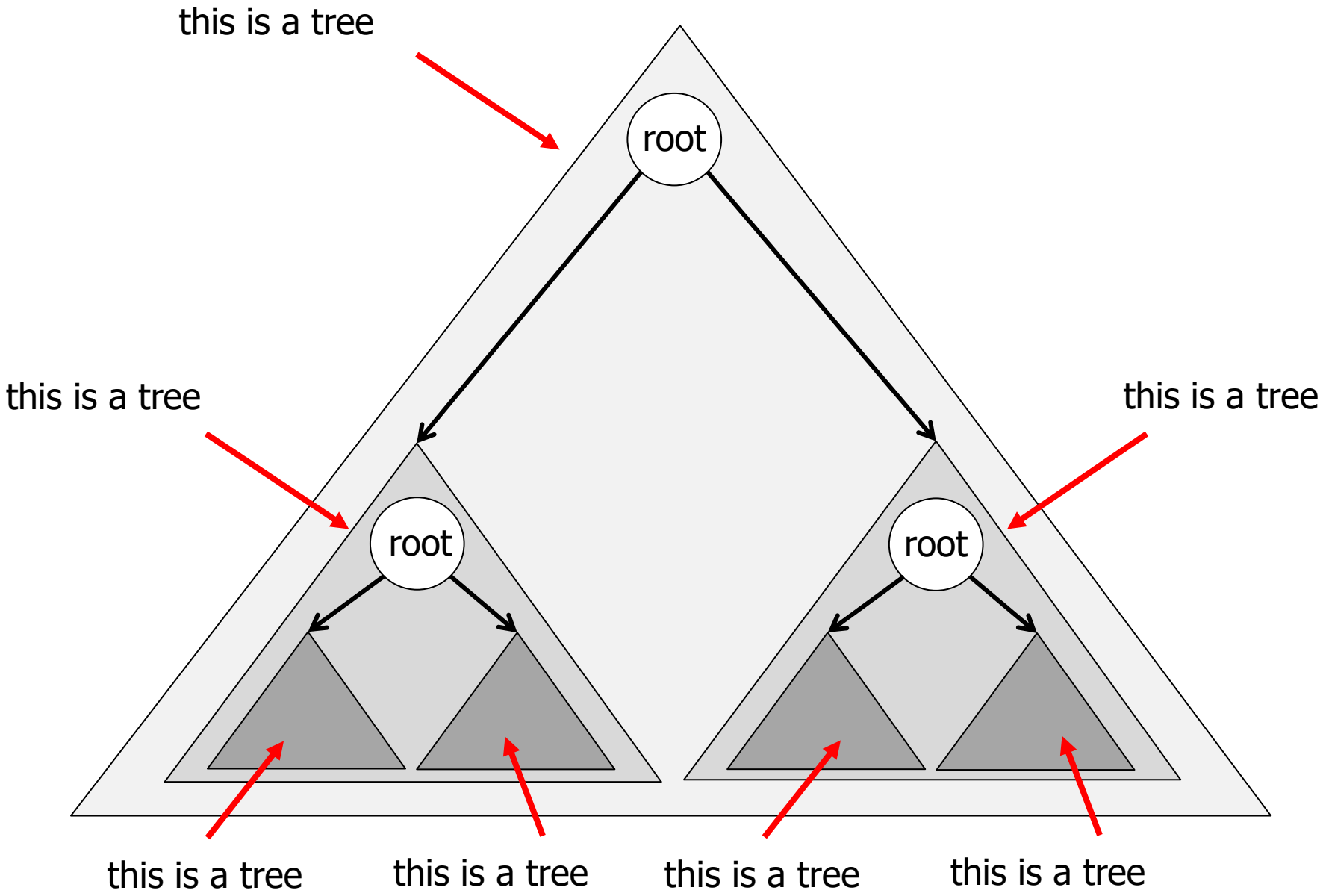
Προγραμματισμός ΙΙ (ECE116)

