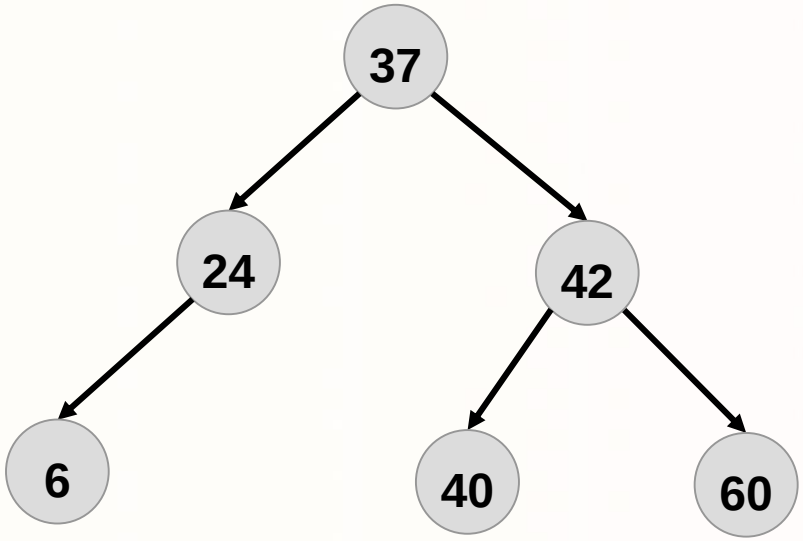
```
public E findMin(){
  BinaryTreeNode<E> tmp = this.findMin(this.head);
  if(tmp!=null)
    return tmp.key;
  return null;
}

public BinaryTreeNode<E> findMin(BinaryTreeNode<E> node){
  if(node!=null)
    while(node.left!=null)
      node = node.left;
  return node;
}
```

Διαδικασία Εισαγωγής

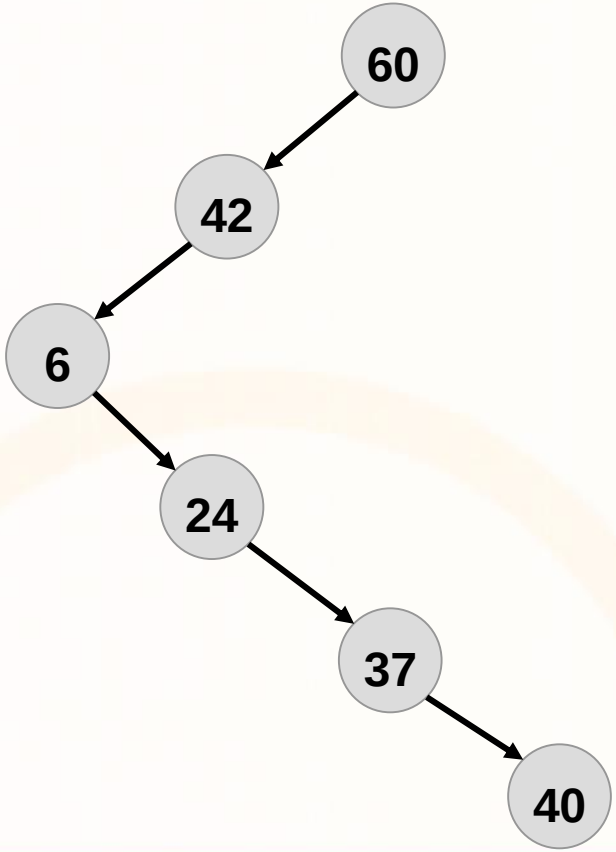
- Σημαντικό: παίζει ρόλο η σειρά εισαγωγής

