# Δένδρα

Νικολουτσόπουλος Σωτήριος
(sotirisnik@gmail.com )

# Δένδρα

Ορισμός γράφου:
Γράφος είναι μία δομή που αποτελείται από ένα σύνολο κόμβων(vertices) που συνδέονται μεταξύ τους με ένα σύνολο ακμών (edges).

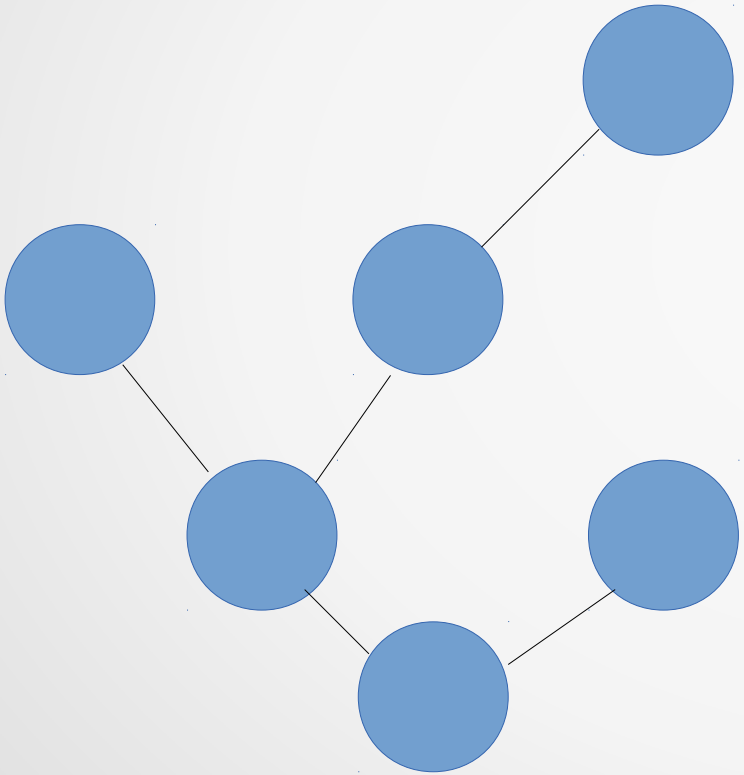
Ορισμός δένδρου:
Ένα συνεκτικό άκυκλο γράφημα λέγεται δένδρο.

Εναλλακτικός ορισμός δένδρου:
Ένας γράφος είναι δένδρο αν δύο οποιεσδήποτε κορυφές ενώνονται με ένα μοναδικό μονοπάτι.

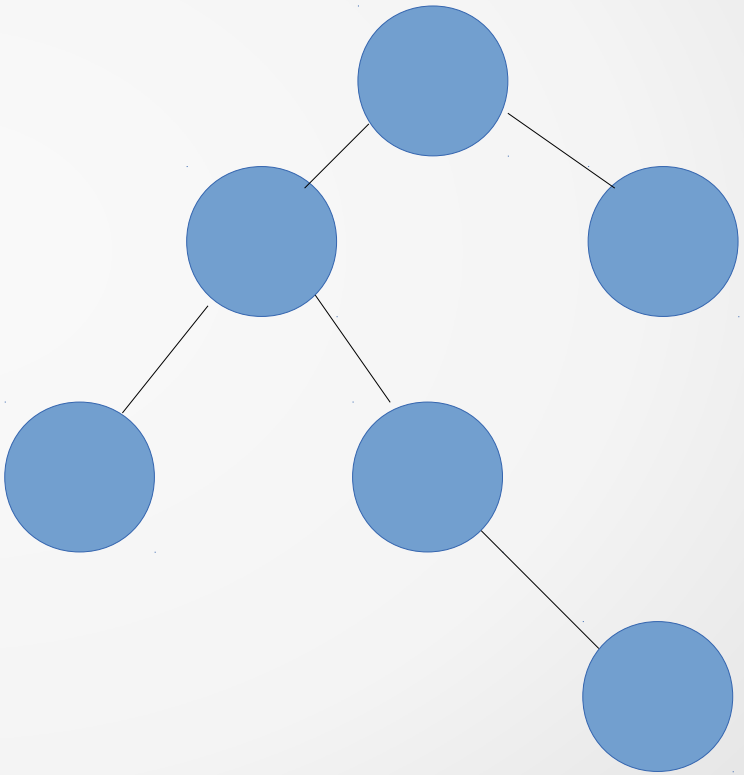
- Κάθε δένδρο περιέχει Ν κόμβους και Ν-1 ακμές.
- Είναι συνεκτικό αν για οποιουσδήποτε δύο κόμβους του υπάρχει μονοπάτι.
- Άκυκλο είναι όταν δεν περιέχει κλειστή διαδρομή μεταξύ δύο κόμβων( δηλαδή ξεκινώντας από μία κορυφή Α ακολουθώντας μία διαδρομή δεν μπορούμε να καταλήξουμε πάλι στον κόμβο Α ).

# Αναπαράσταση δένδρων

Φυσικός Τρόπος



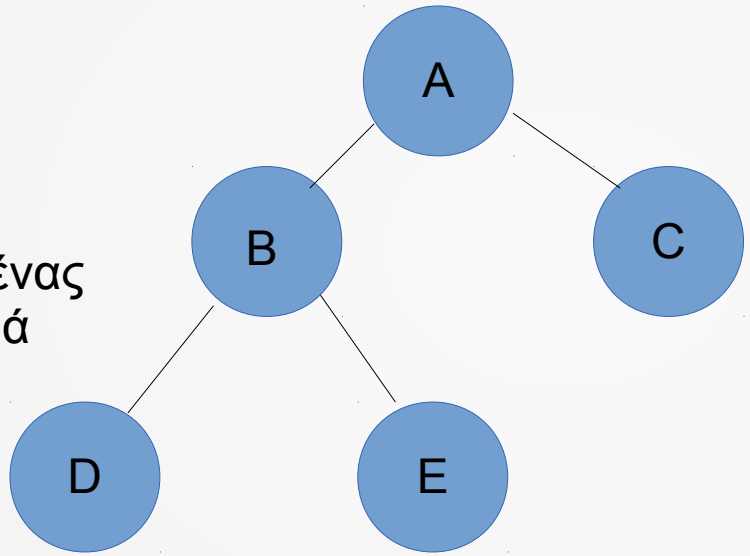
Στην πληροφορική



# Αναπαράσταση δένδρων

ρίζα

Η ρίζα είναι μοναδική και δεν έχει πρόγονο



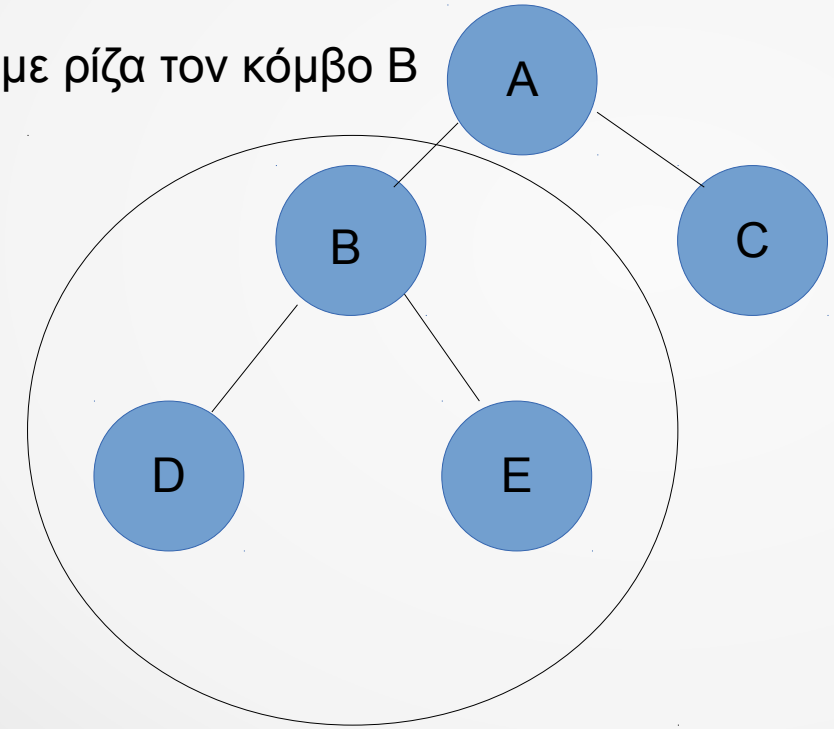

φύλλο

Φύλλο ονομάζεται ένας κόμβος χωρίς παιδιά

- Η ρίζα είναι ο κόμβος Α.
- Οι κόμβοι Β,C είναι παιδιά του κόμβου Α
- Ο κόμβος Α είναι πρόγονος των Β, C, D, E
- Οι κόμβοι C,D, E είναι φύλλα

# Αναπαράσταση δένδρων

Υποδένδρο με ρίζα τον κόμβο B



# Δυαδικά Δένδρα

Ορισμός:

Δυαδικό δένδρο είναι το κενό γράφημα, καθώς και κάθε δένδρο που περιέχει μία ρίζα τα υποδένδρα της οποίας είναι δυαδικά δένδρα – το αριστερό και το δεξί

# Δυαδικά δένδρα αναζήτησης

Δυαδικό δένδρο με της εξής ιδιότητες:

- Το αριστερό υποδένδρο ενός κόμβου περιέχει μονάχα κόμβους με κλειδιά μικρότερα από το κλειδί του κόμβου
- Το δεξί υποδένδρο ενός κόμβου περιέχει μονάχα κόμβους με κλειδιά μεγαλύτερα από το κλειδί του κόμβου
- Το αριστερό και το δεξί υποδένδρο είναι επίσης δυαδικά δένδρα αναζήτησης
- Δεν υπάρχουν διπλά κλειδιά.

# Αναπαράσταση ΔΔΑ σε C

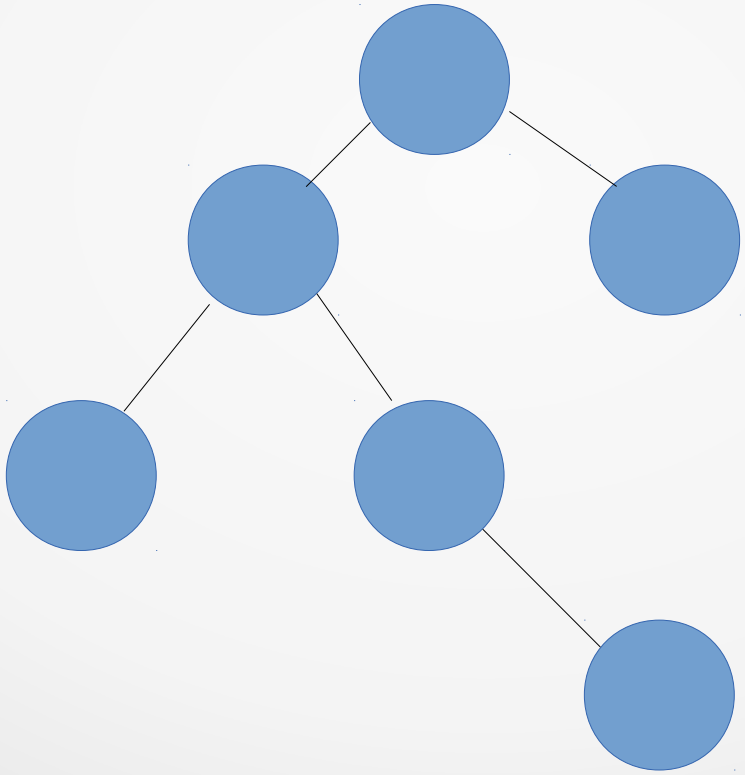
```
struct tree_element {
    int data, cnt;//τιμή, πλήθος παιδιών + του ίδιου του κόμβου
    tree_element *left, *right;//αριστερό, δεξί παιδί
};

typedef tree_element *tree;
```

# Διάτρεξη Δένδρων

- Προδιάταξη
- Ενδοδιάταξη
- Μεταδιάταξη
- Διάσχιση κατά σειρά επιπέδων

# Προδιάταξη

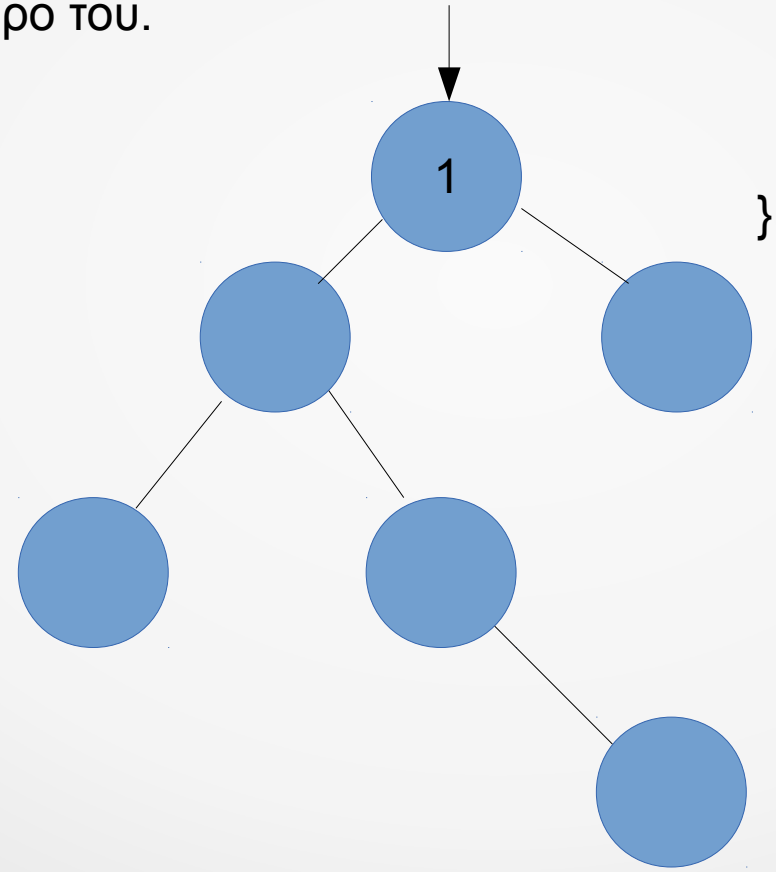
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.



```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

# Προδιάταξη

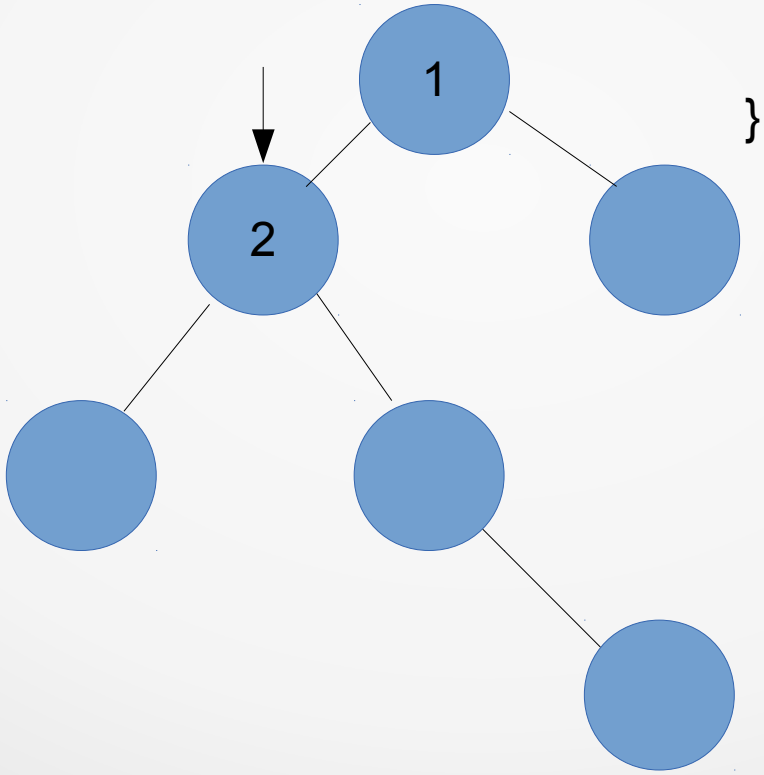
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

# Προδιάταξη

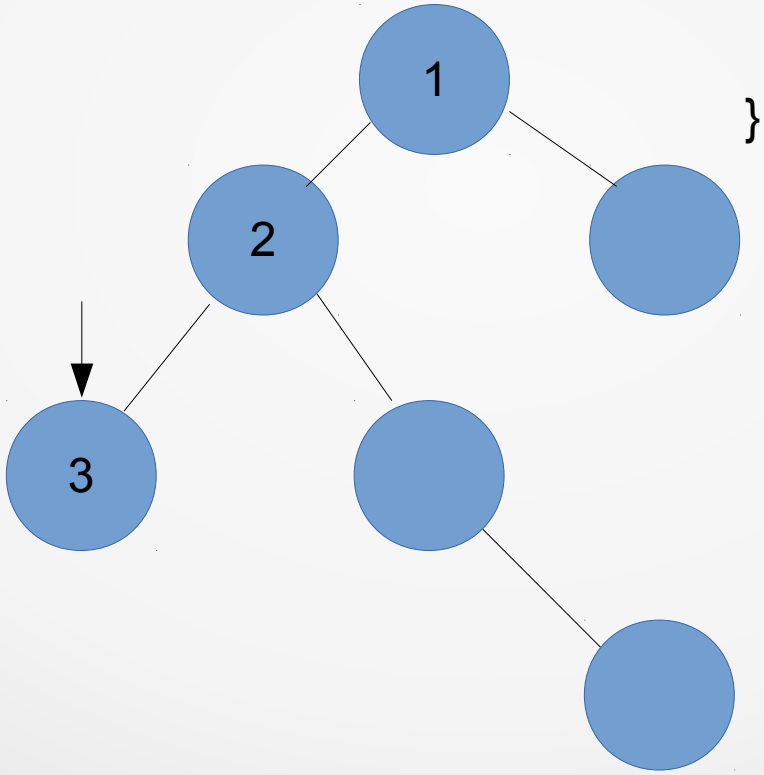
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

# Προδιάταξη

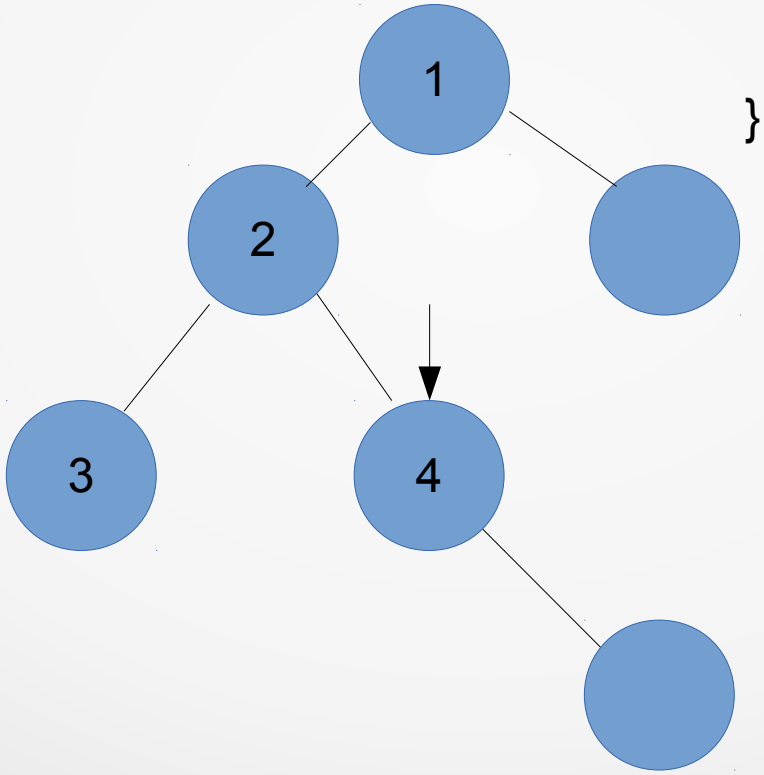
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

# Προδιάταξη

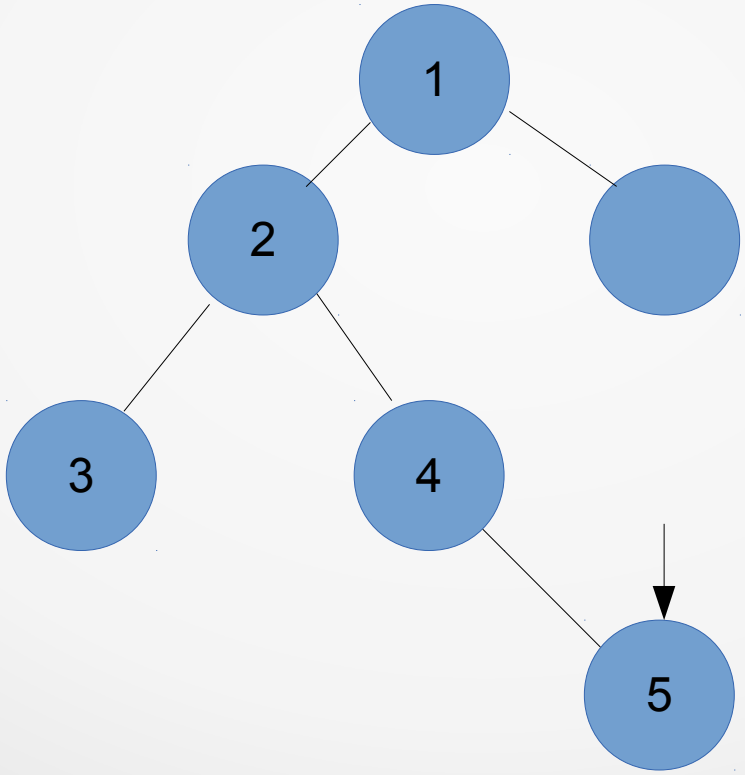
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

# Προδιάταξη

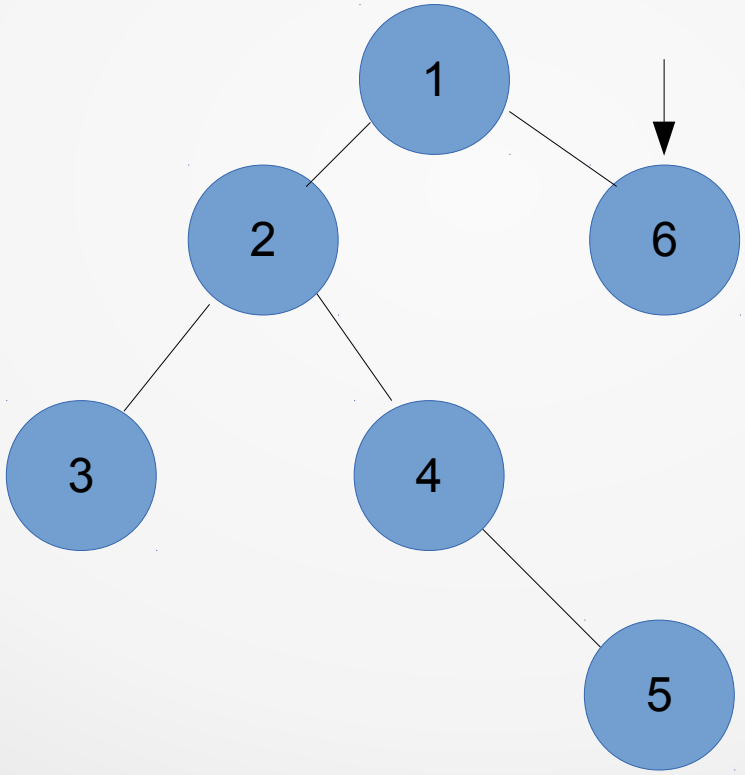
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

# Προδιάταξη

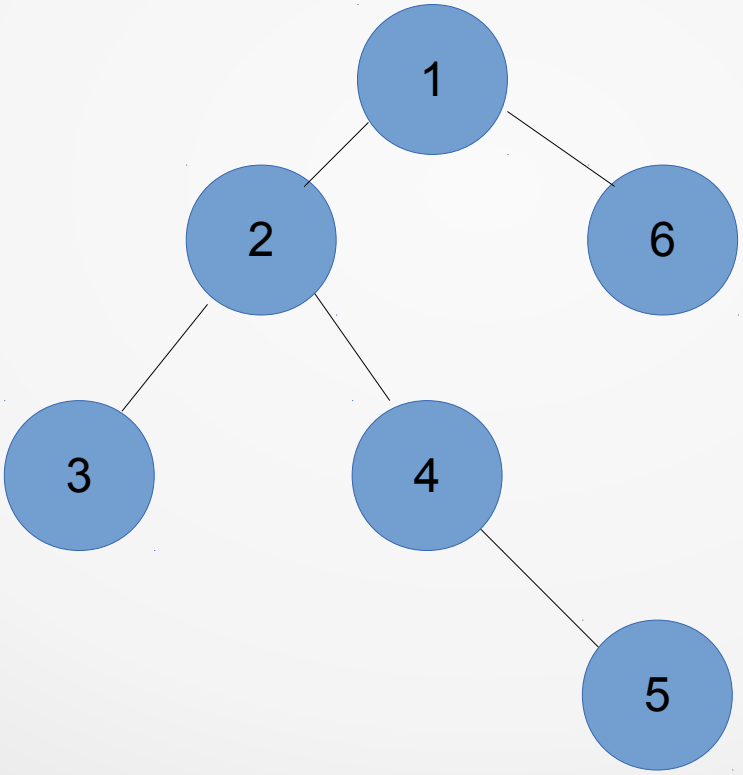
Επισκεπτόμαστε τον ίδιο τον κόμβο σε προδιατάξη προτού επισκεφθούμε σε προδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        Επισκέψου(T)
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
    }
}
```