#8 διασυνδεδεμένες δομές: δυαδικά δέντρα

Σπύρος Λάλης
lalis@uth.gr

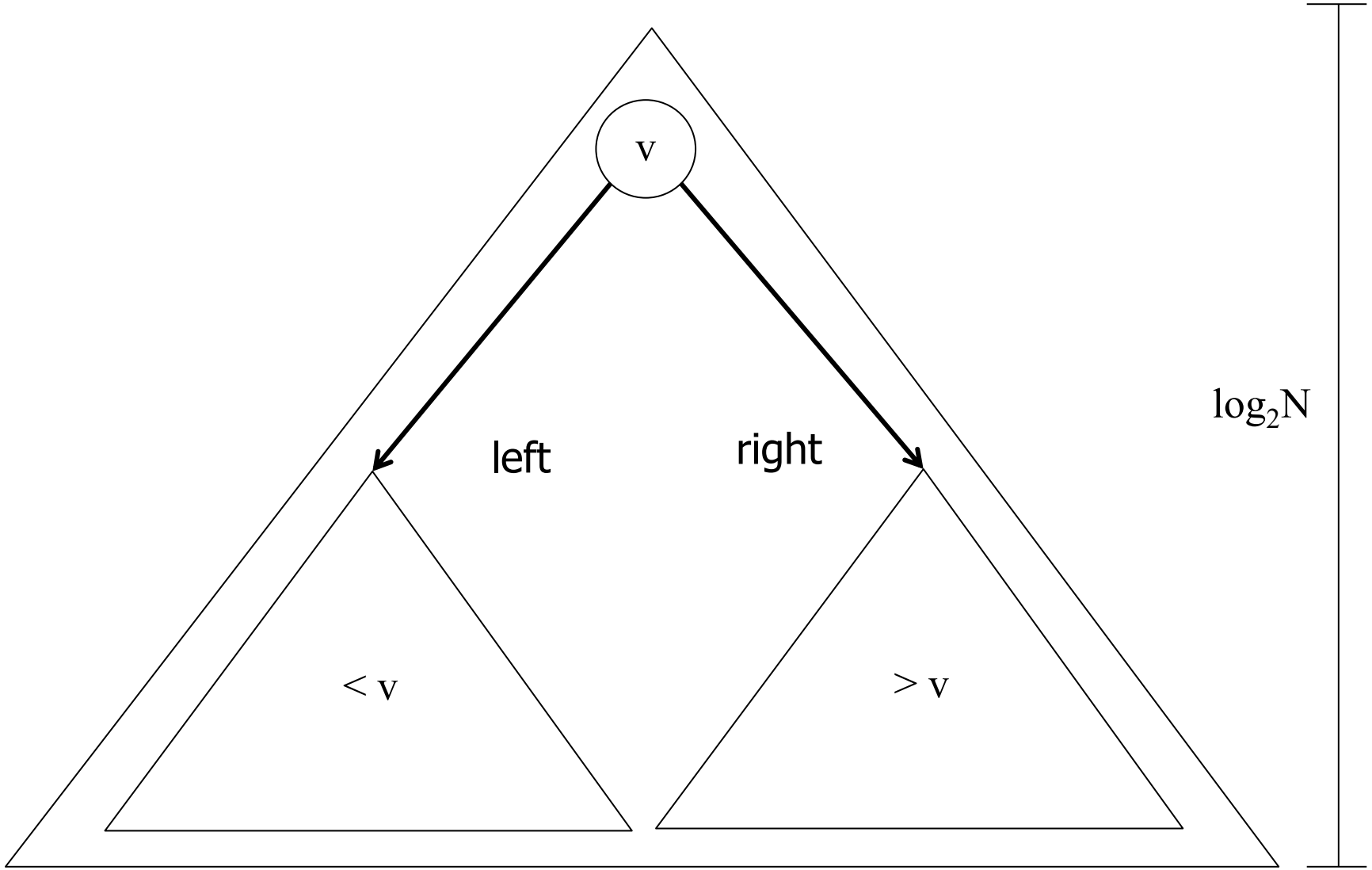
Δέντρα

- Το δέντρο είναι μια κλασική **αναδρομική** δομή
- Ένα δέντρο αποτελείται από υποδέντρα ...
- ... καθένα από τα οποία μπορεί να θεωρηθεί ως ένα (άλλο) δέντρο
- ... που με την σειρά του αποτελείται από υποδέντρα


this is a tree
root
this is a tree
this is a tree
root
root
this is a tree
this is a tree
this is a tree
this is a tree