- Παραδείγματα

A. Εισαγωγή 37, 24, 42, 6, 40, 60

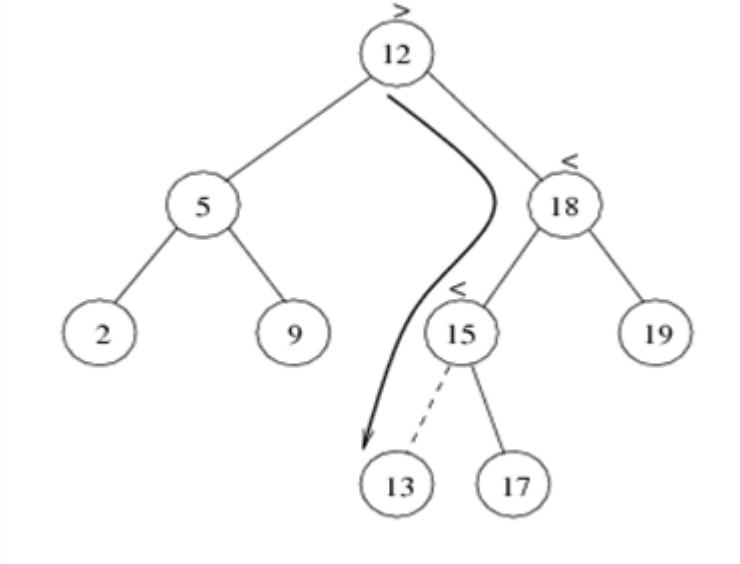


B. Εισαγωγή 60, 42, 6, 24, 37, 40



Διαδικασία Εισαγωγής Κόμβου

- Διασχίζουμε το δέντρο, όπως θα κάναμε με τη find
- Εάν το X βρεθεί, δεν κάνουμε καμία ενέργεια
- Διαφορετικά, εισάγουμε το X στο τελευταίο σημείο του μονοπατιού που διασχίστηκε



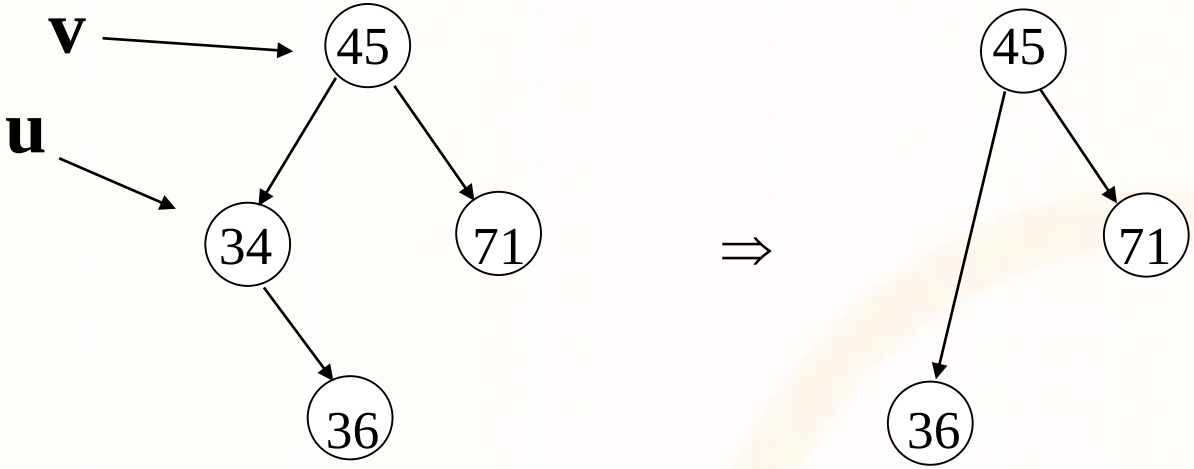
- Χρονική πολυπλοκότητα= O(h - ύψος του δένδρου)

Διαδικασία Εισαγωγής Κόμβου: Υλοποίηση

```
public void insert(E key) {
  this.head = this.insert(key, this.head);
  this.size++;
}
private BinaryTreeNode<E> insert(E key,BinaryTreeNode<E> node) {
  if(node==null){
    node = new BinaryTreeNode<E>(key);
  }
  else if (key.compareTo(node.key) < 0) {
    node.left = insert(key, node.left);
  }
  else if (key.compareTo(node.key) > 0) {
    node.right = insert(key, node.right);
  }
  return node;
}
```

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.