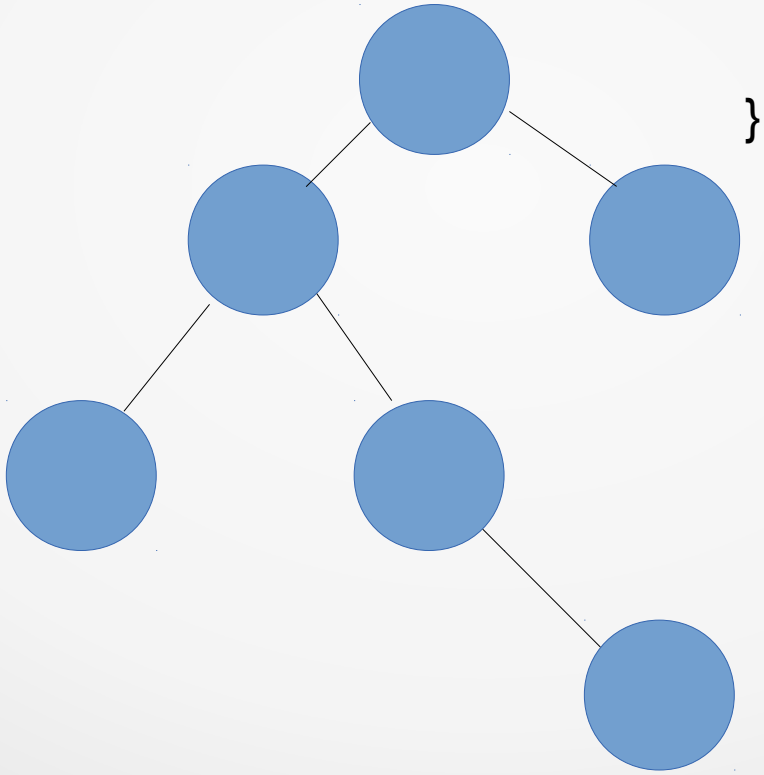
# Προδιάταξη

Αρίθμηση έπειτα από την διάτρεξη.



# Ενδοδιάταξη

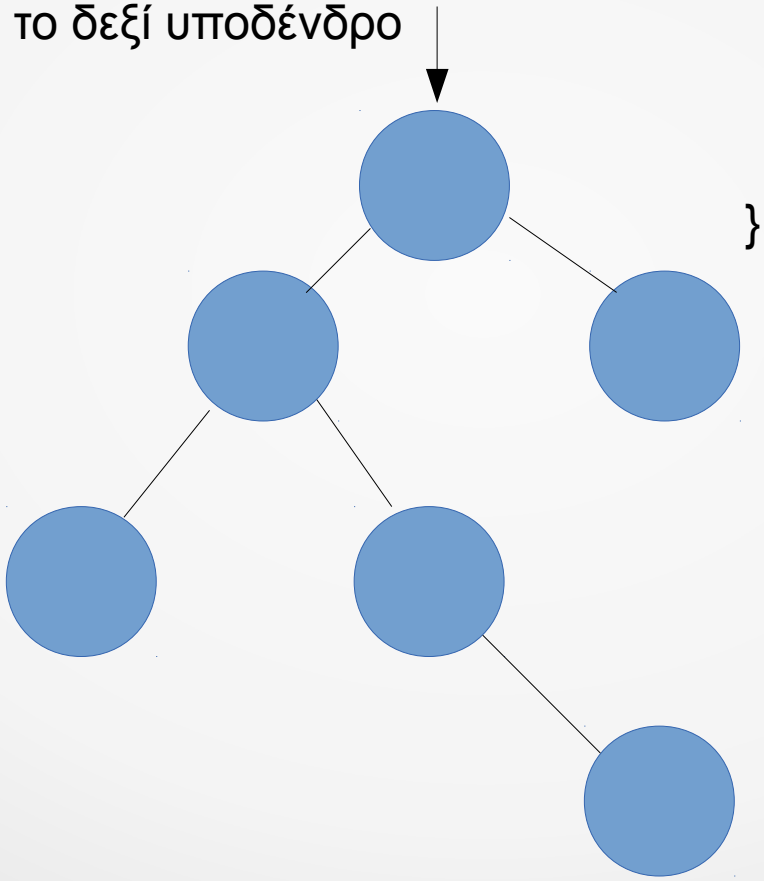
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.



```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

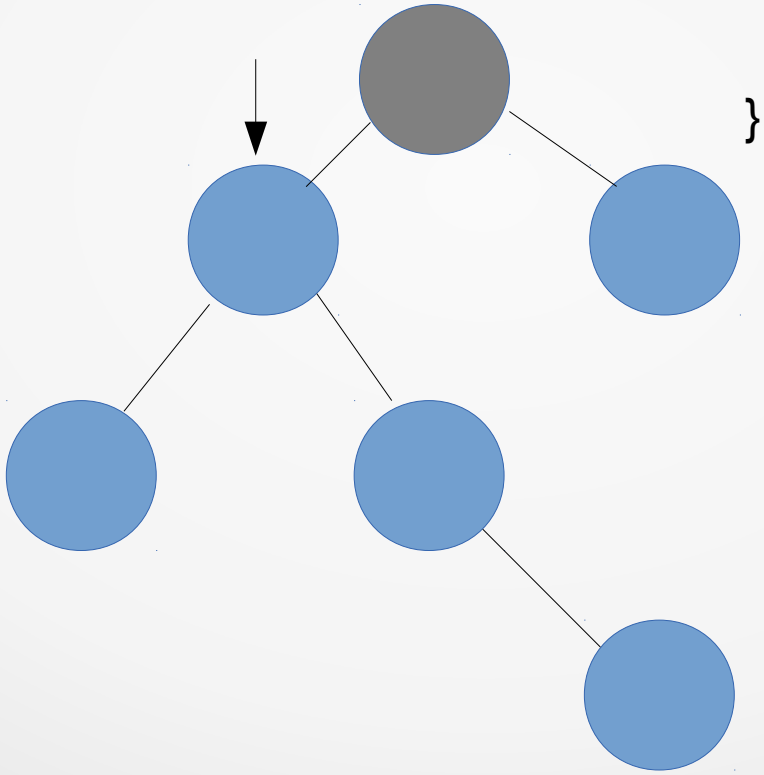
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.



```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

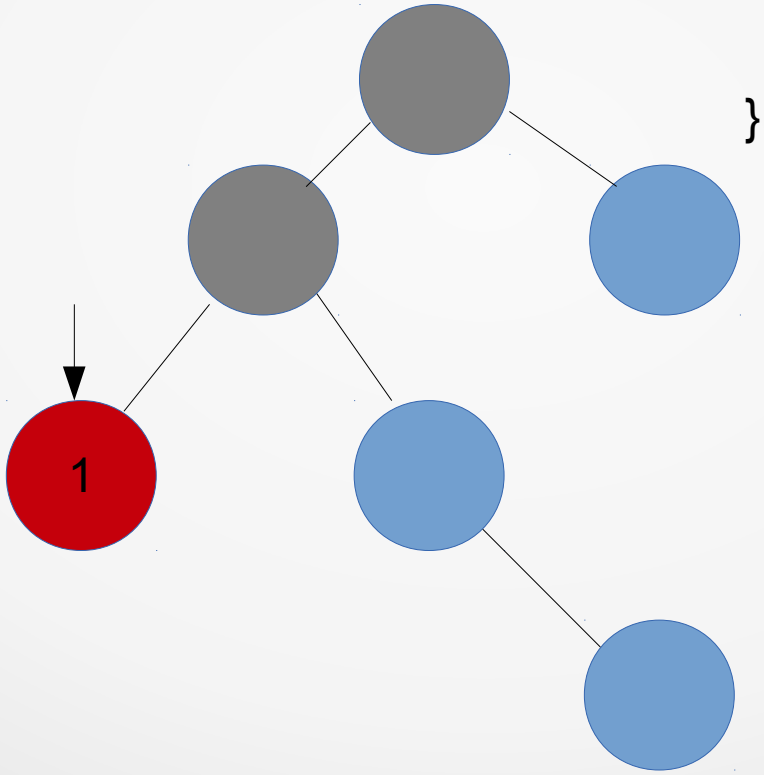
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.



```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

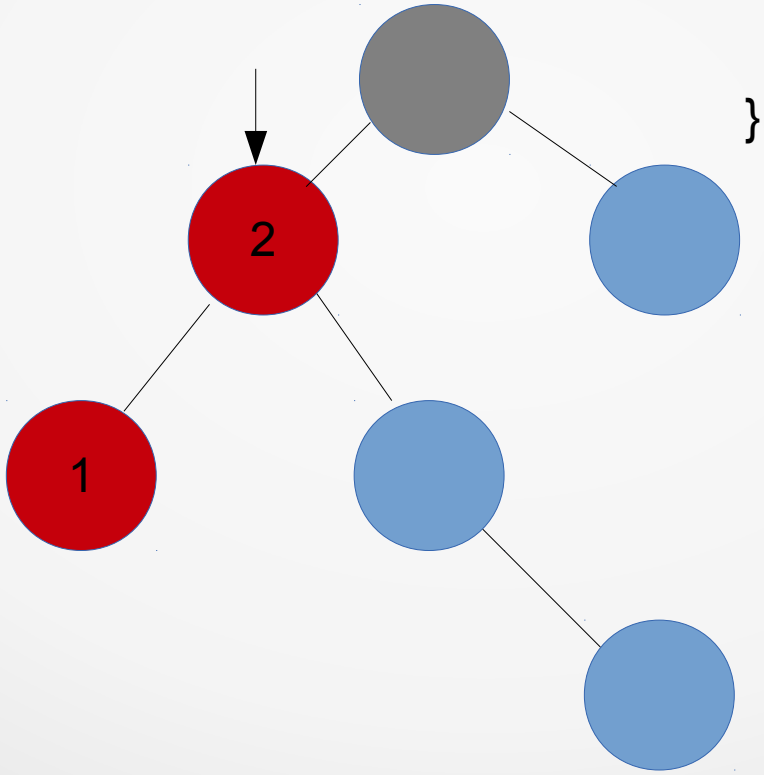
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.




```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

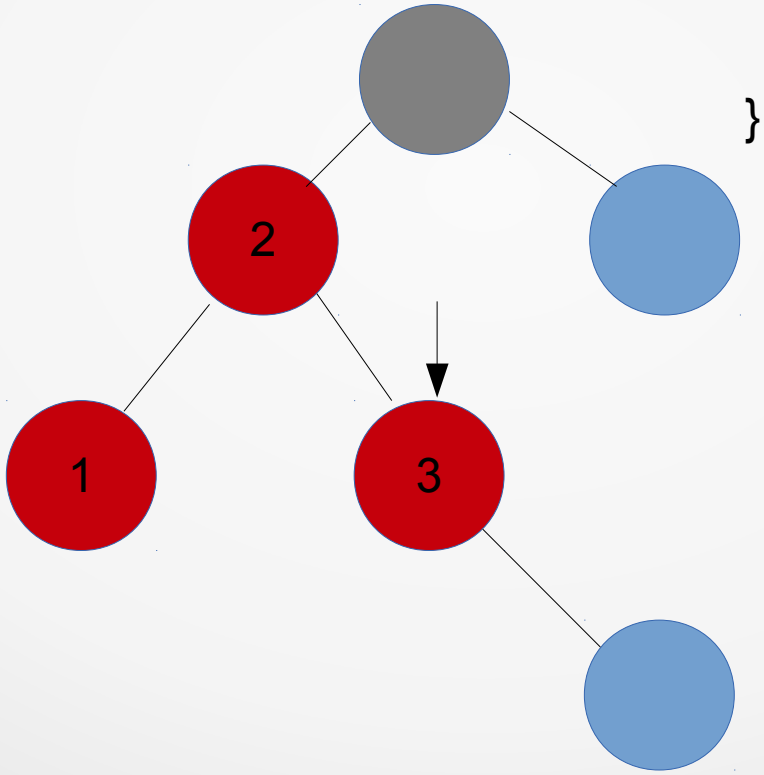
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.




```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.

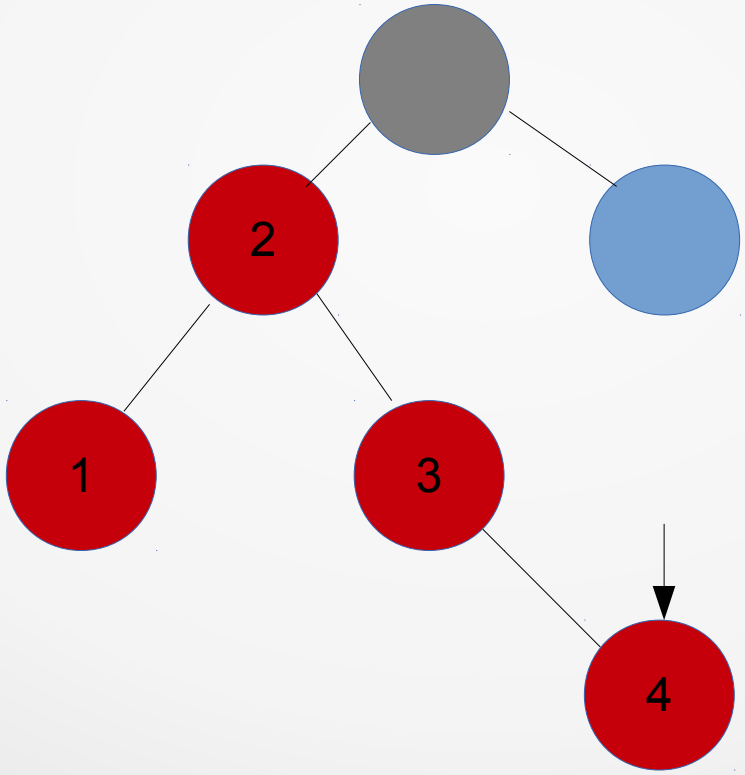