Δυαδικά δέντρα αναζήτησης

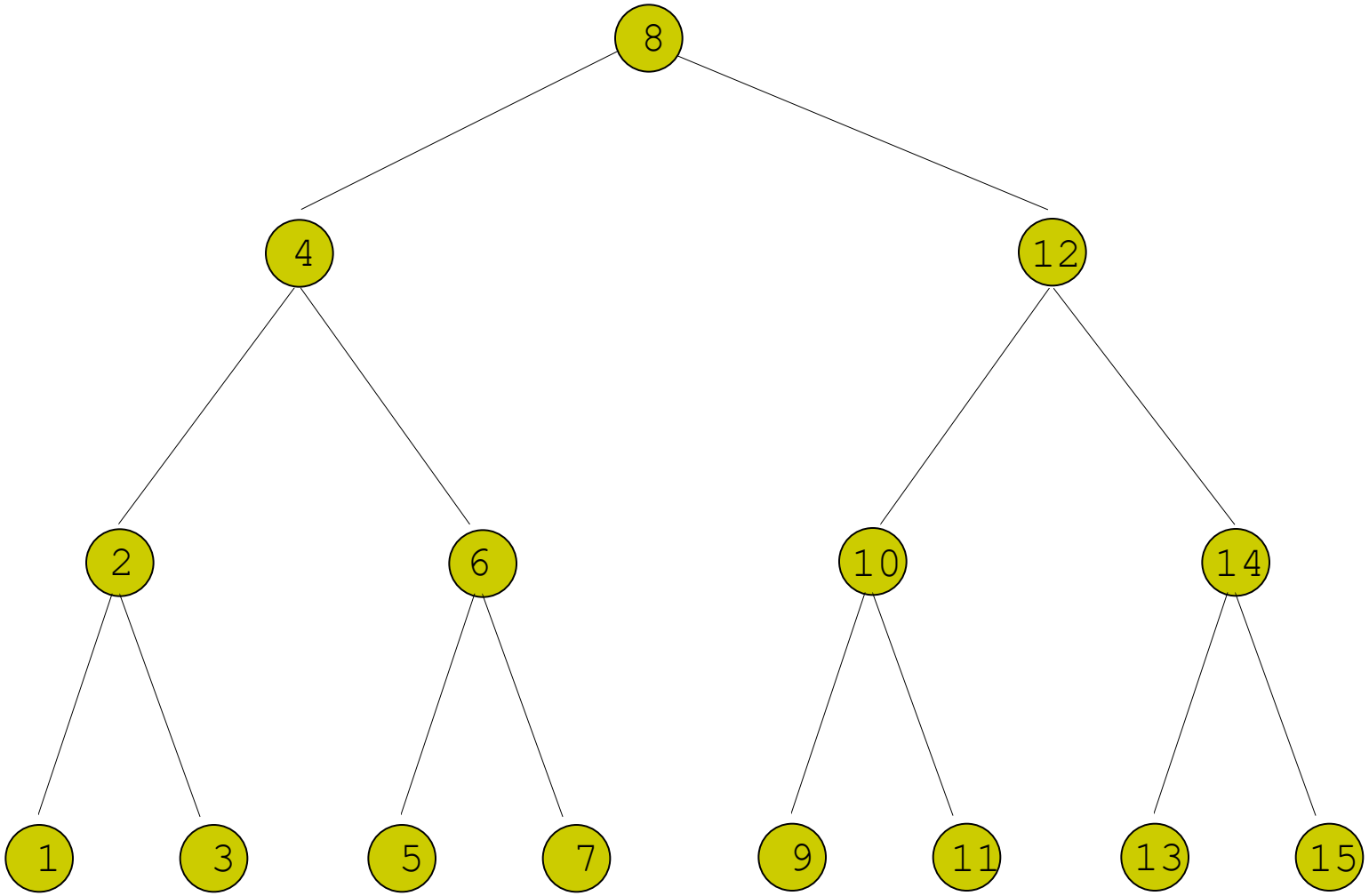
- Τα **δυαδικά** δέντρα είναι ειδική περίπτωση δέντρων, όπου κάθε κόμβος έχει (το πολύ) **δύο** υποδέντρα
- Η κατασκευή τους μπορεί να γίνει με τέτοιο τρόπο έτσι ώστε να υποστηρίζεται **γρήγορη αναζήτηση**
- **Κανόνας:** Για κάθε κόμβο με κλειδί v, όλοι οι κόμβοι του αριστερού υποδέντρου έχουν κλειδιά **μικρότερα** του v, και όλοι οι κόμβοι του δεξιού υποδέντρου έχουν κλειδιά **μεγαλύτερα** του v
  - μπορεί να ακολουθηθεί και ο ακριβός ανάποδος κανόνας
- Αν το δέντρο είναι **ισορροπημένο** (balanced), η αναζήτηση απαιτεί τάξη μεγέθους $\log_2 N$ βήματα όπου N το πλήθος των κόμβων του δέντρου