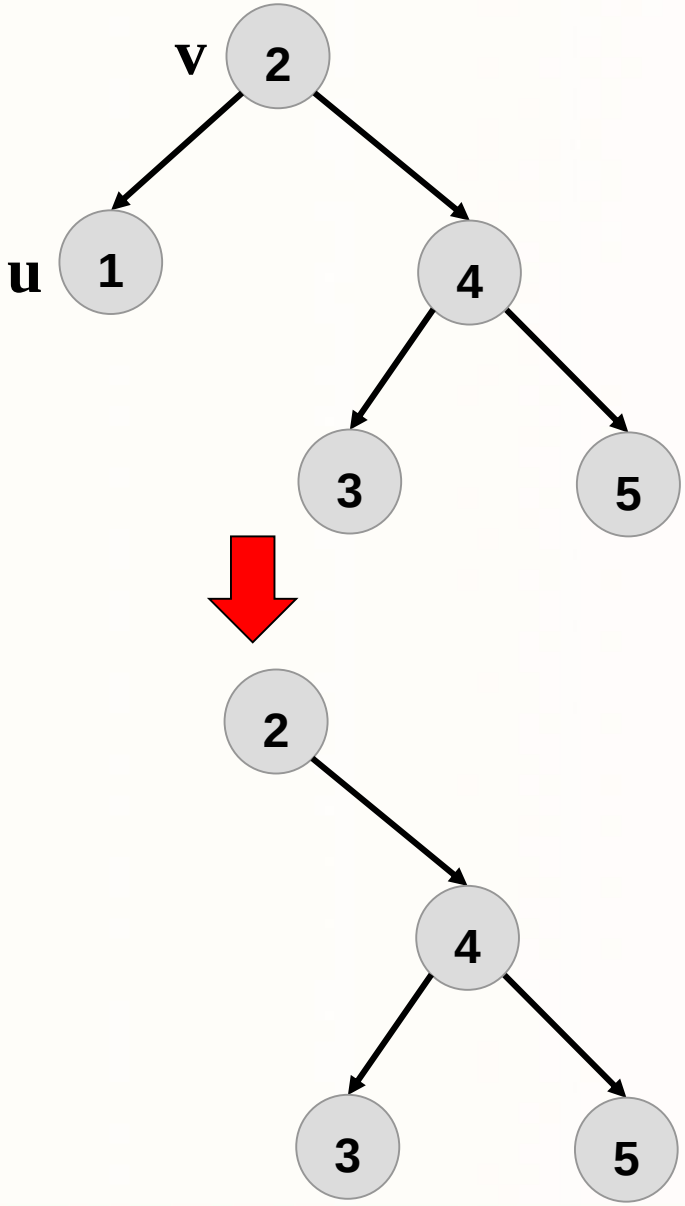


Διαδικασία Διαγραφής Στοιχείου

Για να αφαιρέσουμε ένα κλειδί i από ένα ΔΔΑ:

1. Βρίσκουμε τον κόμβο **u** που περιέχει το i. Ας υποθέσουμε πως **v** είναι ο πατέρας του **u**.
2. Αν **u** είναι φύλλο, τότε αλλάζουμε τον δείκτη του v που δείχνει στο **u**, ώστε να γίνει null.
3. Αν ο **u** έχει ένα παιδί, τότε αλλάζουμε τον δείκτη του **v** που δείχνει τον **u**, ώστε να δείχνει στο παιδί του **u**.
4. Αν ο **u** έχει δύο παιδιά,
   - αλλάζουμε το κλειδί του **u** ώστε να γίνει το μικρότερο από τα κλειδιά όλων των απογόνων του που έχουν κλειδιά μεγαλύτερα του i.
   - καλούμε τη μέθοδο deleteMin στο δεξιό παιδί του **u**.