```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

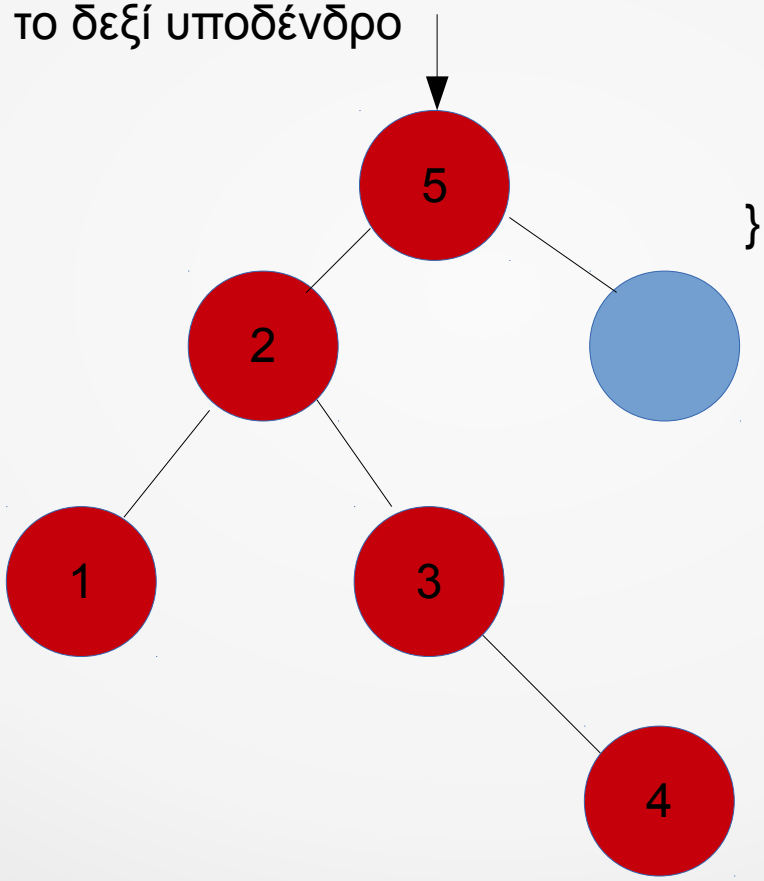
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.

```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```



# Ενδοδιάταξη

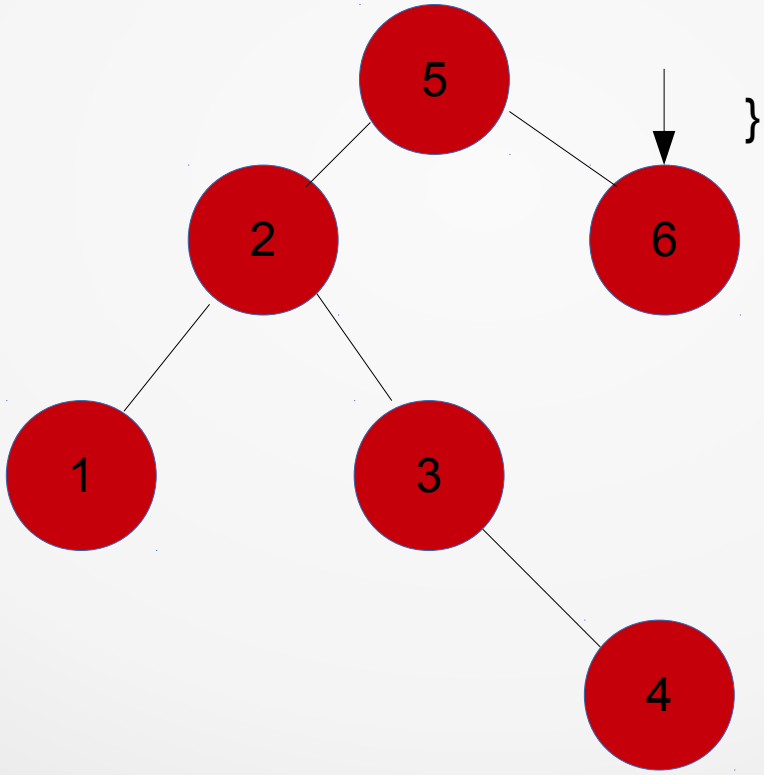
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.




```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

# Ενδοδιάταξη

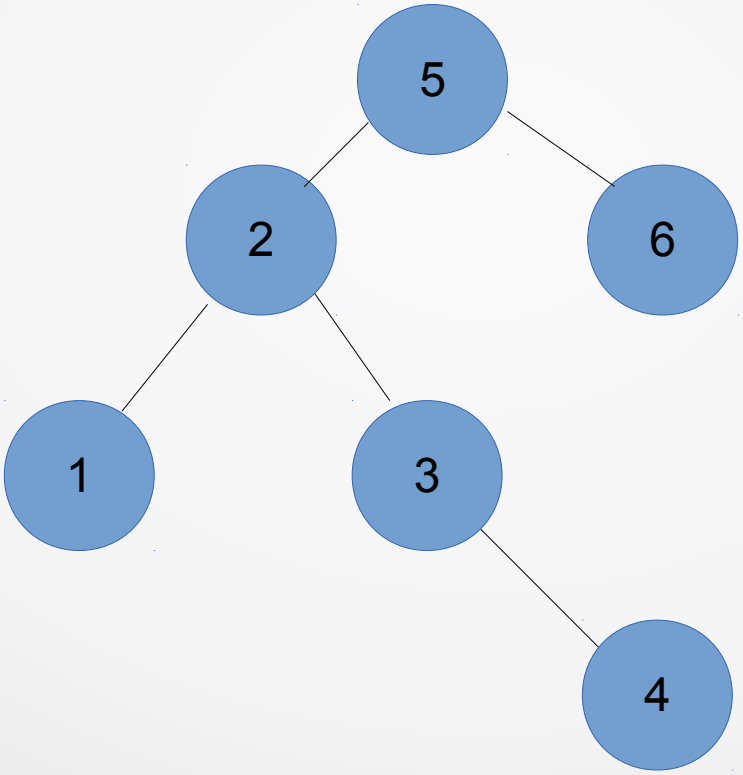
Επισκεπτόμαστε κάθε κόμβο, αφού επισκεφθούμε σε ενδοδιάταξη το αριστερό υποδένδρο του και πριν επισκεφθούμε το δεξί υποδένδρο του.




```
InOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        InOrder( Αριστερό_παιδί(T) )
        Επισκέψου(T)
        InOrder( Δεξί_παιδί(T) )
    }
}
```

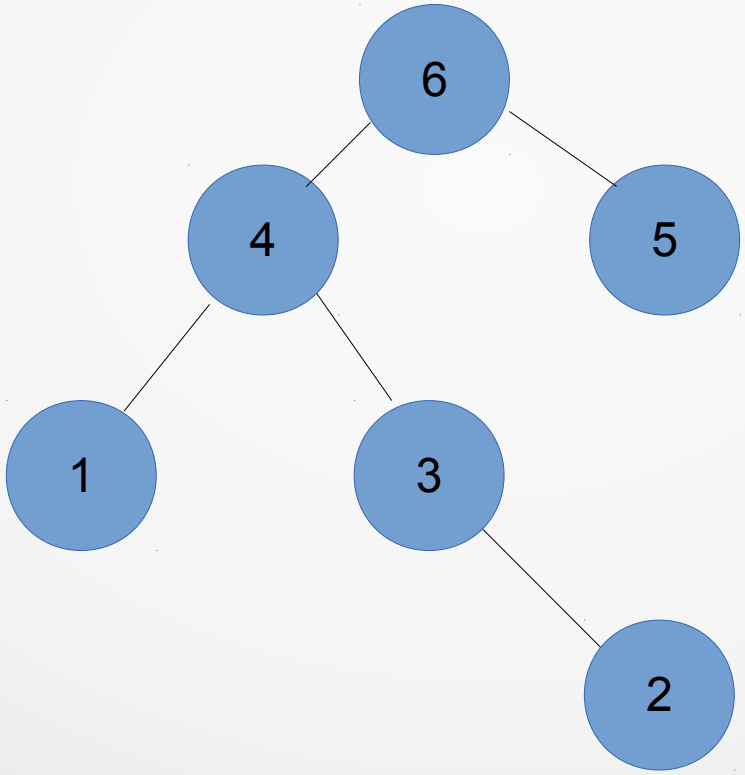
# Ενδοδιάταξη

Αρίθμηση έπειτα από την διάτρεξη.



# Μετάδιάταξη

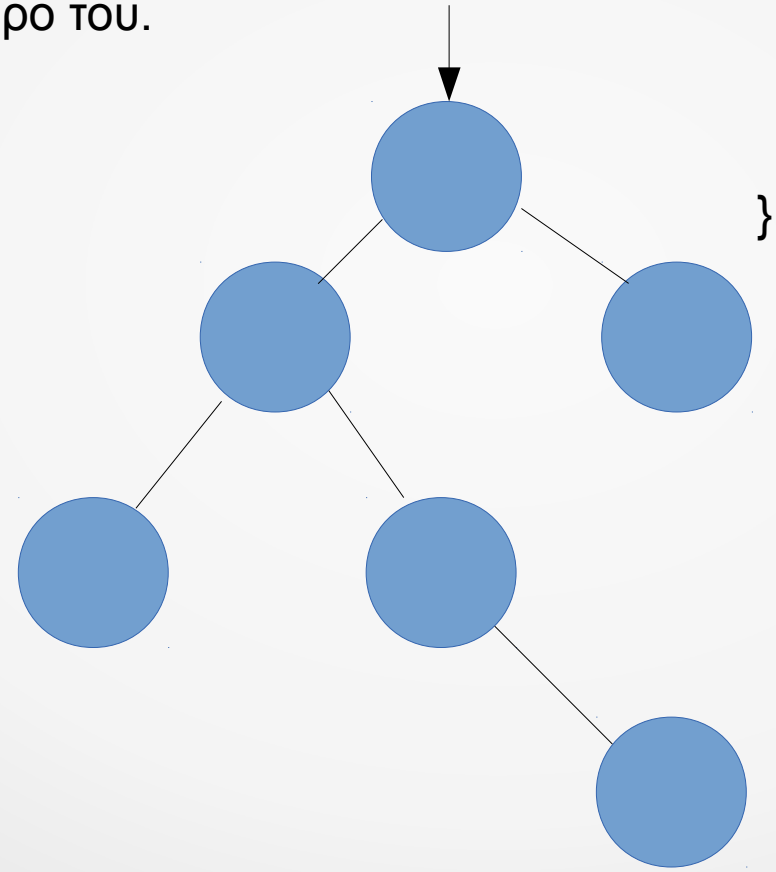
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

# Μετάδιάταξη

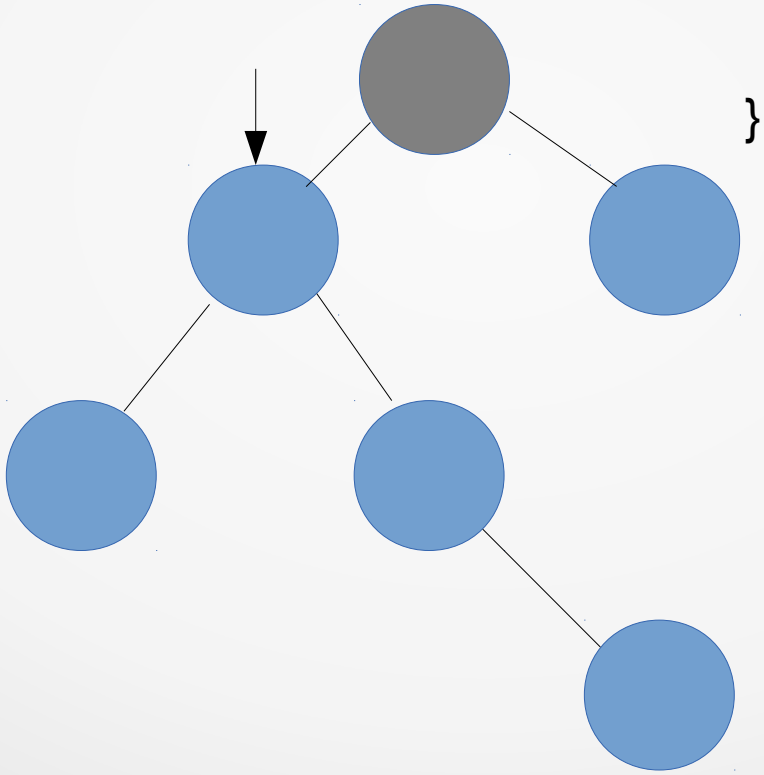
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.



```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

# Μετάδιάταξη

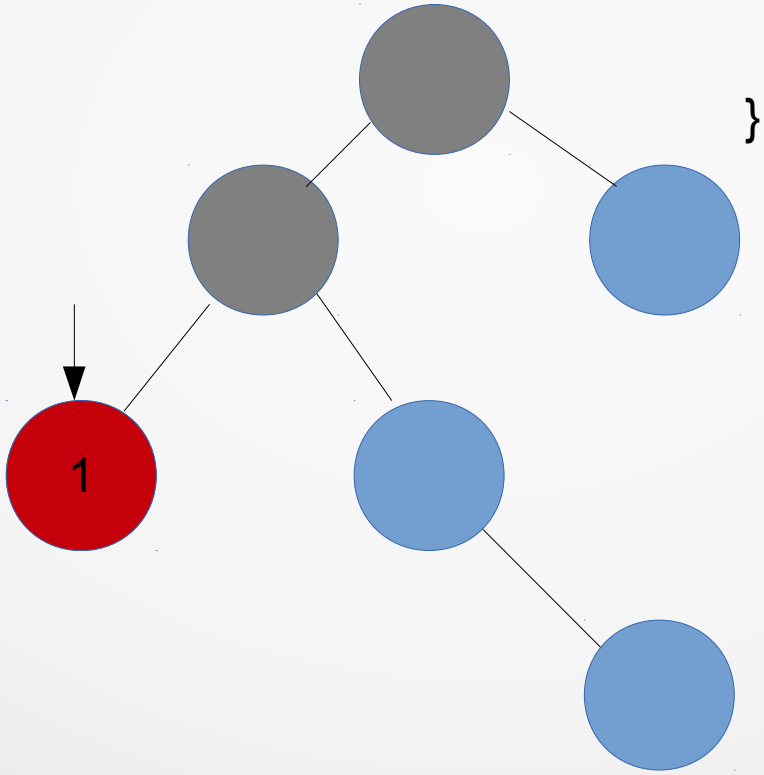
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.



```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

# Μετάδιάταξη

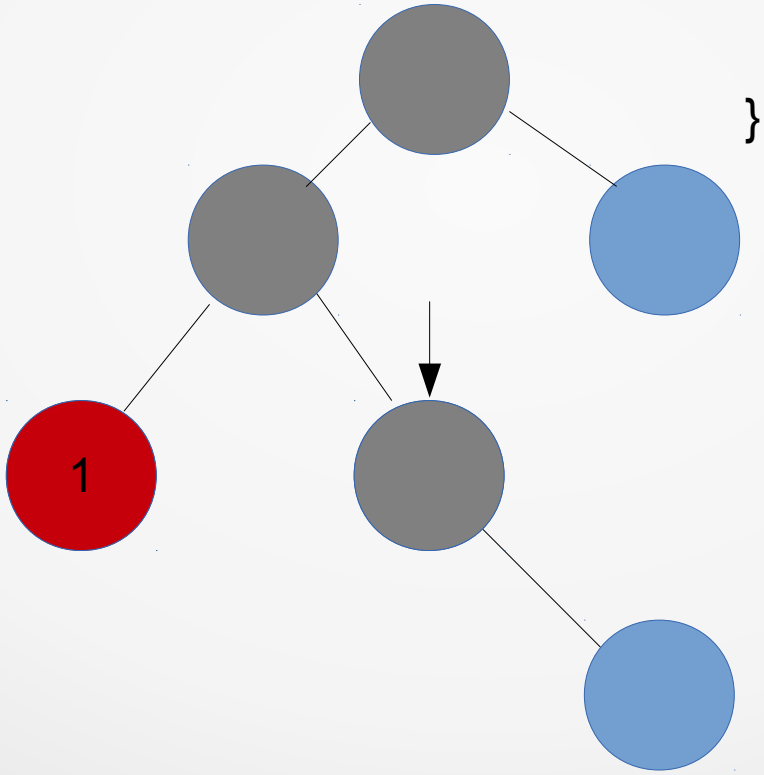
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

# Μετάδιάταξη

Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.