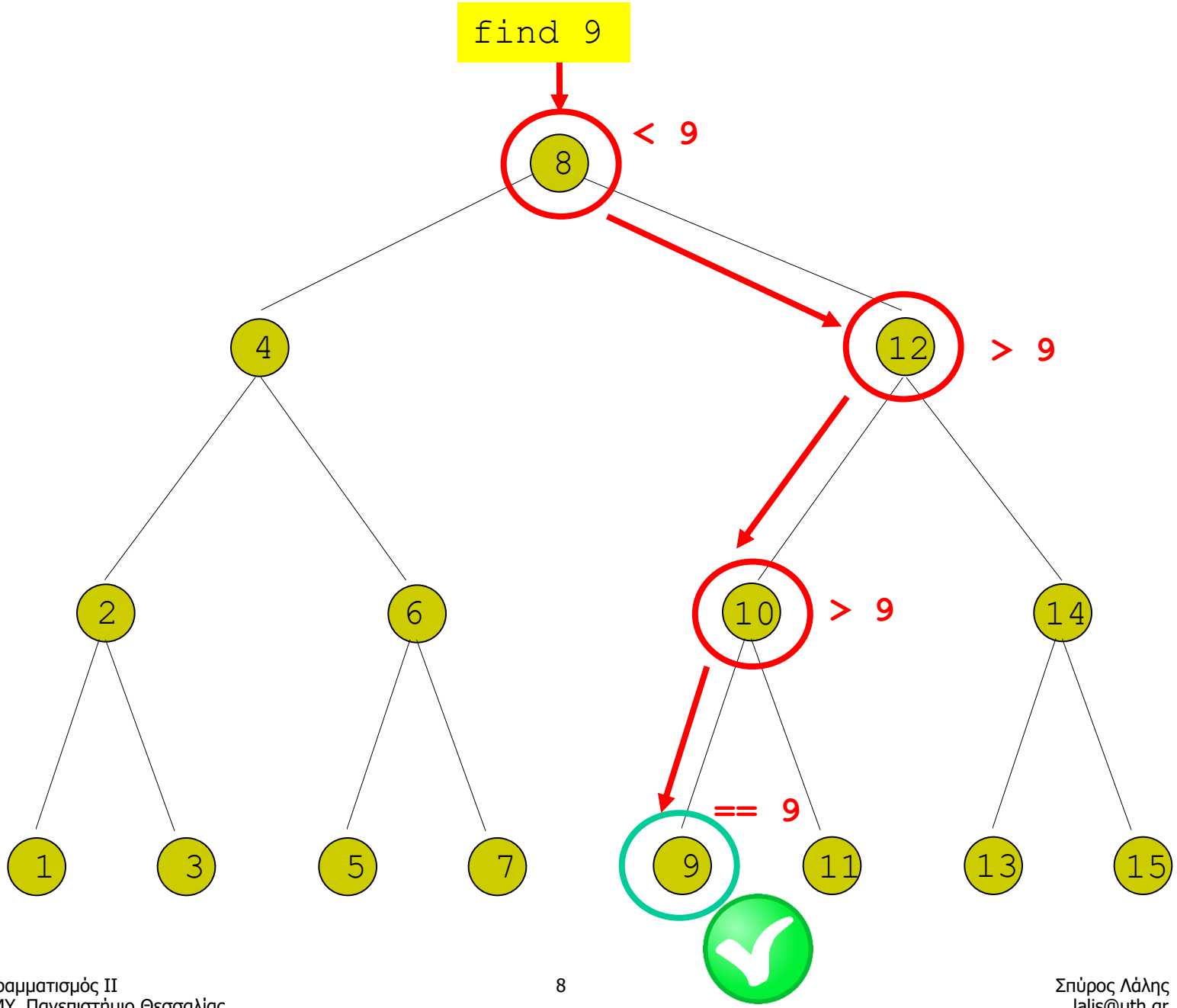

v
left
right
< v
> v
log2N

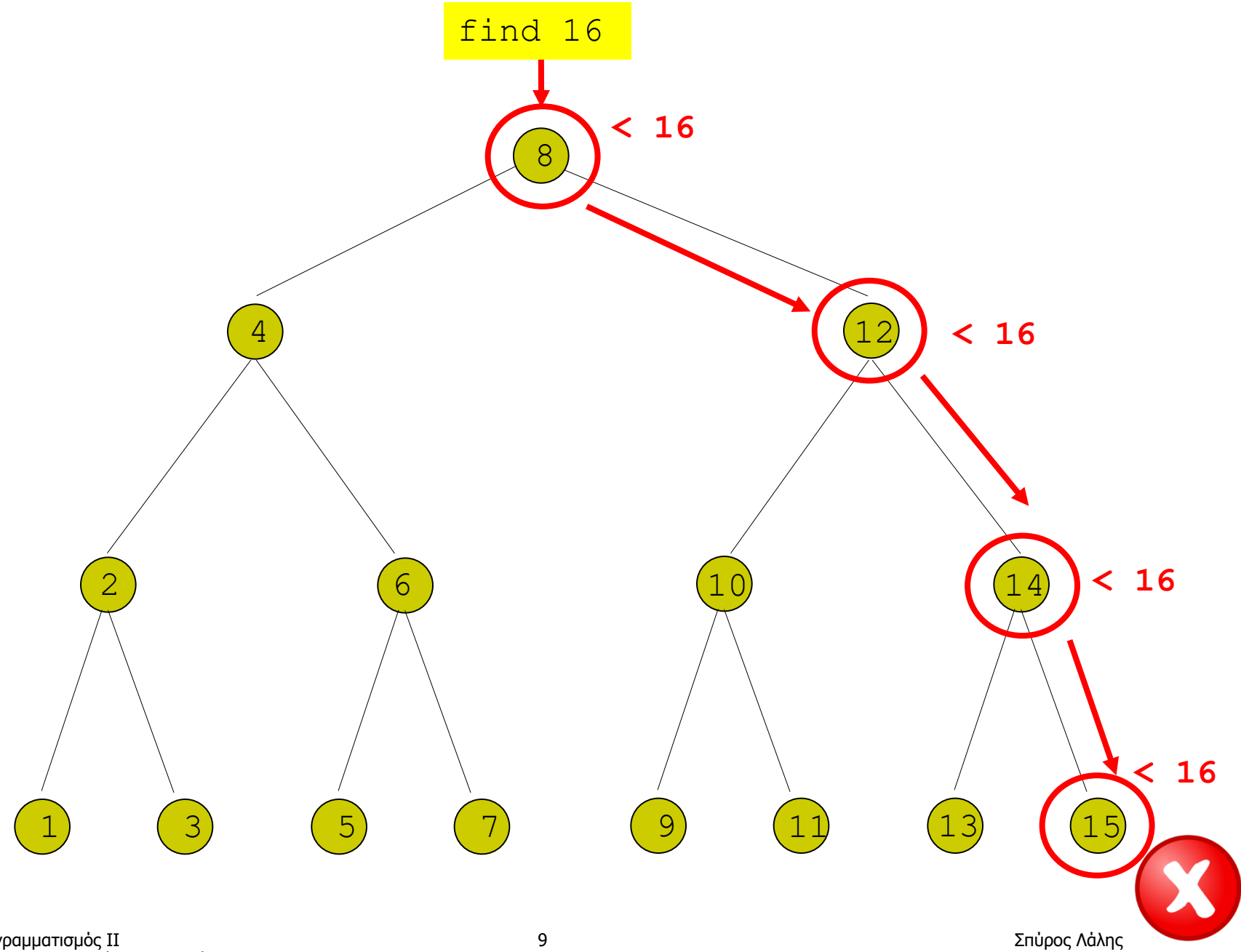
```
struct btree {
  int key; /* plays the role of a search key/id */
  …        /* more data, irrelevant for this example */
  struct btree *left,*right;
};
```

```
void btree_init(struct btree **root) {
  *root = NULL;
}
```

```
struct btree *btree_new() {
  return(NULL);
}
```