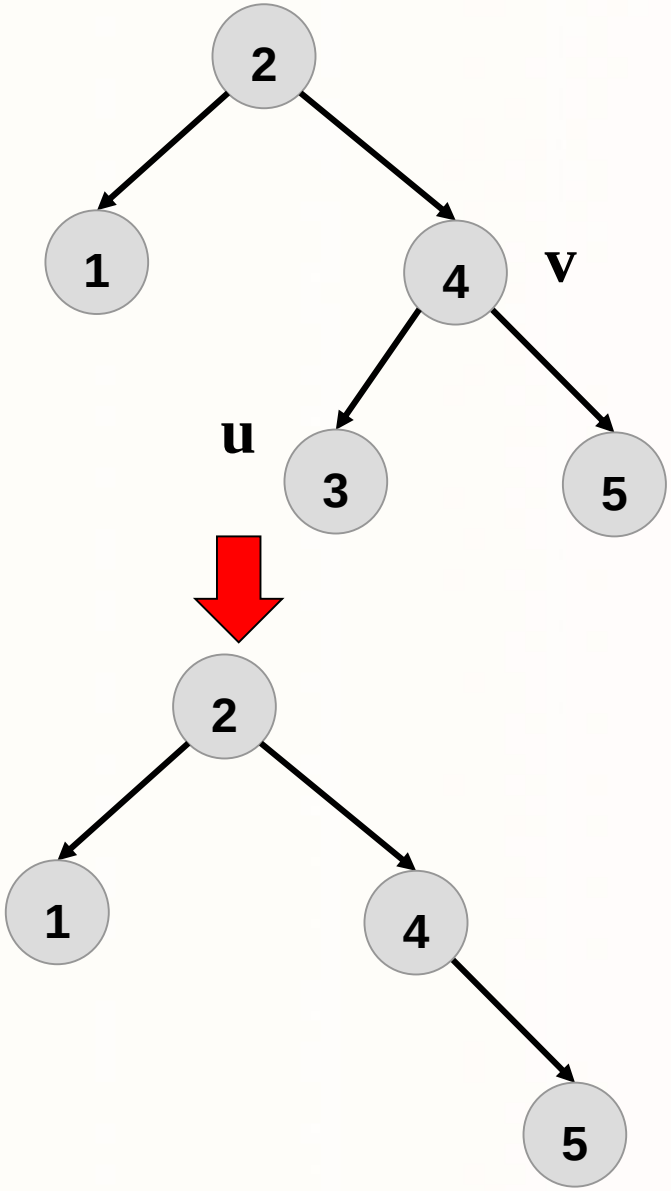
Παραδείγματα Διαγραφής Στοιχείου



Διαγραφή του 1

1. Βρίσκουμε τον κόμβο **u** που περιέχει το i. Ας υποθέσουμε πως **v** είναι ο πατέρας του **u**.
   - 2: μετακίνηση αριστερά
   - 1: τέλος αναζήτησης
2. Αν **u** είναι φύλλο, τότε αλλάζουμε τον δείκτη του v που δείχνει στο **u**, ώστε να γίνει null.
   - v.left = null

Παραδείγματα Διαγραφής Στοιχείου

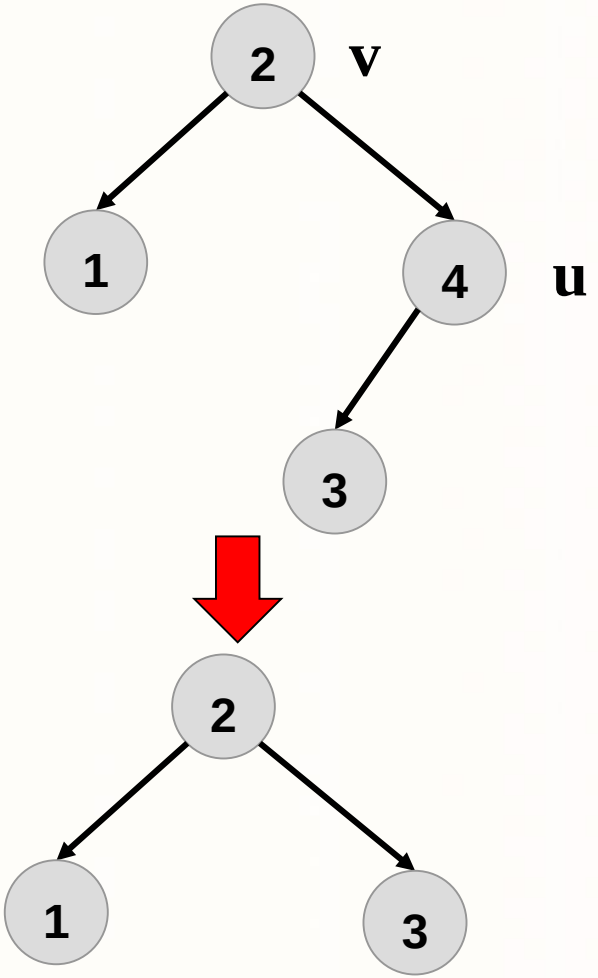


Διαγραφή του 3

1. Βρίσκουμε τον κόμβο **u** που περιέχει το i. Ας υποθέσουμε πως **v** είναι ο πατέρας του **u**.
   - 2: μετακίνηση δεξιά
   - 4: μετακίνηση αριστερά
   - 3: τέλος αναζήτησης