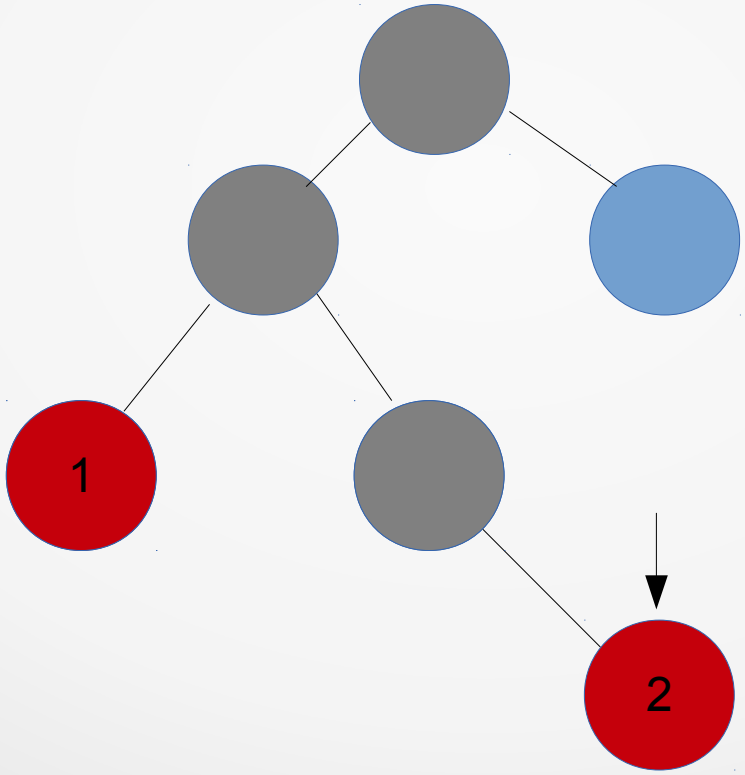



```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

# Μετάδιάταξη

Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.

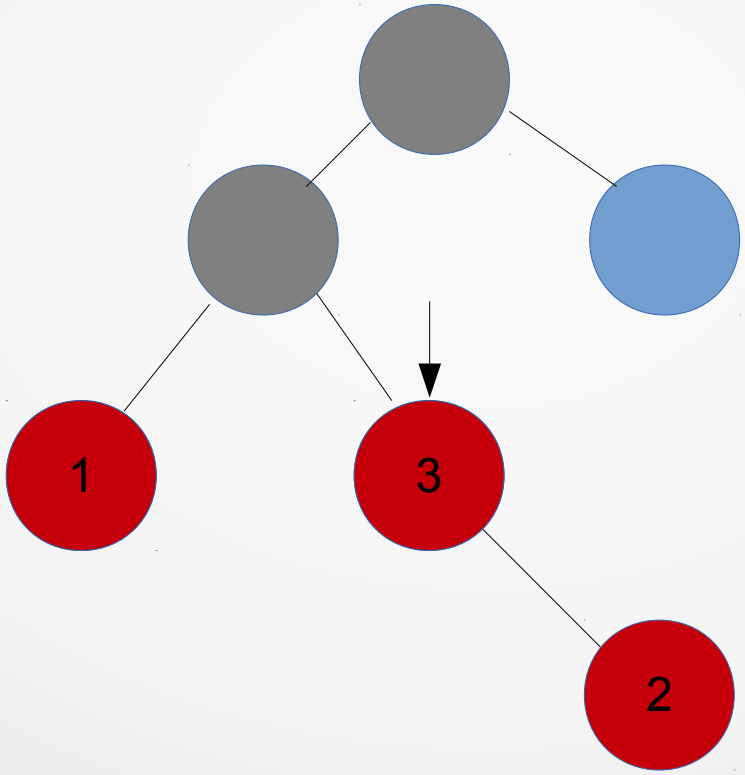
```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```



# Μετάδιάταξη

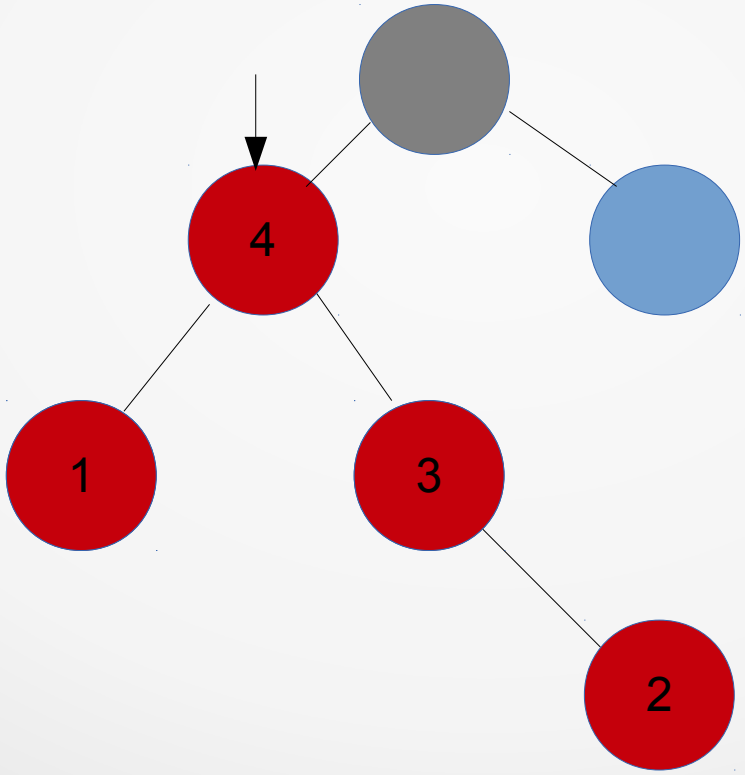
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.

```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```



# Μετάδιάταξη

Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.

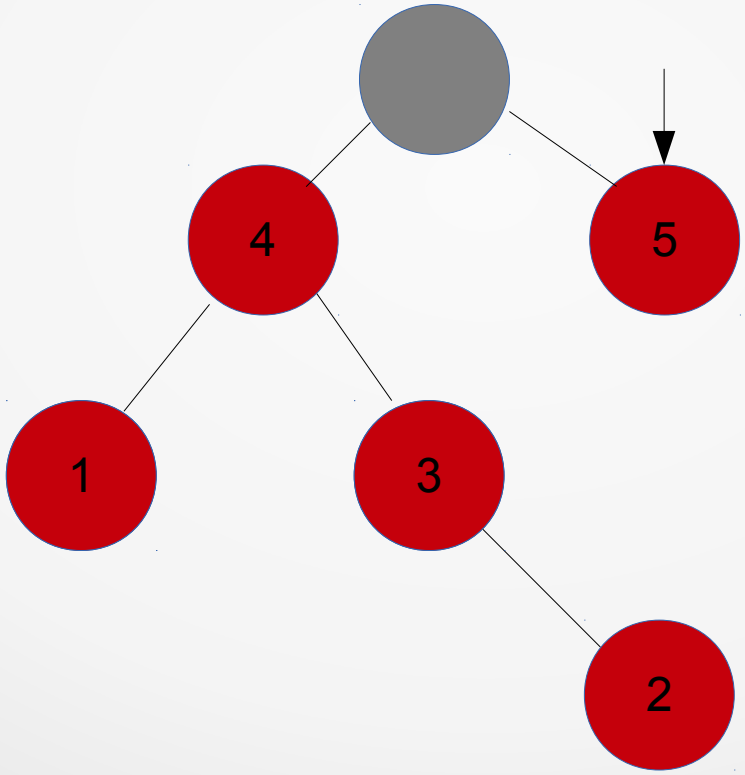



```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

# Μετάδιάταξη

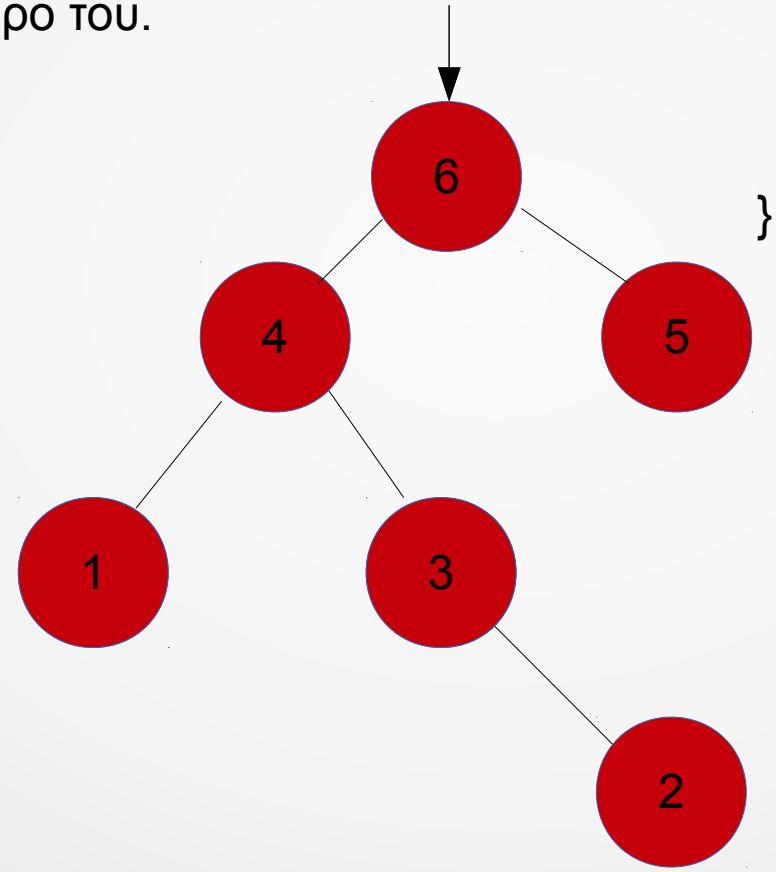
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.

```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```



# Μετάδιάταξη

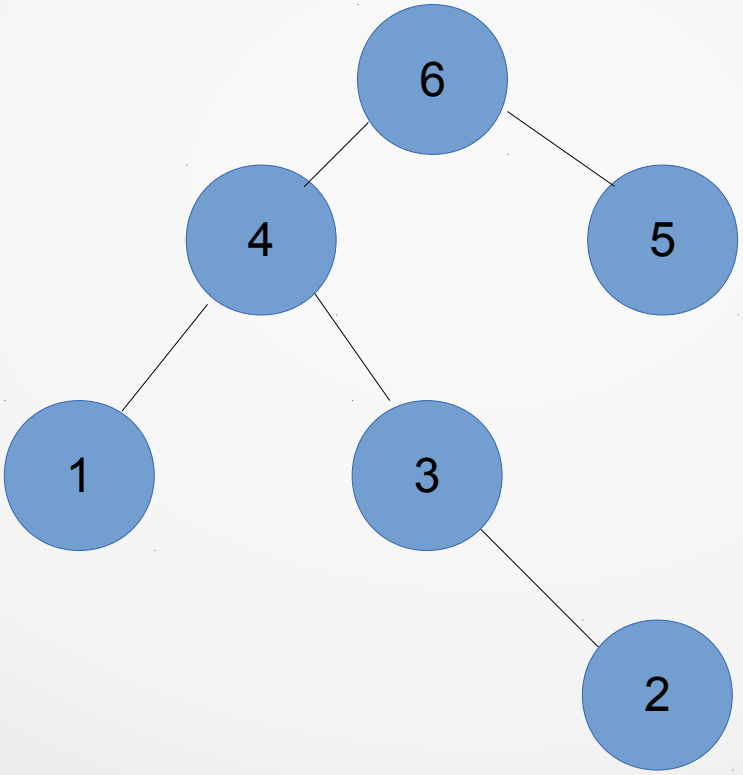
Επισκεπτόμαστε κάθε κόμβο, αφού έχουμε επισκεφθεί σε μεταδιάταξη το αριστερό και το δεξί υποδένδρο του.




```
PostOrder( Κόμβος T ) {
    Εάν ( T != κενό ) {
        PostOrder( Αριστερό_παιδί(T) )
        PostOrder( Δεξί_παιδί(T) )
        Επισκέψου(T)
    }
}
```

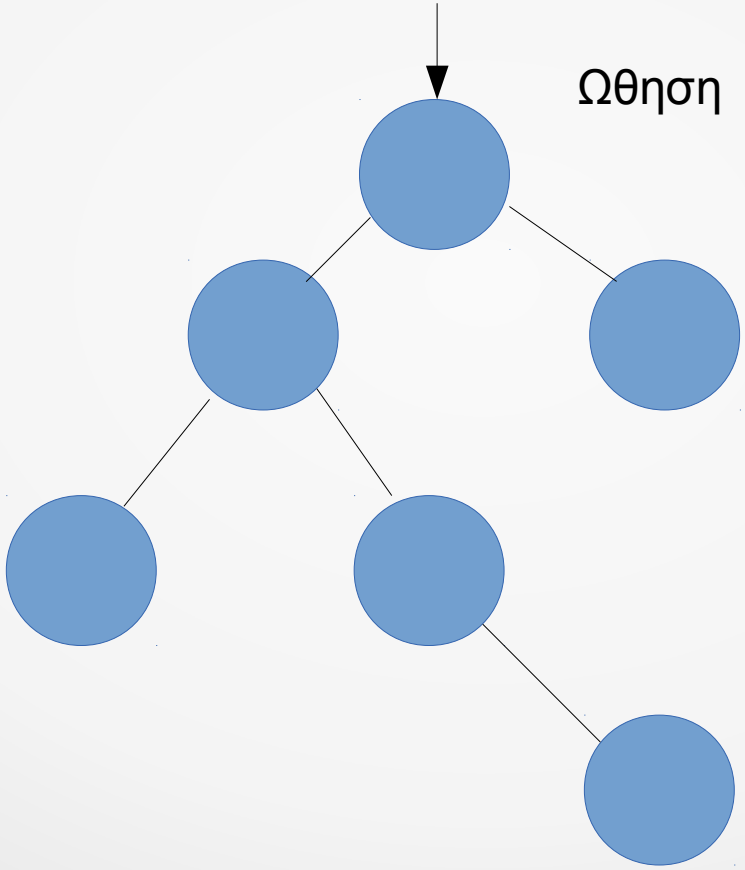
# Μετάδιάταξη

Αρίθμηση έπειτα από την διάτρεξη.



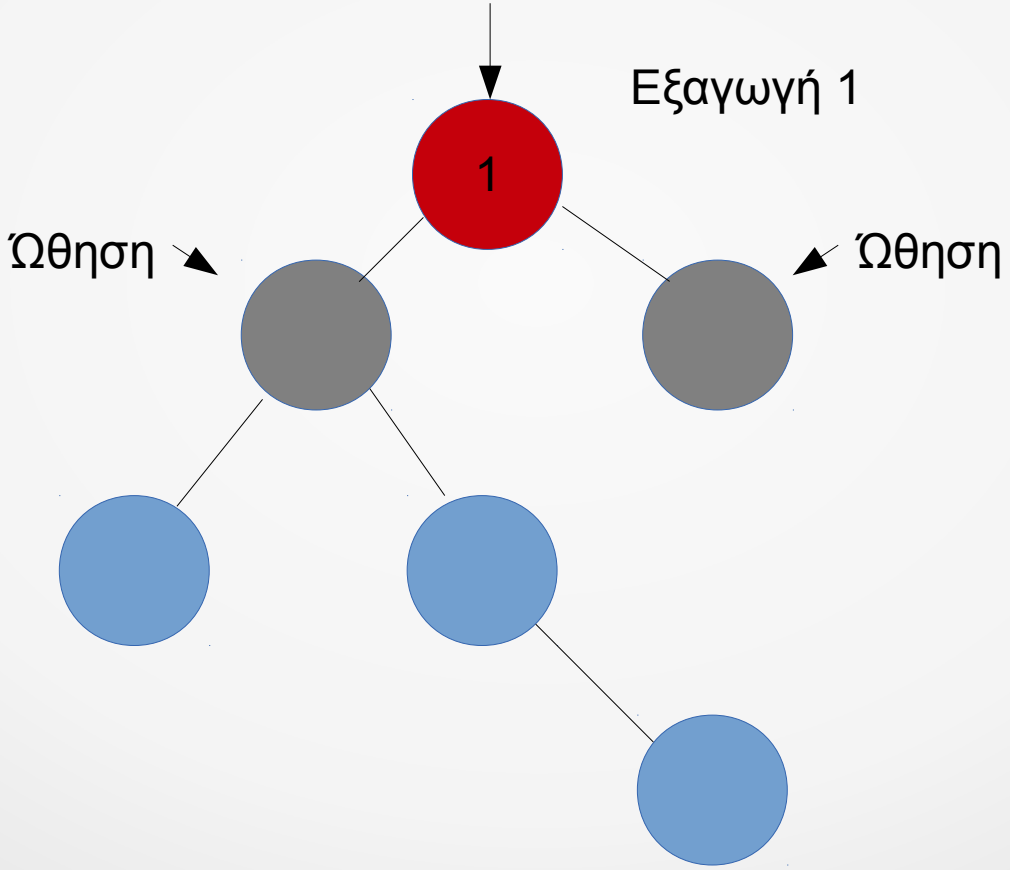
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



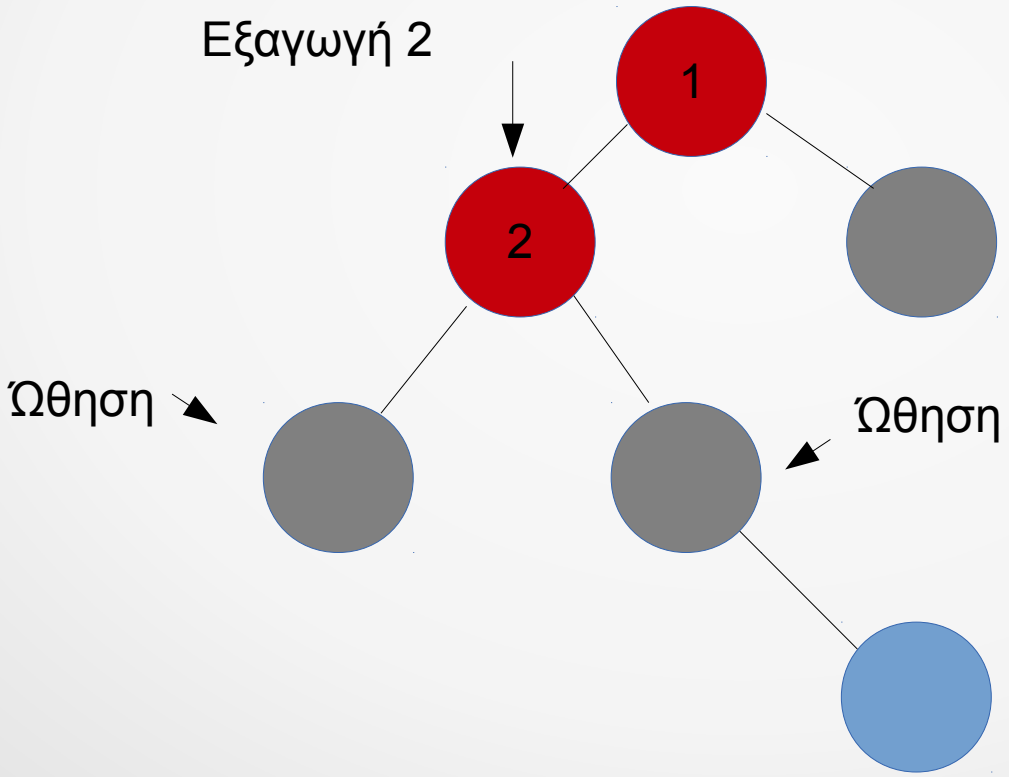
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



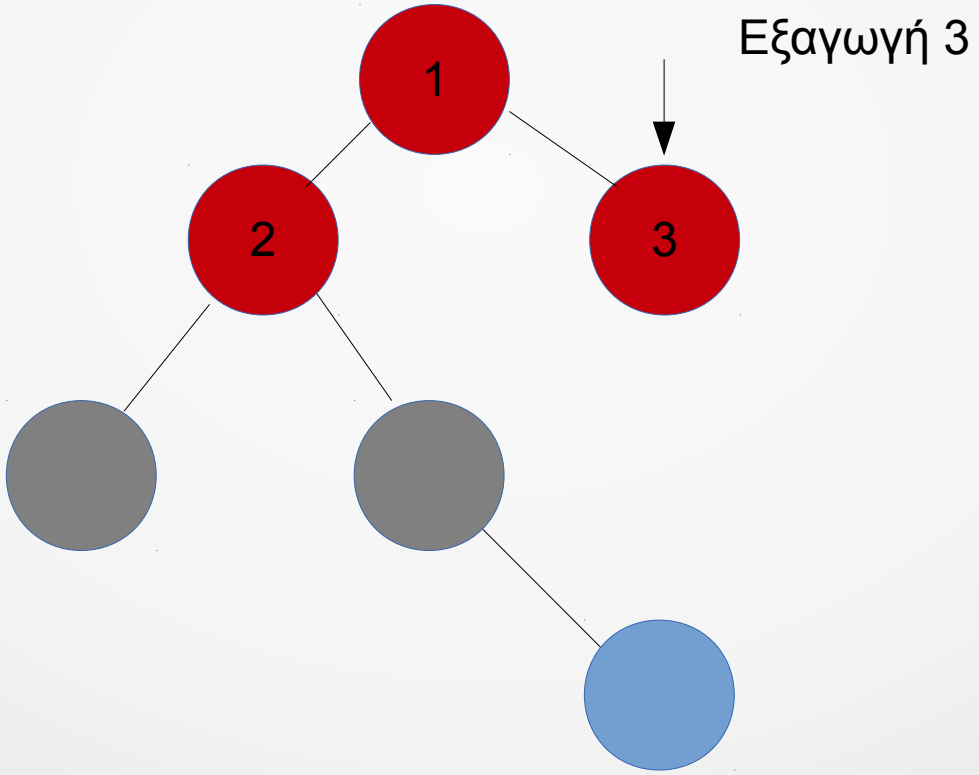
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



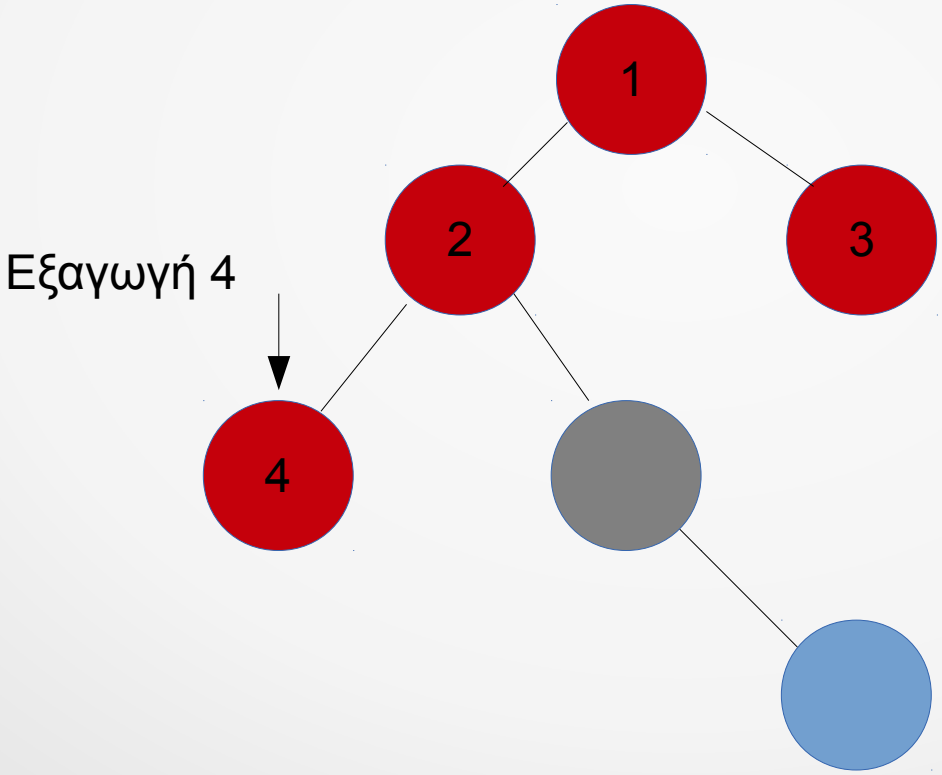
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



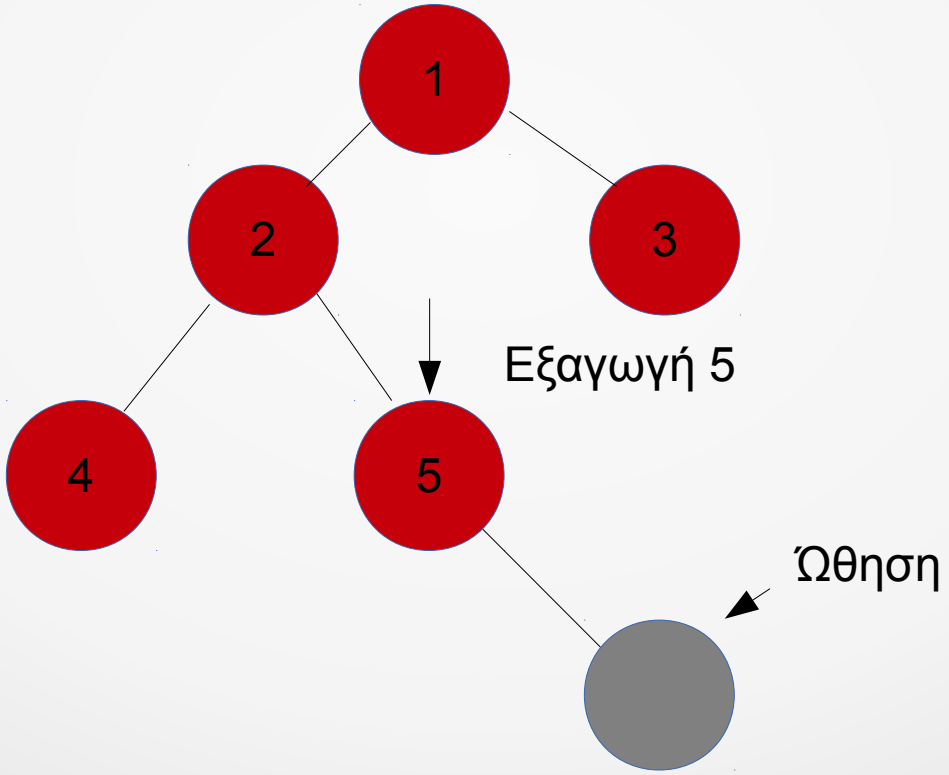
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



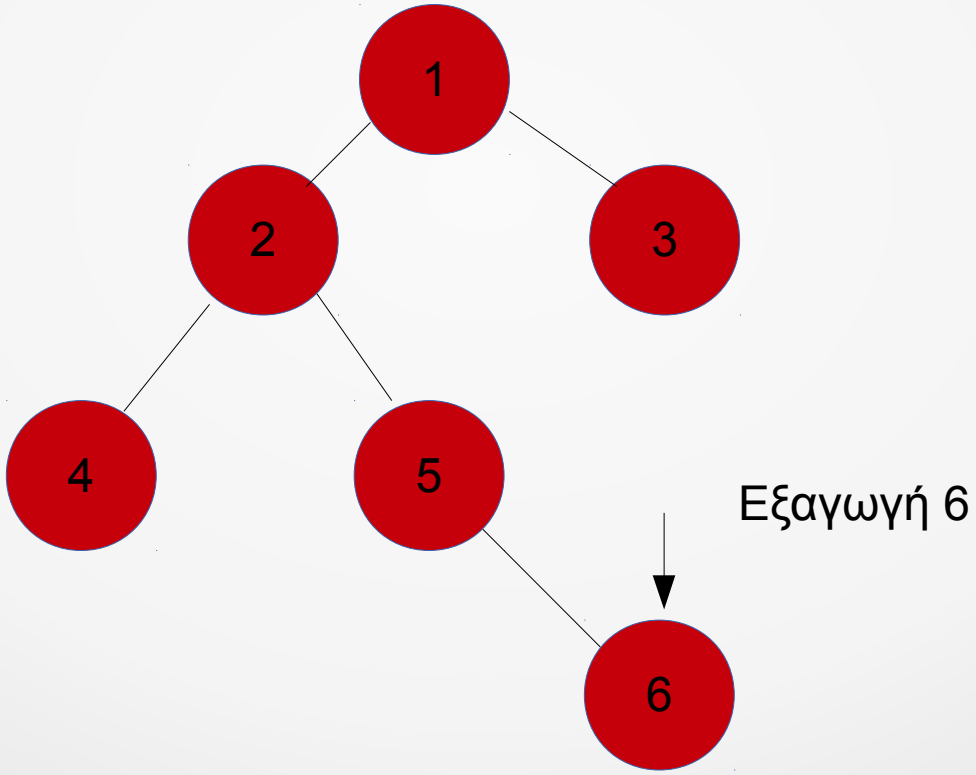
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



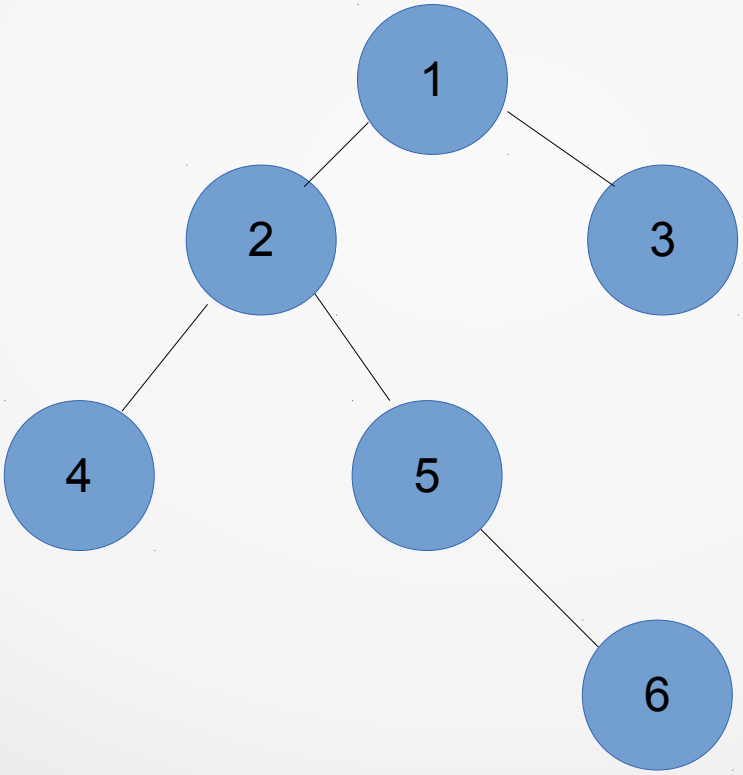
# Διάσχιση κατά σειρά επιπέδων

Αρχικά εισάγουμε την ρίζα σε μία ουρά. Έπειτα όσο η ουρά δεν είναι κενή
για κάθε κόμβο που επισκεπτόμαστε, εισάγουμε τους γείτονες του στην ουρά κ.ο.κ..



# Διάσχιση κατά σειρά επιπέδων

Αρίθμηση έπειτα από την διάσχιση κατά σειρά επιπέδων.

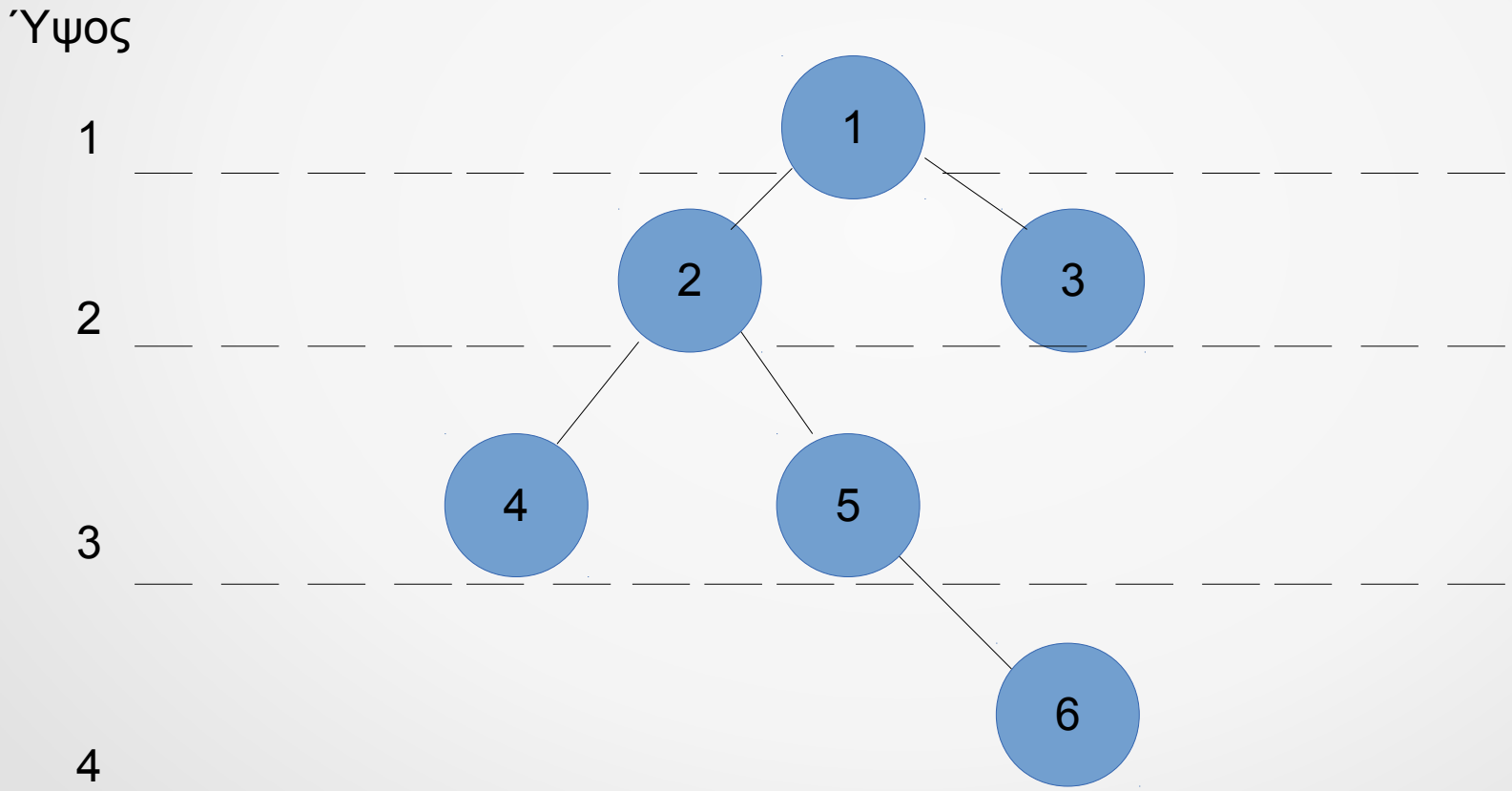


# ΄Υψος δένδρου

Το ύψος ενός κόμβου είναι το μήκος του μακρινότερου μονοπατιού ξεκινώντας από αυτόν τον κόμβο και φτάνοντας σε ένα φύλλο.

Το ύψος της ρίζας είναι το ύψος του δένδρου.

# Ύψος δένδρου



# Κώδικας σε C

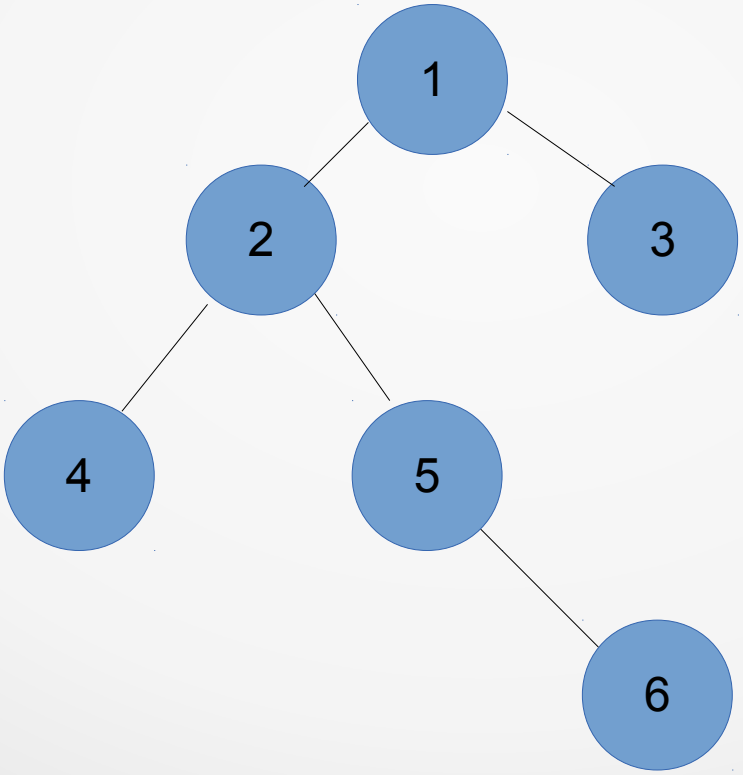
```
int height (tree t ) {
    if ( t == NULL ) {
        return 0;
    }else {
        return max( height(t->left), height(t->right) ) + 1;
    }
}
```

# Βάρος δένδρου

Το βάρος ενός κόμβου είναι το πλήθος των υπο κόμβων του.

Το βάρος της ρίζας είναι το πλήθος όλων των υπο κόμβων.

# Βάρος δένδρου




| Κόμβος | Βάρος |
|---|---|
| 1 | 5 |
| 2 | 3 |
| 3 | 0 |
| 4 | 0 |
| 5 | 1 |
| 6 | 0 |

# Κώδικας σε C

```
int weight( tree t ) {
    if ( t == NULL ) {
        return 0;
    }else {
        return weight(t->left) + weight(t->right) + 1;
    }
}
```

# Λειτουργίες Δυαδικού Δένδρου Αναζήτησης

- Ελάχιστο
- Μέγιστο
- Αναζήτηση
- Αναζήτηση K-οστου
- Εισαγωγή
- Διαγραφή

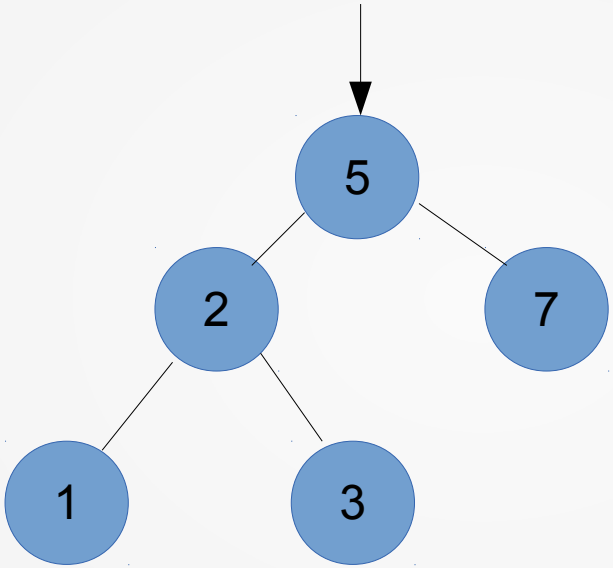
Όλες οι λειτουργίες έχουν πολυπλοκότητα O( log(N) ), σε περίπτωση όμως που το δένδρο δεν είναι ισορροπημένο μπορούμε να έχουμε O ( N )

# Ελάχιστο

Υπενθύμιση:

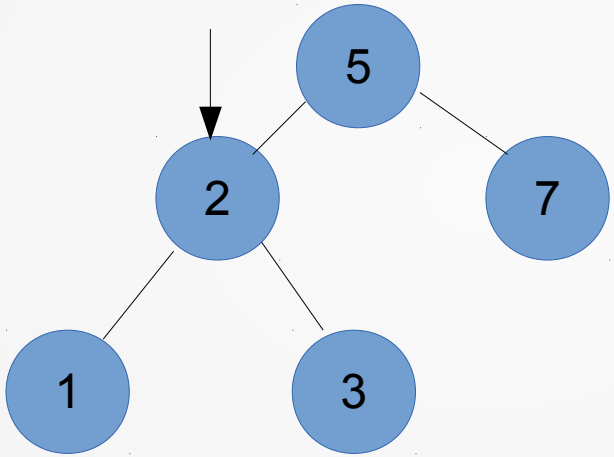
- Το αριστερό υποδένδρο ενός κόμβου περιέχει μονάχα κόμβους με κλειδιά μικρότερα από το κλειδί του κόμβου.
  Οπότε ξεκινώντας από την ρίζα, μεταβαίνουμε επαναληπτικά στο αριστερό παιδί του εώς ότου ο κόμβος που ελέγχουμε να έχει το αριστερό παιδί του κενό.

# Ελάχιστο - παράδειγμα