2. Αν **u** είναι φύλλο, τότε αλλάζουμε τον δείκτη του v που δείχνει στο **u**, ώστε να γίνει null.
   - v.left = null

Παραδείγματα Διαγραφής Στοιχείου



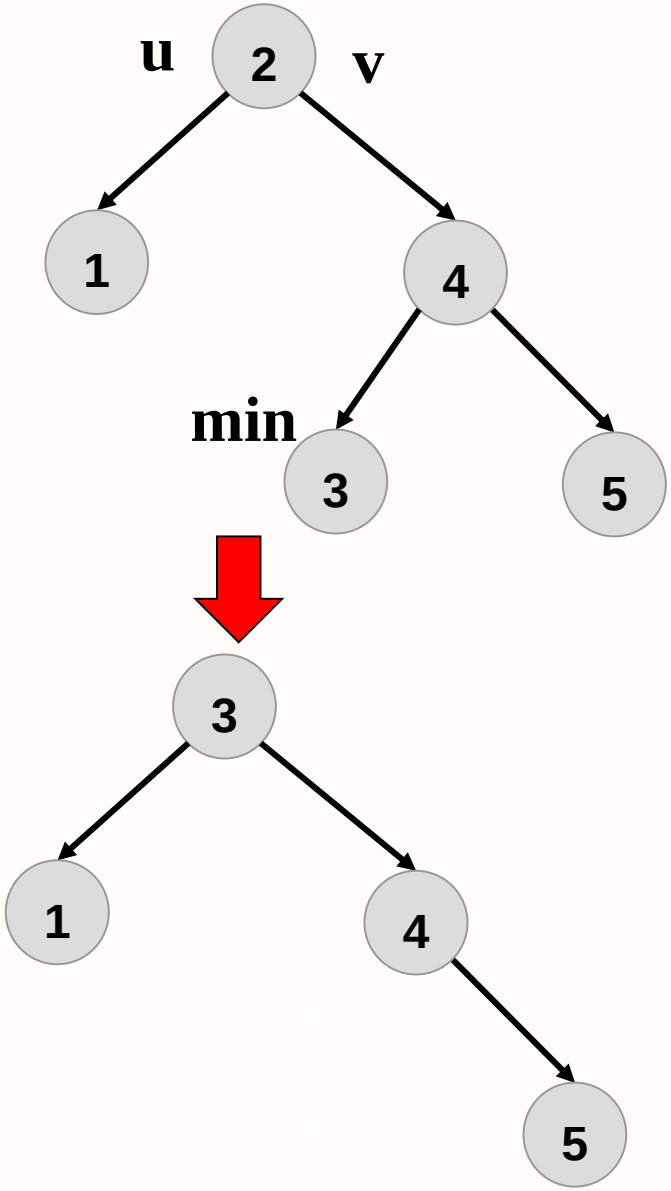
Διαγραφή του 4

1. Βρίσκουμε τον κόμβο **u** που περιέχει το i. Ας υποθέσουμε πως **v** είναι ο πατέρας του **u**.
   - 2: μετακίνηση δεξιά
   - 4: τέλος αναζήτησης

2. ~~Αν **u** είναι φύλλο, τότε αλλάζουμε τον δείκτη του v που δείχνει στο **u**, ώστε να γίνει null.~~

3. Αν ο **u** έχει ένα παιδί, τότε αλλάζουμε τον δείκτη του **v** που δείχνει τον **u**, ώστε να δείχνει στο παιδί του **u**.
   - v.right = u.left;

Παραδείγματα Διαγραφής Στοιχείου



Διαγραφή του 2

1. Βρίσκουμε τον κόμβο **u** που περιέχει το i. Ας υποθέσουμε πως **v** είναι ο πατέρας του **u**.
   - 2: τέλος αναζήτησης

~~2. Αν **u** είναι φύλλο, …~~

~~3. Αν ο **u** έχει ένα παιδί, …~~

4. Αν ο **u** έχει δύο παιδιά,
   - αλλάζουμε το κλειδί του **u** ώστε να γίνει το μικρότερο από τα κλειδιά όλων των απογόνων του που έχουν κλειδιά μεγαλύτερα του i.
   - καλούμε τη μέθοδο deleteMin στο δεξιό παιδί του **u**.