```
FindMin( Κόμβος T ) {
    Όσο ( Αριστερό_παιδί(T) != κενό ) {
        T = Αριστερό_παιδί(T)
    }
    #Απάντηση T
}
```

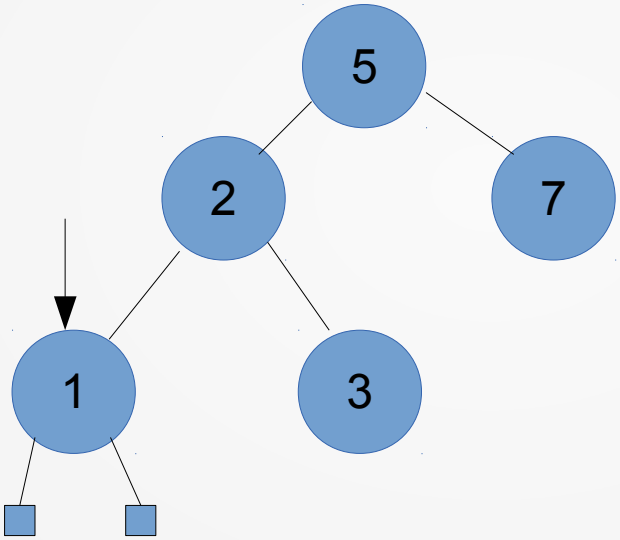
# Ελάχιστο - παράδειγμα




```
FindMin( Κόμβος T ) {
    Όσο ( Αριστερό_παιδί(T) != κενό ) {
        T = Αριστερό_παιδί(T)
    }
    #Απάντηση T
}
```

# Ελάχιστο - παράδειγμα

```
FindMin( Κόμβος T ) {
    Όσο ( Αριστερό_παιδί(T) != κενό ) {
        T = Αριστερό_παιδί(T)
    }
    #Απάντηση T
}
```




Το αριστερό παιδί του κόμβου είναι κενό, οπότε σταματάμε την αναζήτηση.

- αναπαριστά τον κενό κόμβο

# Κώδικας σε C

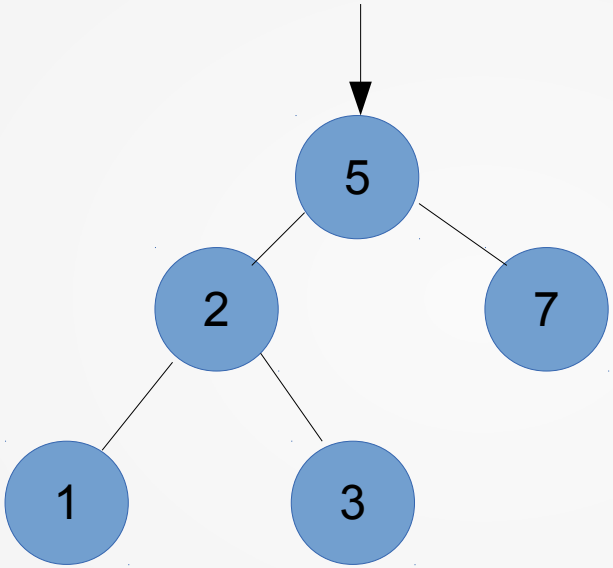
```
tree FindMin (tree t ) {
    while ( t->left != NULL ) {
        t = t->left;
    }
    return t;
}
```

# Μέγιστο

Υπενθύμιση:

- Το δεξιό υποδένδρο ενός κόμβου περιέχει μονάχα κόμβους με κλειδιά μεγαλύτερα από το κλειδί του κόμβου.
  Οπότε ξεκινώντας από την ρίζα, μεταβαίνουμε επαναληπτικά στο δεξιό παιδί του εώς ότου ο κόμβος που ελέγχουμε να έχει το δεξιό παιδί του κενό.

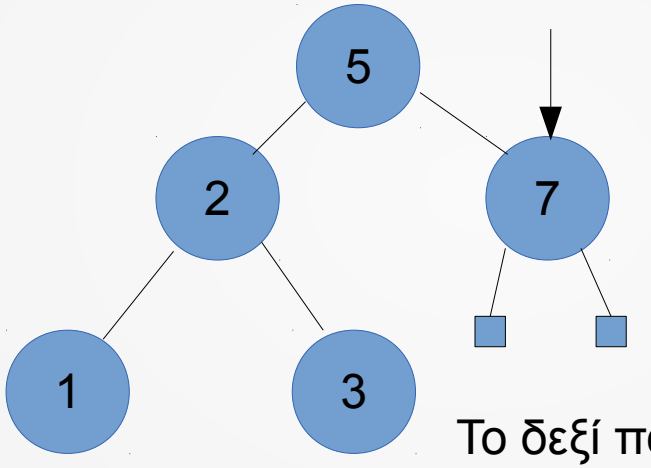
# Μέγιστο - παράδειγμα




```
FindMin( Κόμβος T ) {
    Όσο ( Αριστερό_παιδί(T) != κενό ) {
        T = Αριστερό_παιδί(T)
    }
    #Απάντηση T
}
```

# Μέγιστο - παράδειγμα

```
FindMax( Κόμβος T ) {
    Όσο ( Δεξι_παιδί(T) != κενό ) {
        T = Δεξι_παιδί(T)
    }
    #Απάντηση T
}
```




Το δεξί παιδί του κόμβου είναι κενό, οπότε σταματάμε την αναζήτηση.

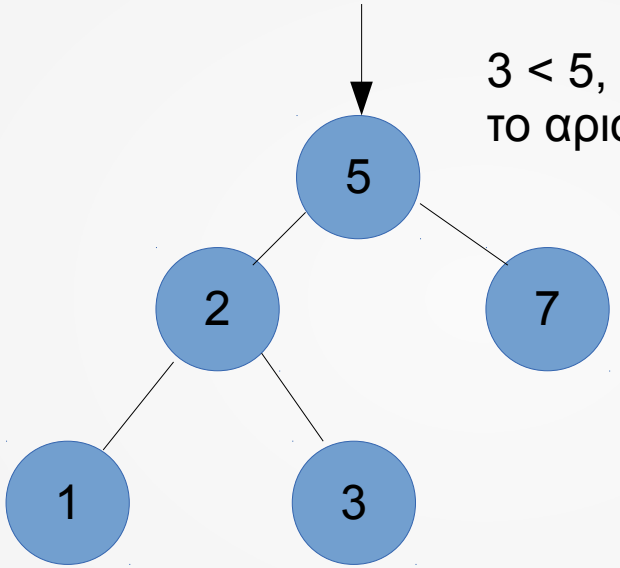
αναπαριστά τον κενό κόμβο

# Κώδικας σε C

```
tree FindMax ( tree t ) {
    while ( t->right != NULL ) {
        t = t->right;
    }
    return t;
}
```

# Αναζήτηση - Παράδειγμα 1

Εύρευση της υπάρξης της τιμής 3

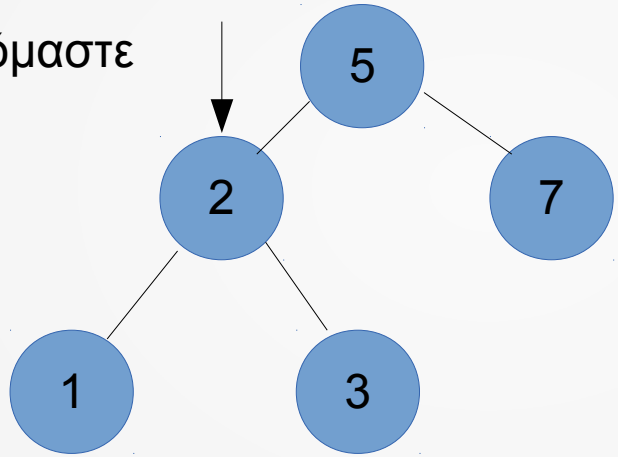



3 < 5, οπότε επισκεπτόμαστε το αριστερό υποδένδρο

# Αναζήτηση - Παράδειγμα 1

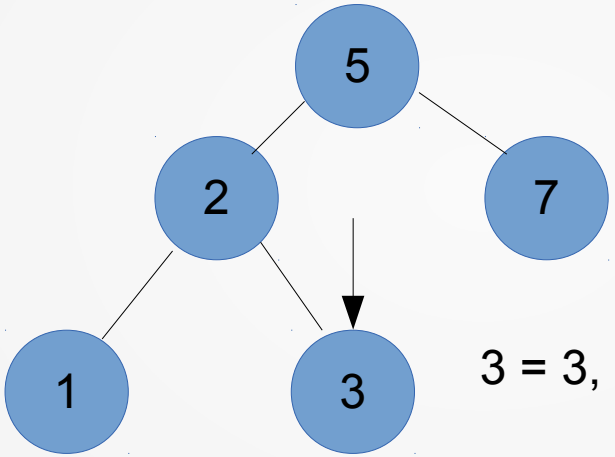
Εύρευση της υπάρξης της τιμής 3

3 > 2, οπότε επισκεπτόμαστε το δεξί υποδένδρο



# Αναζήτηση - Παράδειγμα 1

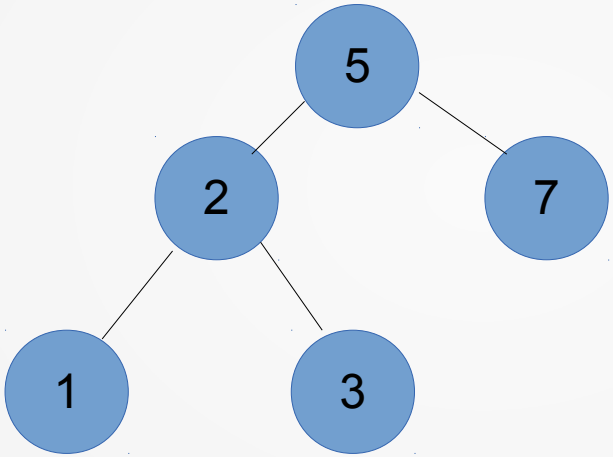
Εύρευση της υπάρξης της τιμής 3




3 = 3, οπότε υπάρχει η τιμή 3

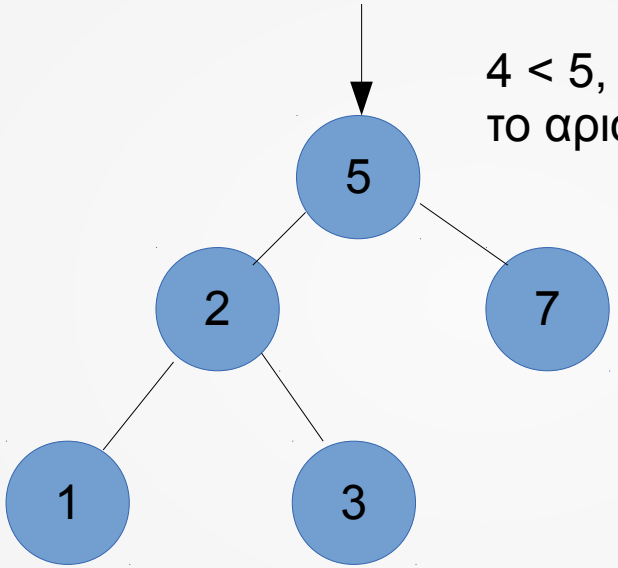
# Αναζήτηση - Παράδειγμα 2

Εύρεση της υπάρξης της τιμής 4



# Αναζήτηση - Παράδειγμα 2

Εύρευση της υπάρξης της τιμής 4

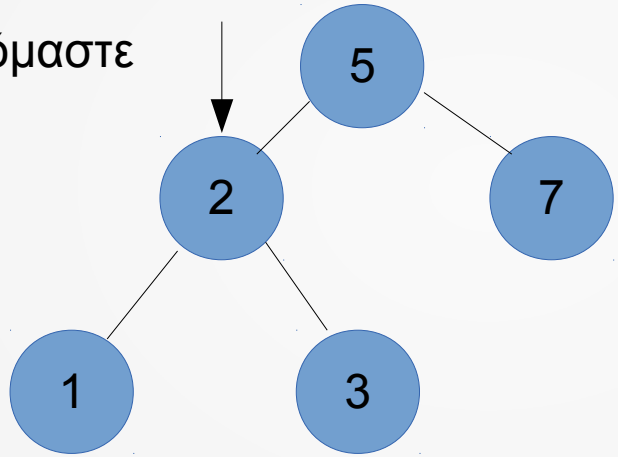



4 < 5, οπότε επισκεπτόμαστε το αριστερό υποδένδρο

# Αναζήτηση - Παράδειγμα 2

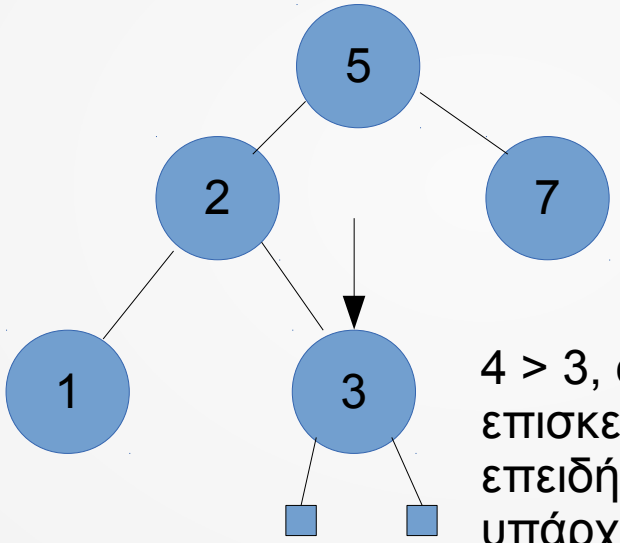
Εύρευση της υπάρξης της τιμής 4

4 > 2, οπότε επισκεπτόμαστε το δεξί υποδένδρο



# Αναζήτηση - Παράδειγμα 2

Εύρευση της υπάρξης της τιμής 3




4 > 3, οπότε θα έπρεπε να επισκεφθούμε το δεξί υποδένδρο, επειδή όμως είναι κενό, η τιμή 4 δεν υπάρχει.

- αναπαριστά τον κενό κόμβο

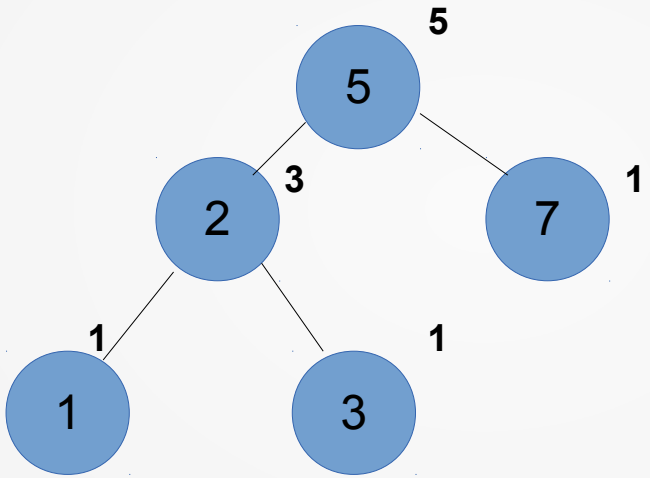
# Εισαγωγή

Η εισαγωγή ενός κόμβου γίνεται ως εξής

- Αρχικά πρέπει να βρούμε την θέση που πρέπει να μπει το νέο στοιχείο
- Εάν υπάρχει ήδη, τότε δεν κάνουμε κάτι
- Εάν όμως δεν υπάρχει, τότε καταλήγουμε σε έναν null σύνδεσμο όπου και κάνουμε την εισαγωγή.

# Εισαγωγή – Παράδειγμα 1

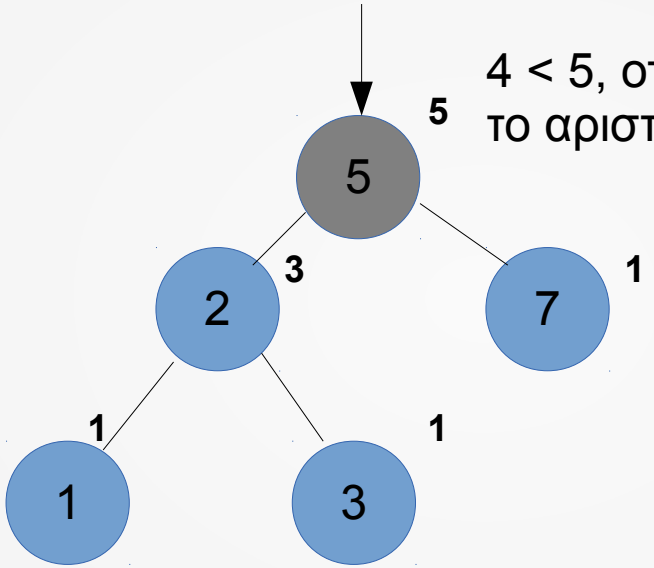
Εισαγωγή της τιμής 4




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

Εισαγωγή της τιμής 4



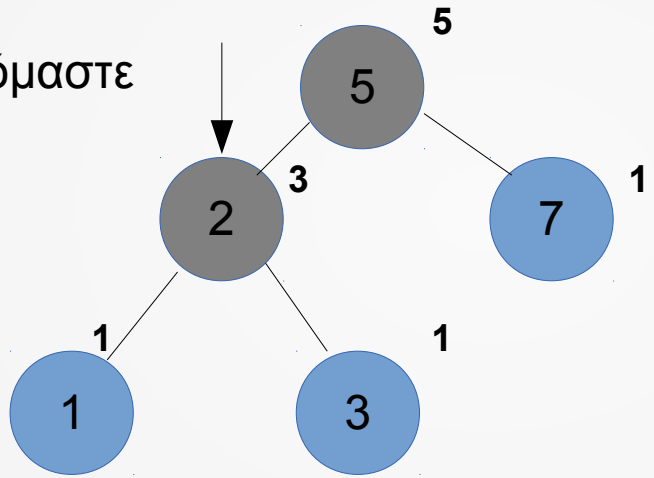

4 < 5, οπότε επισκεπτόμαστε το αριστερό υποδένδρο

Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

Εισαγωγή της τιμής 4

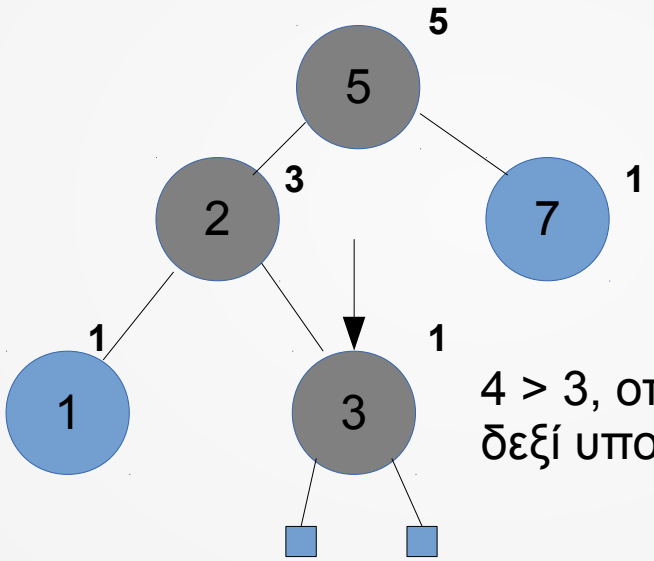
4 > 2, οπότε επισκεπτόμαστε το δεξί υποδένδρο




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

Εισαγωγή της τιμής 4.



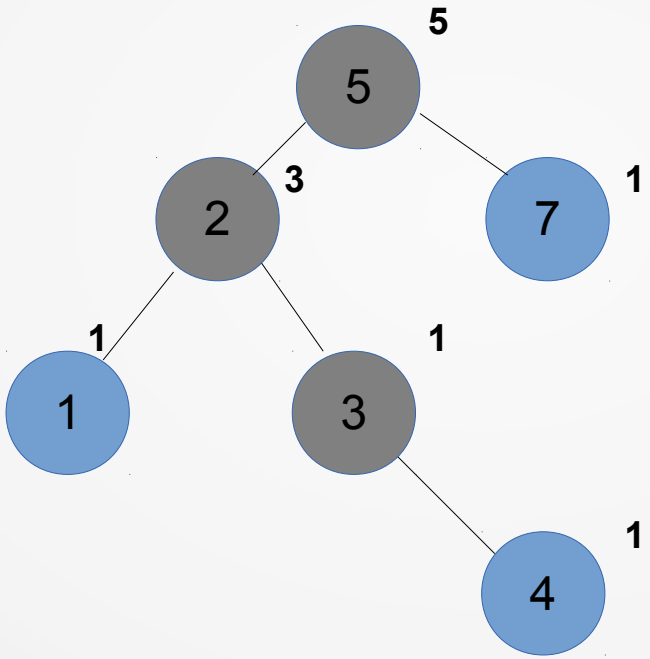

4 > 3, οπότε εισάγουμε την τιμή στο δεξί υποδένδρο

Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

- αναπαριστά τον κενό κόμβο

# Εισαγωγή – Παράδειγμα 1

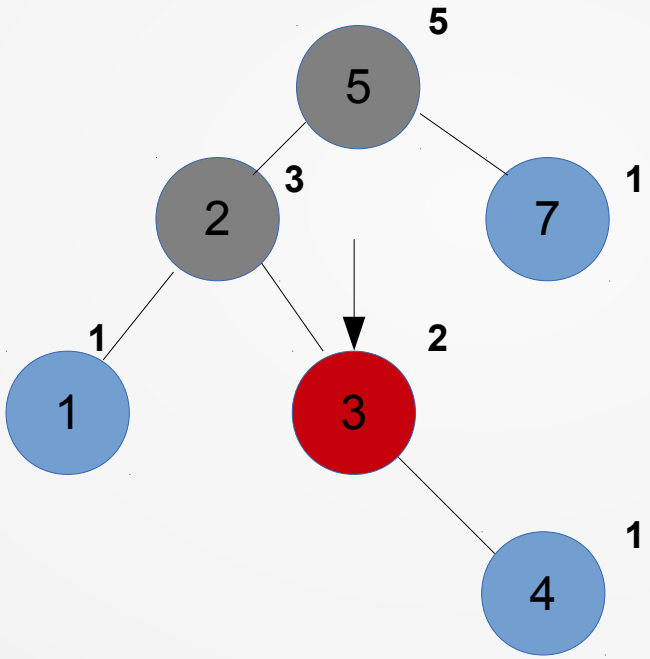
Εισαγωγή της τιμής 4.
Αφού εισάγαμε την τιμή 4, τώρα απομένει να ενημερώσουμε το μονοπάτι.




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

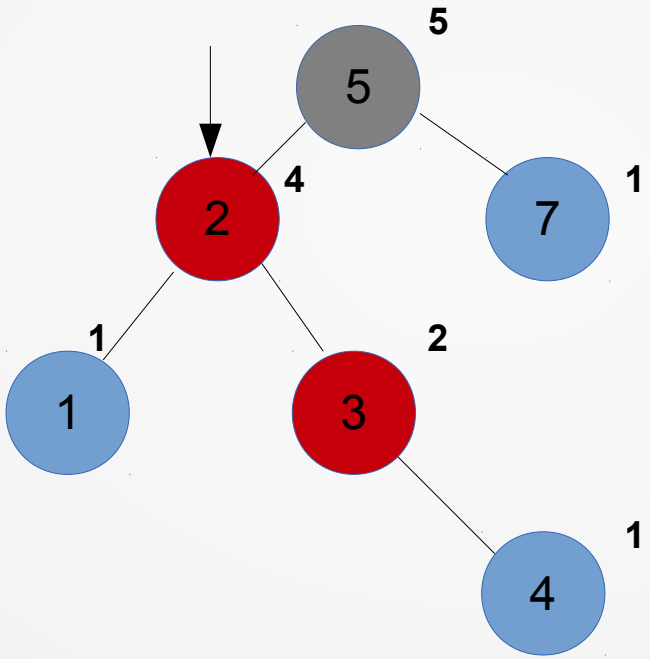
Εισαγωγή της τιμής 4.
Αφού εισάγαμε την τιμή 4, τώρα απομένει να ενημερώσουμε το μονοπάτι.




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

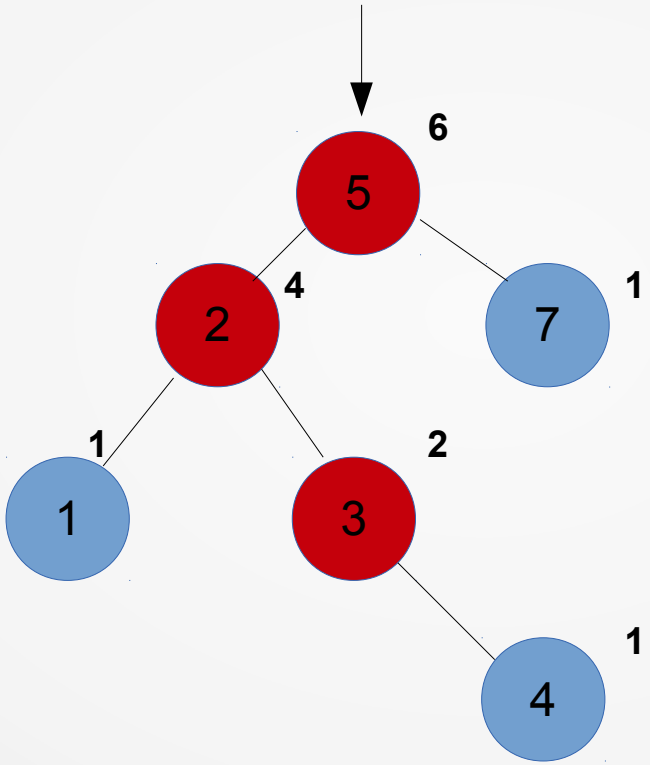
Εισαγωγή της τιμής 4.
Αφού εισάγαμε την τιμή 4, τώρα απομένει να ενημερώσουμε το μονοπάτι.




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

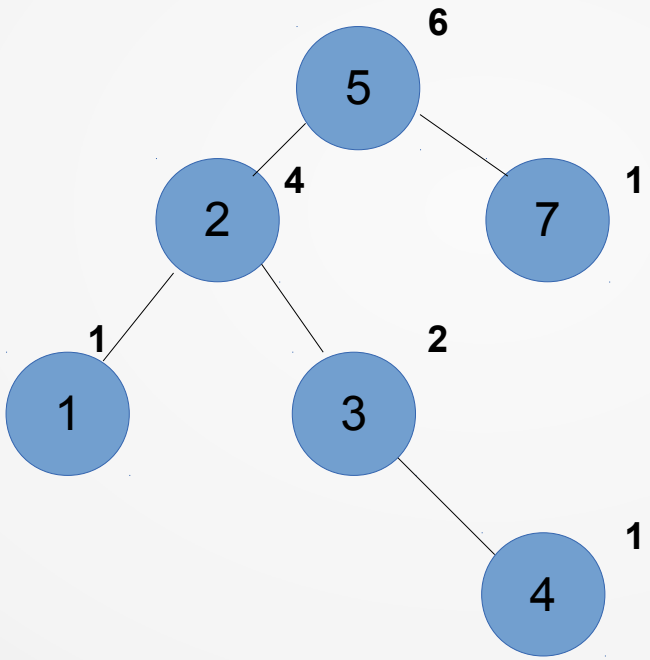
Εισαγωγή της τιμής 4.
Αφού εισάγαμε την τιμή 4, τώρα απομένει να ενημερώσουμε το μονοπάτι.




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Εισαγωγή – Παράδειγμα 1

Δένδρο μετά την εισαγωγή της τιμής 4.




Στα bold γράμματα κάθε κόμβου φαίνεται το πλήθος των υποκόμβων του + 1.

# Κώδικας σε C

```
int TreeSize( tree T ) {
    return  T == NULL ? 0 : T->cnt;
}

void UpdateCnt( tree T ) {
    if ( T ) {
        T->cnt = 1 + TreeSize(T->left) + TreeSize(T->right);
    }
}
```

# Κώδικας σε C

```
tree insert (tree t, int x) {
    if ( t == NULL ) {
        tree n = new tree_element;
        n->data = x;
        n->cnt = 1;
        n->left = n->right = NULL;
        return n;
    }
    if ( x < t->data ) {
        t->left = insert(t->left, x);
    }else if ( x > t->data ) {
        t->right = insert(t->right, x);
    }
    UpdateCnt( t );
    return t;
}
```

# Διαγραφή

Έστω t ο κόμβος που θέλουμε να διαγράψουμε, πρέπει να λάβουμε υπόψιν τις εξής τρεις περιπτώσεις:

- Εάν ο t δεν έχει καθόλου παιδιά, απλά τον διαγράφουμε.
- Εάν ο t έχει μονάχα ένα παιδί, τότε διαγράφουμε τον t και συνδέουμε το μοναδικό παιδί με τον πατέρα του t.
- Εάν ο t έχει δύο παιδιά τότε:

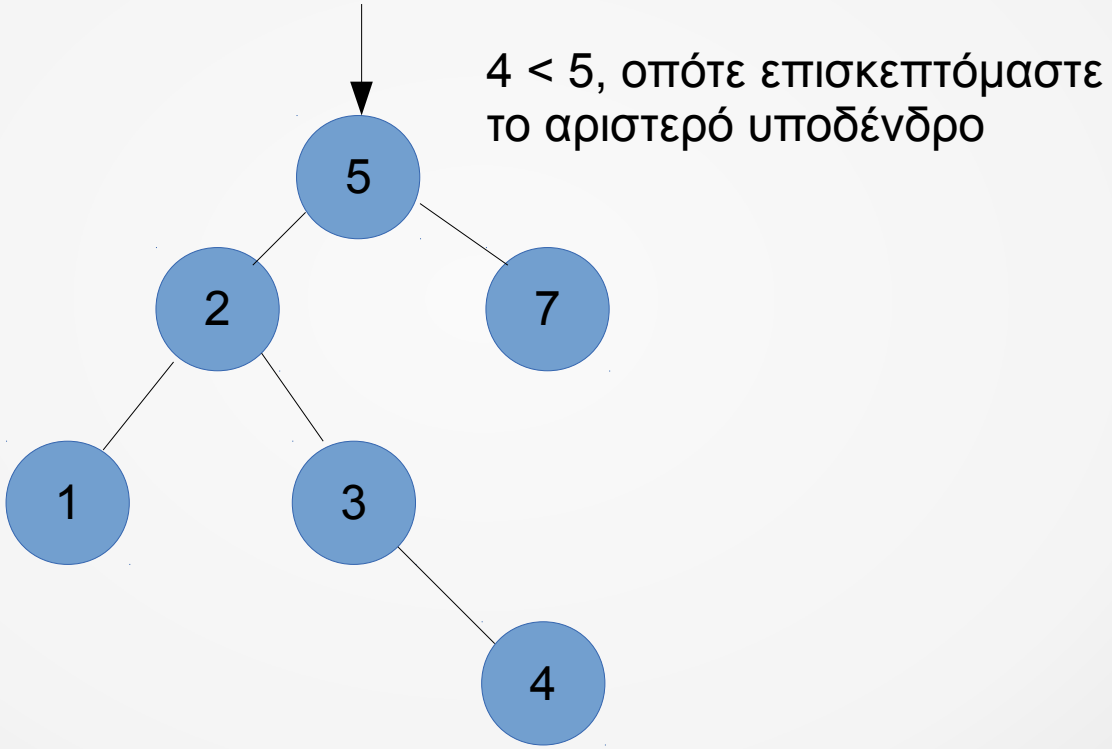
  i)Βρίσκουμε τον μικρότερο κόμβο στο δεξί υποδένδρο του t έστω k

  ii)Διαγράφουμε τον k

  iii) Αντικαθιστούμε την τιμή του t με την τιμή του k

# Διαγραφή – Περίπτωση 1

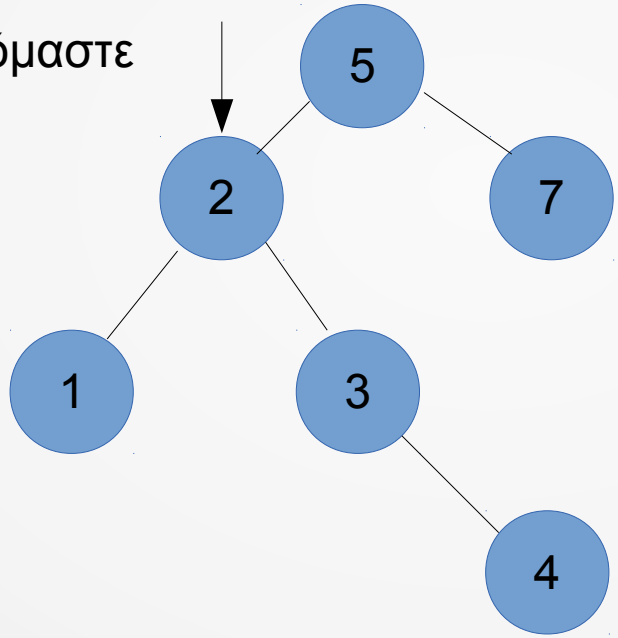
Διαγραφή της τιμής 4.



# Διαγραφή – Περίπτωση 1

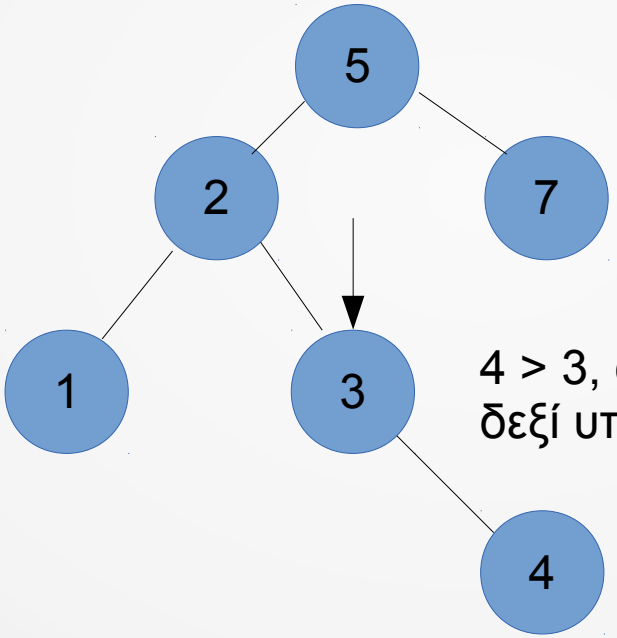
Διαγραφή της τιμής 4.

4 > 2, οπότε επισκεπτόμαστε το δεξί υποδένδρο



# Διαγραφή – Περίπτωση 1

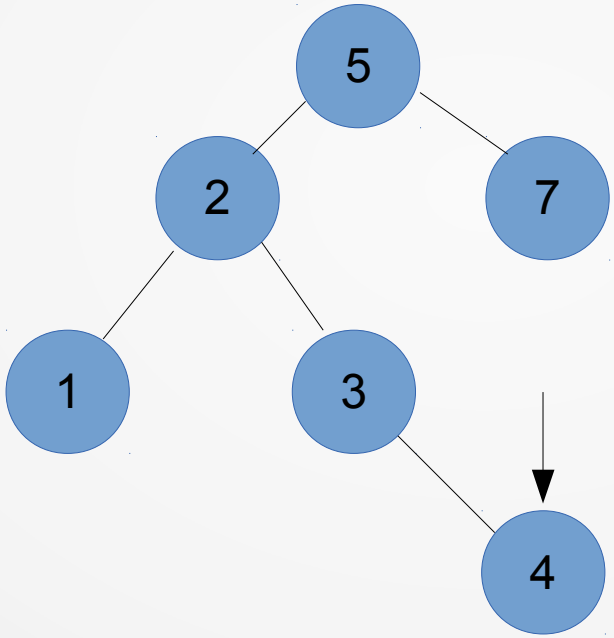
Διαγραφή της τιμής 4.




4 > 3, οπότε επισκεπτόμαστε το δεξί υποδένδρο

# Διαγραφή – Περίπτωση 1

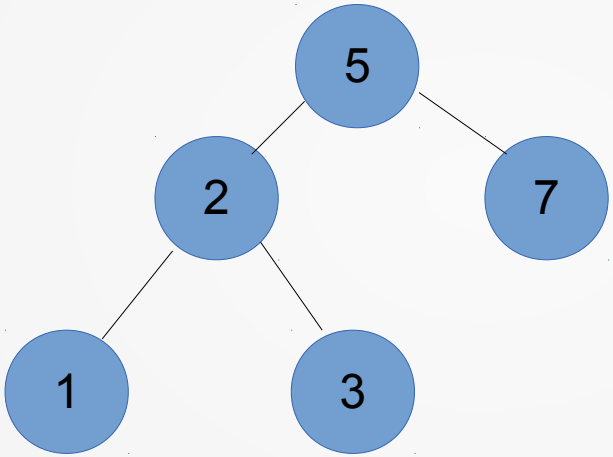
Διαγραφή της τιμής 4.




Δεν περιέχει παιδιά. Οπότε απλά τον διαγράφουμε.

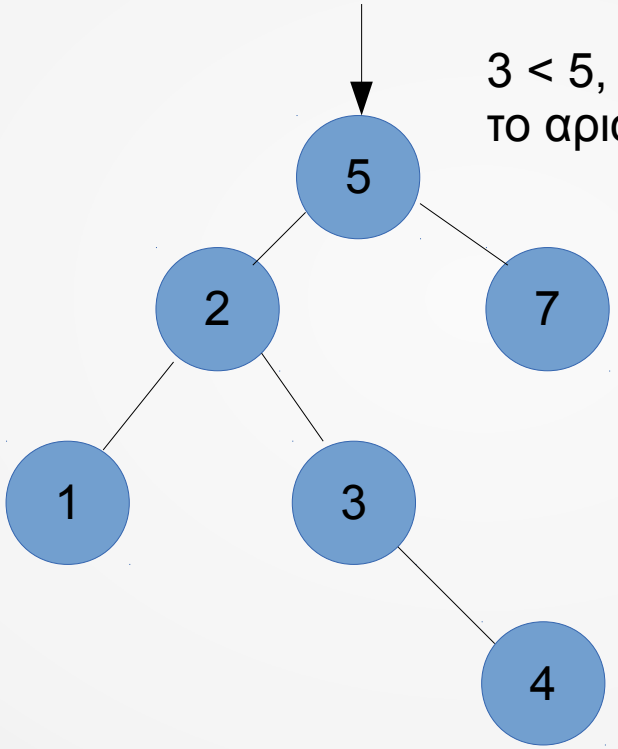
# Διαγραφή – Περίπτωση 1

Δένδρο μετά την διαγραφή της τιμής 4.



# Διαγραφή – Περίπτωση 2

Διαγραφή της τιμής 3.

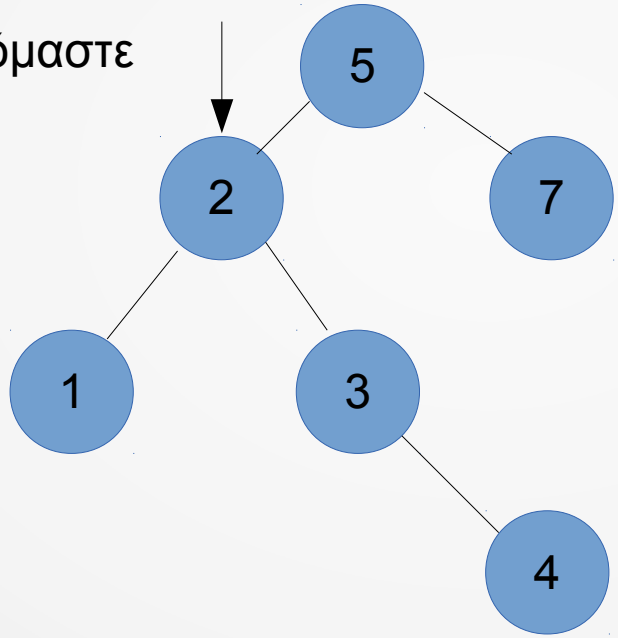



3 < 5, οπότε επισκεπτόμαστε το αριστερό υποδένδρο

# Διαγραφή – Περίπτωση 2

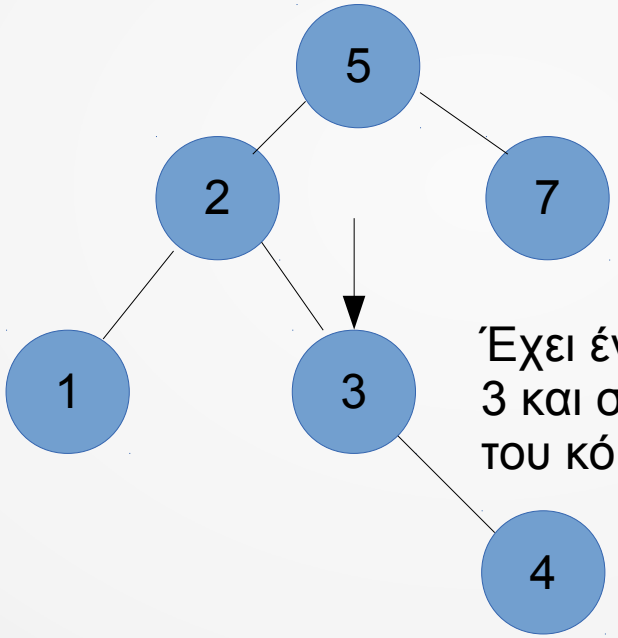
Διαγραφή της τιμής 3.

3 > 2, οπότε επισκεπτόμαστε το δεξί υποδένδρο



# Διαγραφή – Περίπτωση 2

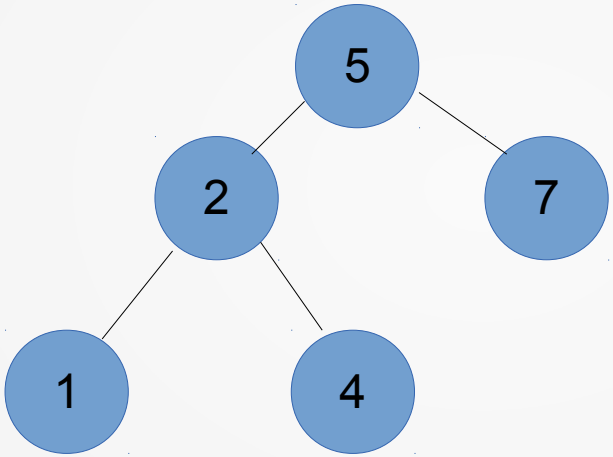
Διαγραφή της τιμής 3.




Έχει ένα παιδί, οπότε διαγράφουμε τον κόμβο 3 και συνδέουμε τον κόμβο 4 πλέον ως παιδί του κόμβου 2.

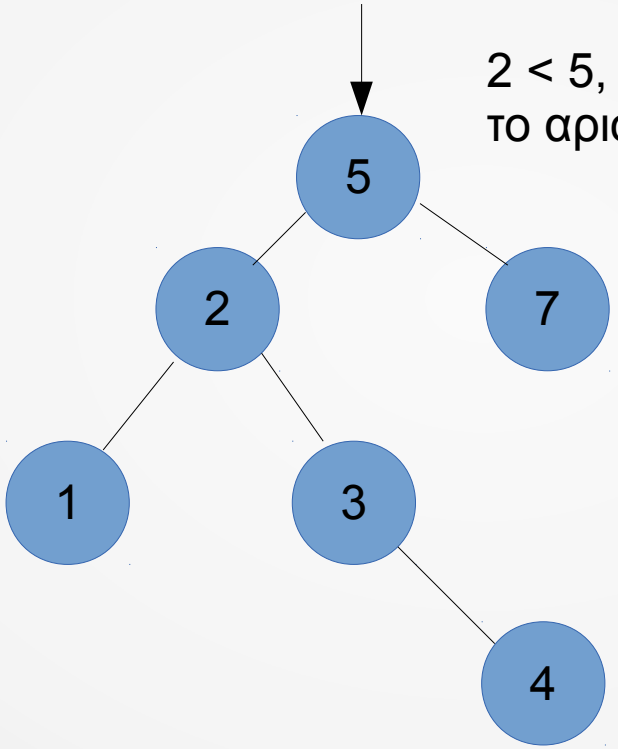
# Διαγραφή – Περίπτωση 2

Δένδρο μετά την διαγραφή της τιμής 3.



# Διαγραφή – Περίπτωση 3

Διαγραφή της τιμής 2.

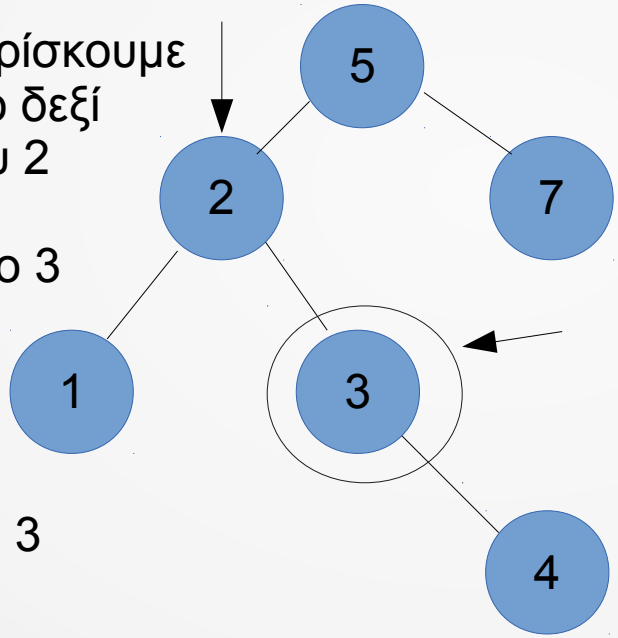



2 < 5, οπότε επισκεπτόμαστε το αριστερό υποδένδρο

# Διαγραφή – Περίπτωση 3

Διαγραφή της τιμής 2.

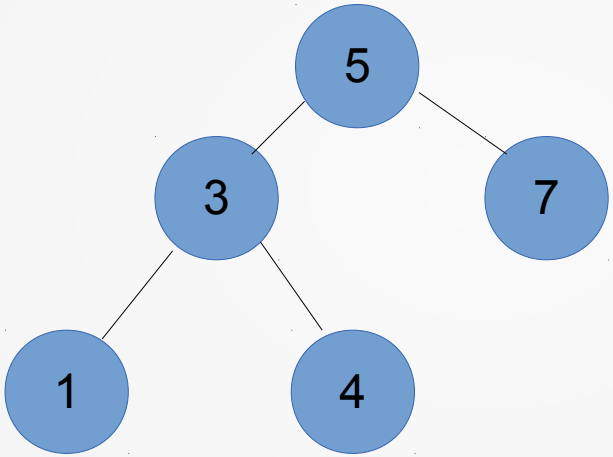
Έχει 2 παιδιά, οπότε βρίσκουμε την μικρότερη τιμή, στο δεξί υποδένδρο του κόμβου 2 που είναι το 3, έπειτα διαγράφουμε τον κόμβο 3 και εφαρμόζεται η περίπτωση 2. Τέλος αντικαθιστούμε την τιμή του κόμβου 2 με την τιμή του κόμβου 3




Ελάχιστο στοιχείο του δεξιού υποδένδρου του κόμβου 2.

# Διαγραφή – Περίπτωση 3

Δένδρο μετά την διαγραφή της τιμής 2.