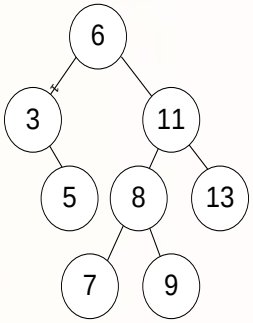
Άσκηση

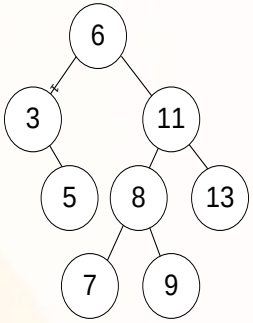
- Υλοποιήστε τον κώδικα διαγραφής στοιχείου σε δυαδικά δέντρα αναζήτησης

Σύγκριση Δυαδικών Δένδρων

Η σύγκριση 2 ΔΔΑ επιτυγχάνεται με διάσχιση του κάθε κόμβου αναδρομικά. Αν όλοι οι κόμβοι έχουν το ίδιο κλειδί τότε επιστρέφει “true” και “false” στην αντίθετη περίπτωση

```
public boolean sameTree(BinaryTree<E> t) {
  return this.sameTree(this.head, t.head);
}

private boolean sameTree(BinaryTreeNode<E> a, BinaryTreeNode<E> b) {
  // και τα δυο δένδρα είναι κενά => άρα επιστρέφουμε true
  if (a == null && b == null)
    return true;

  // και τα δυο δένδρα δεν είναι κενά – συγκρίνουμε τις ρίζες τους
  else if (a != null && b != null)
    return ((a.key.compareTo(b.key) == 0)
           && sameTree(a.left, b.left)
           && sameTree(a.right, b.right));

  // το ένα εκ των δυο υπό-δένδρων είναι κενό => επιστρέφουμε false
  else
    return false;
}
```





Διάσχιση ΔΔΑ

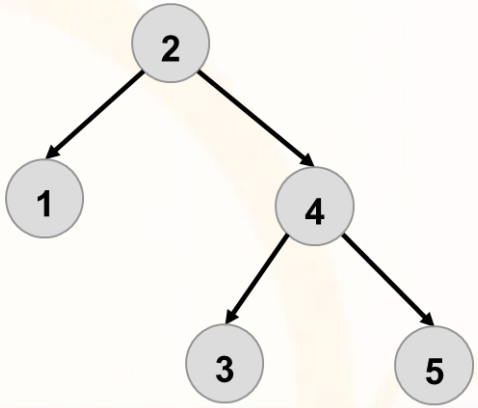
- Αν θέλουμε να **επισκεφθούμε όλους τους κόμβους** ενός δένδρου, μπορούμε να χρησιμοποιήσουμε ένα από τους πιο κάτω τρόπους, οι οποίοι **διαφέρουν στη σειρά με την οποία εξετάζουν τους κόμβους.**

1. **Προθεματική Διάσχιση: (Preorder Traversal)** επισκεπτόμαστε (εκτυπώνουμε) πρώτα κάποιο κόμβο και μετά τα παιδιά του.

2. **Μεταθεματική Διάσχιση: (Postorder Traversal)** επισκεπτόμαστε (εκτυπώνουμε) πρώτα τα παιδιά και ύστερα τον κόμβο.

3. **Ενδοθεματική Διάσχιση: (Inorder Traversal)** επισκεπτόμαστε (εκτυπώνουμε) πρώτα τα αριστερά παιδιά, μετά τον κόμβο και μετά τα δεξιά παιδιά.

- PreOrder: 2 1 4 3 5
- PostOrder: 1 3 5 4 2
- InOrder: 1 2 3 4 5



Διάσχιση ΔΔΑ: Υλοποίηση

```
public void preOrder() { this.preOrder(this.head); }
private void preOrder(BinaryTreeNode<E> node) {
  if (node != null) {
    System.out.print(node.key + " ");
    preOrder(node.left);
    preOrder(node.right);
  }
}
```

```
public void postOrder() { this.postOrder(this.head); }
private void postOrder(BinaryTreeNode<E> node) {
  if (node != null) {
    postOrder(node.left);
    postOrder(node.right);
    System.out.print(node.key + " ");
  }
}
```

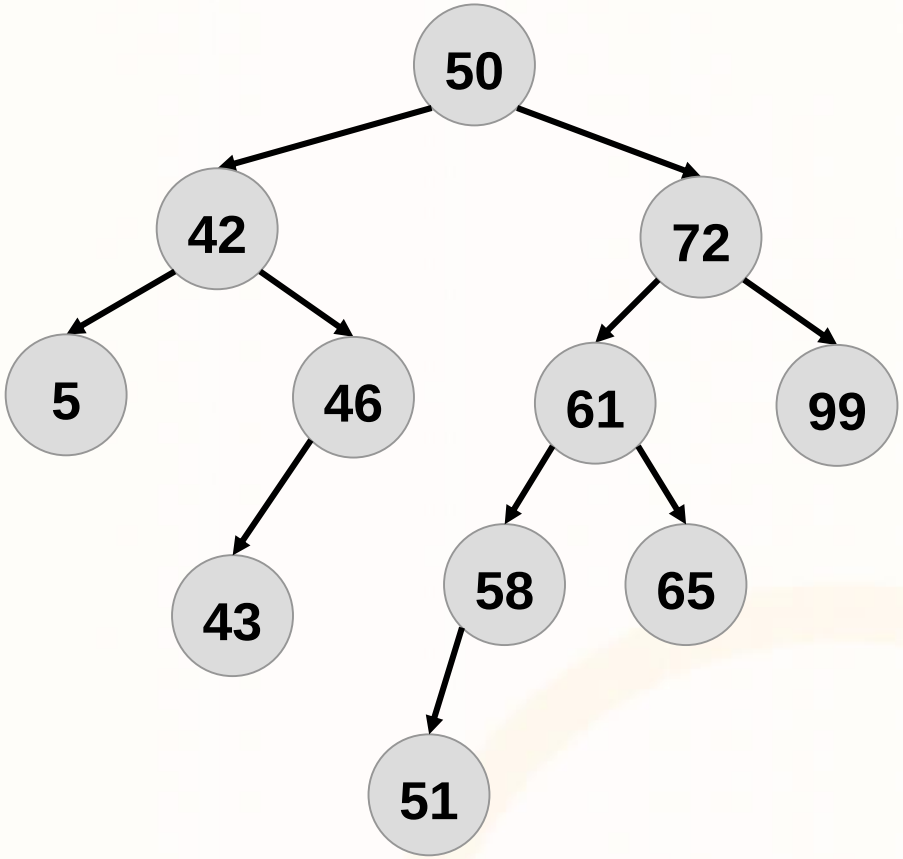
```
public void inOrder() { this.inOrder(this.head); }
private void inOrder(BinaryTreeNode<E> node) {
  if (node != null) {
    inOrder(node.left);
    System.out.print(node.key + " ");
    inOrder(node.right);
  }
}
```

- Εναλλακτικά της τύπωσης:
  - Queue<E> queue ως δομή αποθήκευσης των κλειδιών
  - Αντί System.out.print() → queue.enqueue()

```
public Queue<E> inOrder() {
        Queue<E> q = new Queue<E>();
        this.inOrder(this.head, q);
        return q;
}
private void inOrder(BinaryTreeNode<E> node, Queue<E> q) {
  if (node != null) {
    inOrder(node.left);
    q.enqueue(node.key);
    inOrder(node.right);
  }
}
```