# Αναζήτηση k-οστού

Για να απαντήσουμε στο ερώτημα του k-οστου θα μπορούσαμε να κάνουμε μία διάσχιση inOrder σε O(N). Εκμεταλευόμαστε όμως τις ιδιότητες του δυαδικού δένδρου που σε κάθε κόμβο αφήνει τις μικρότερες τιμές στο αριστερό παιδί και τις μεγαλύτερες τιμές στο δεξιό ακολουθώντας τον ακόλουθο αλγόριθμο.

# Αναζήτηση k-οστού

1. Ξεκινώντας από την ρίζα(T) ελέγχουμε τον αριθμό των κόμβων στο αριστερό παιδί.

2. Εάν το k = NUM( left_subtree(T) ) + 1, τότε βρήκαμε αυτό που ψάχνουμε.

3. Εάν k > NUM( left_subtree(T) ) + 1, τότε μπορούμε να αγνοήσουμε το αριστερό υποδένδρο επειδή τα στοιχεία θα είναι μικρότερα από το k-οστό στοιχείο. Έτσι, επιλύουμε το πρόβλημα αναδρομικά στο δεξιό υποδένδρο για k' = k - NUM( left_subtree(T) ) στοιχεία.

4. Εάν k < NUM( left_subtree(T) ) + 1, τότε η απάντηση βρίσκεται κάπου στο αριστερό υποδένδρο και λύνουμε το πρόβλημα αναδρομικά στο αριστερό υποδένδρο.

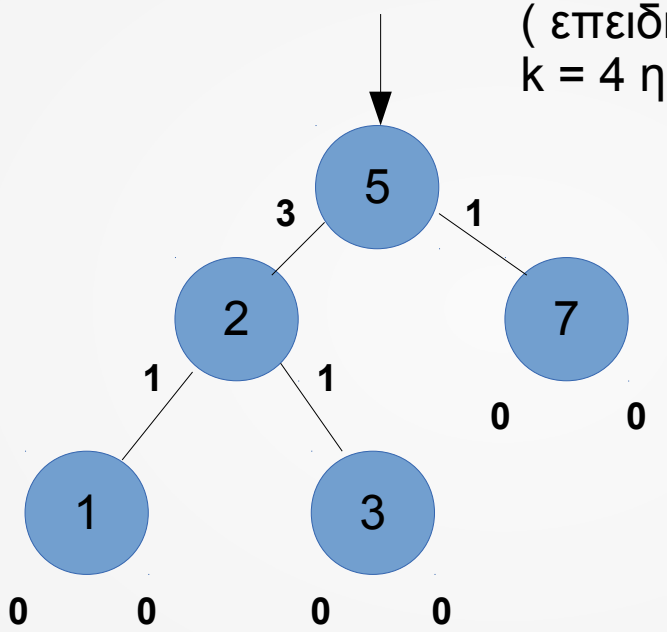
# Αναζήτηση k-οστού

Ας το δούμε αναλυτικότερα.

# Αναζήτηση k-οστού – Παράδειγμα 1

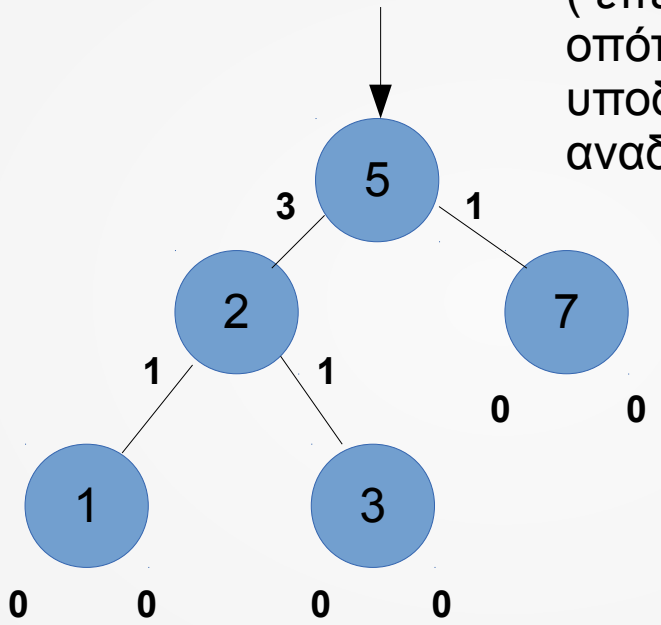
Αναζήτηση 4ου στοιχείου

Το αριστερό υποδένδρο έχει 3 στοιχεία + 1 ( επειδή μετράται και την ρίζα ), επειδή k = 4 η τιμή που ψάχνουμε είναι το 5.



# Αναζήτηση k-οστού – Παράδειγμα 2

Αναζήτηση 5ου στοιχείου

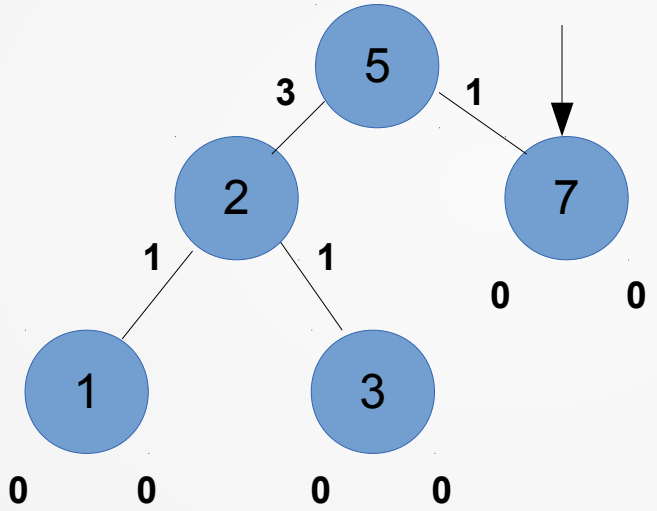



Το αριστερό υποδένδρο έχει 3 στοιχεία + 1 ( επειδή μετράμε την ρίζα ), όμως 4 < 5=k, οπότε η απάντηση βρίσκεται στο δεξί υποδένδρο και λύνουμε το πρόβλημα αναδρομικά για k' = k-4 = 5-4 =1

# Αναζήτηση k-οστού – Παράδειγμα 2

Αναζήτηση 5ου στοιχείου

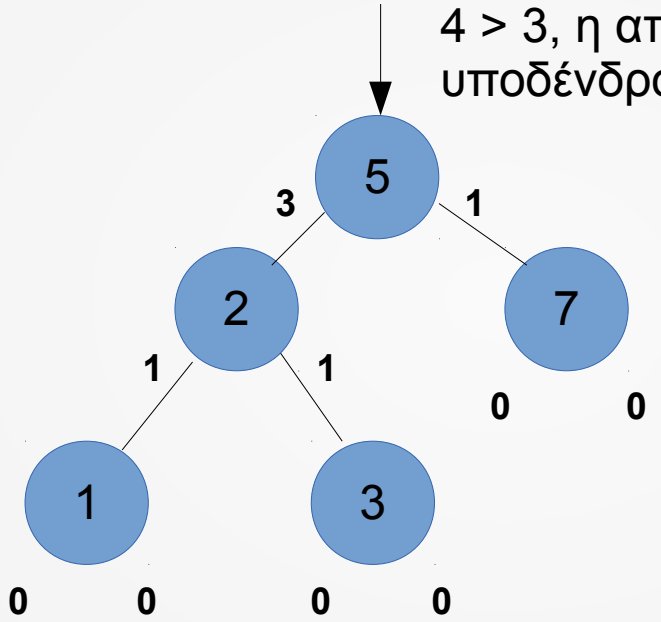
Το αριστερό υποδένδρο έχει 0 στοιχεία + 1 (επειδή μετράται ο ίδιος ο κόμβος) και επειδή 1 = κ' η απάντηση είναι ο ίδιος κόμβος που είναι το 7.



# Αναζήτηση k-οστού – Παράδειγμα 3

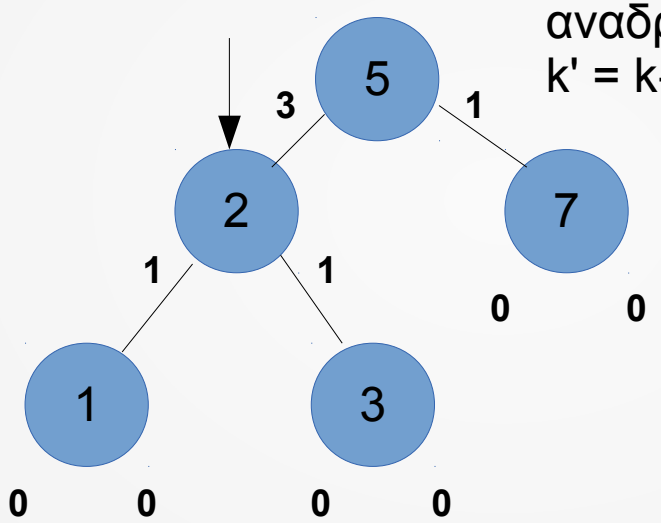
Αναζήτηση 3ου στοιχείου

Το αριστερό υποδένδρο έχει 3 στοιχεία + 1 ( επειδή μετράται και η ρίζα ), επειδή 4 > 3, η απάντηση βρίσκεται κάπου στο αριστερό υποδένδρο



# Αναζήτηση k-οστού – Παράδειγμα 3

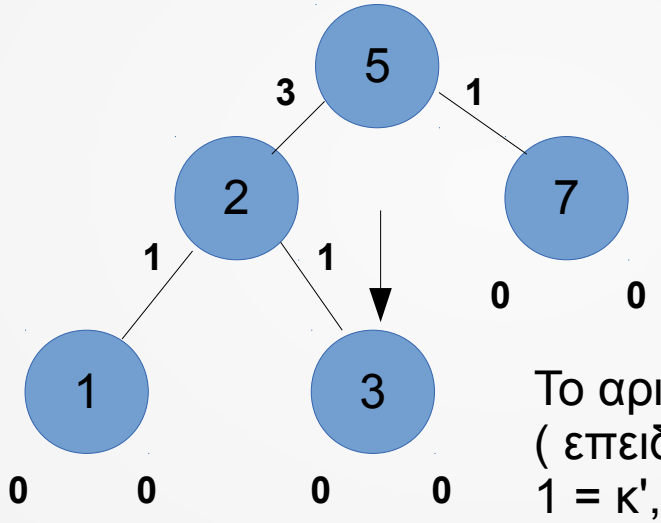
Αναζήτηση 3ου στοιχείου




Το αριστερό υποδένδρο έχει 1 στοιχείο + 1 ( επειδή μετράται ο ίδιος ο κόμβος ), επειδή 2 < 3=k, η απάντηση βρίσκεται στο δεξί υποδένδρο και λύνουμε το πρόβλημα αναδρομικά στο δεξιό υποδένδρο για k' = k-2 = 3-2 = 1

# Αναζήτηση k-οστού – Παράδειγμα 3

Αναζήτηση 3ου στοιχείου




Το αριστερό υποδένδρο έχει 0 στοιχεία + 1 ( επειδή μετράται ο ίδιος ο κόμβος ), επειδή 1 = κ', η απάντηση είναι ο ίδιος ο κόμβος, δηλαδή το 3

# Κώδικας σε C

```
tree FindKth( tree T, int k ) {

    if ( TreeSize(T) < k )
        return ( NULL );

    int sz = 1 + TreeSize(T->left);

    if( sz == k ) {
        return ( T );
    }

    if( sz < k ) {
        return FindKth( T->right, k-sz );
    }

    return ( FindKth(T->left, k) );

}
```