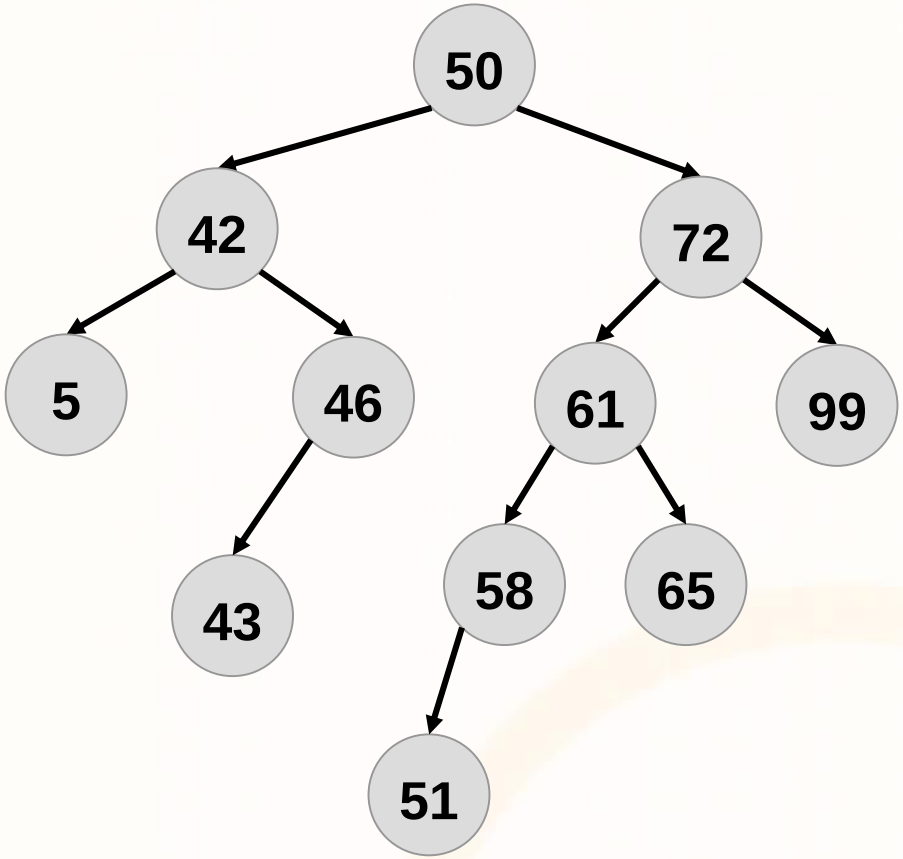
Δυαδικά Δέντρα και ΔΔΑ: Ασκήσεις

Πιο θα είναι το αποτέλεσμα της προθεματικής διάσχισης?



Δυαδικά Δέντρα και ΔΔΑ: Ασκήσεις

Πιο θα είναι το αποτέλεσμα της μεταθεματικής διάσχισης?



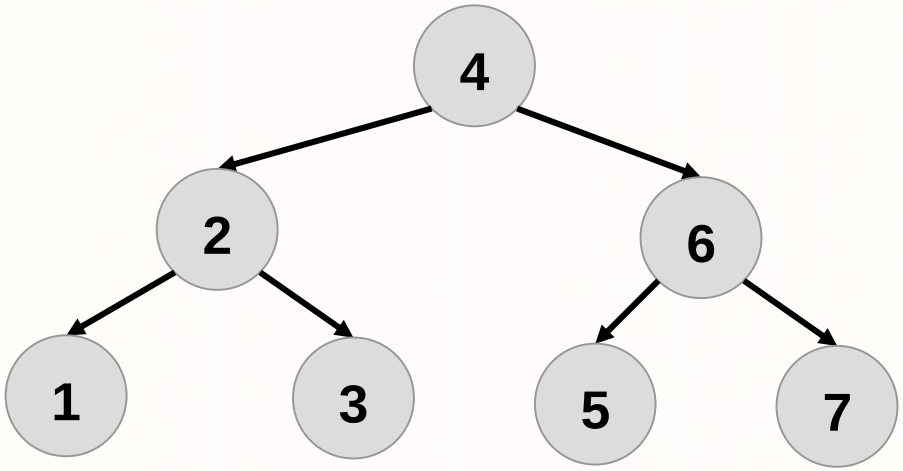
Δυαδικά Δέντρα και ΔΔΑ: Ασκήσεις

Ποιο είναι το ύψος ενός δυαδικού δέντρου όταν εισάξουμε τα στοιχεία 1,2,3,4,5,6,7 με αυτή τη σειρά;

Ποιος θα είναι ο χρόνος αναζήτησης χείριστης περίπτωσης στο πιο πάνω δέντρο σε σχέση με το n;

Δυαδικά Δέντρα και ΔΔΑ: Ασκήσεις

Με ποια σειρά πρέπει να εισάξουμε τα στοιχεία 1,2,3,4,5,6,7 ώστε να πάρουμε το πιο κάτω δέντρο;



Τι είναι το πιο πάνω δέντρο;