8
4
12
2
6
10
14
1
3
5
7
9
11
13
15


find 9
8
< 9
4
12
> 9
2
6
10
> 9
14
1
3
5
7
== 9
9
11
13
15

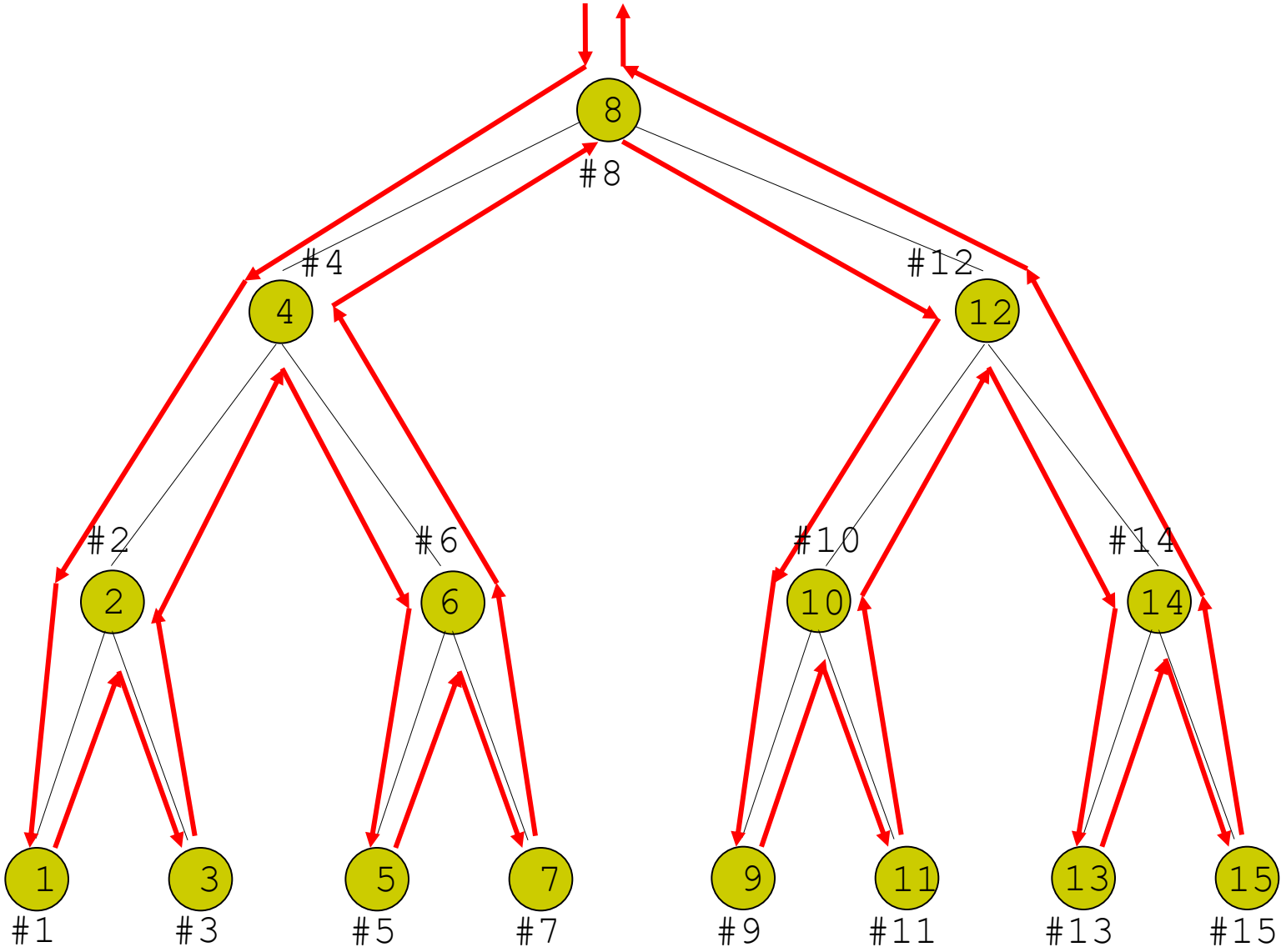

find 16
8
< 16
4
12
< 16
2
6
10
14
< 16
1
3
5
7
9
11
13
15
< 16

```
struct btree *btree_find(struct btree *root, int key) {
  if (root == NULL) {
    return(root);
  }
  else if (key == root->key) {
    return(root);
  }
  else if (key < root->key) {
    return(btree_find(root->left,key));
  }
  else /* key > root->key */ {
    return(btree_find(root->right,key));
  }
}
```

Διάσχιση δυαδικών δέντρων

- Η διάσχιση ενός δυαδικού δέντρου αναζήτησης μπορεί επίσης να γίνει εύκολα με **αναδρομικό** τρόπο
- Μπορεί να γίνει επίσκεψη των κόμβων του δέντρου με «αύξουσα» ή με «φθίνουσα» σειρά
- Αναλόγως με το πως εισάγουμε την αναδρομή για τα υποδέντρα στον κώδικα της διάσχισης

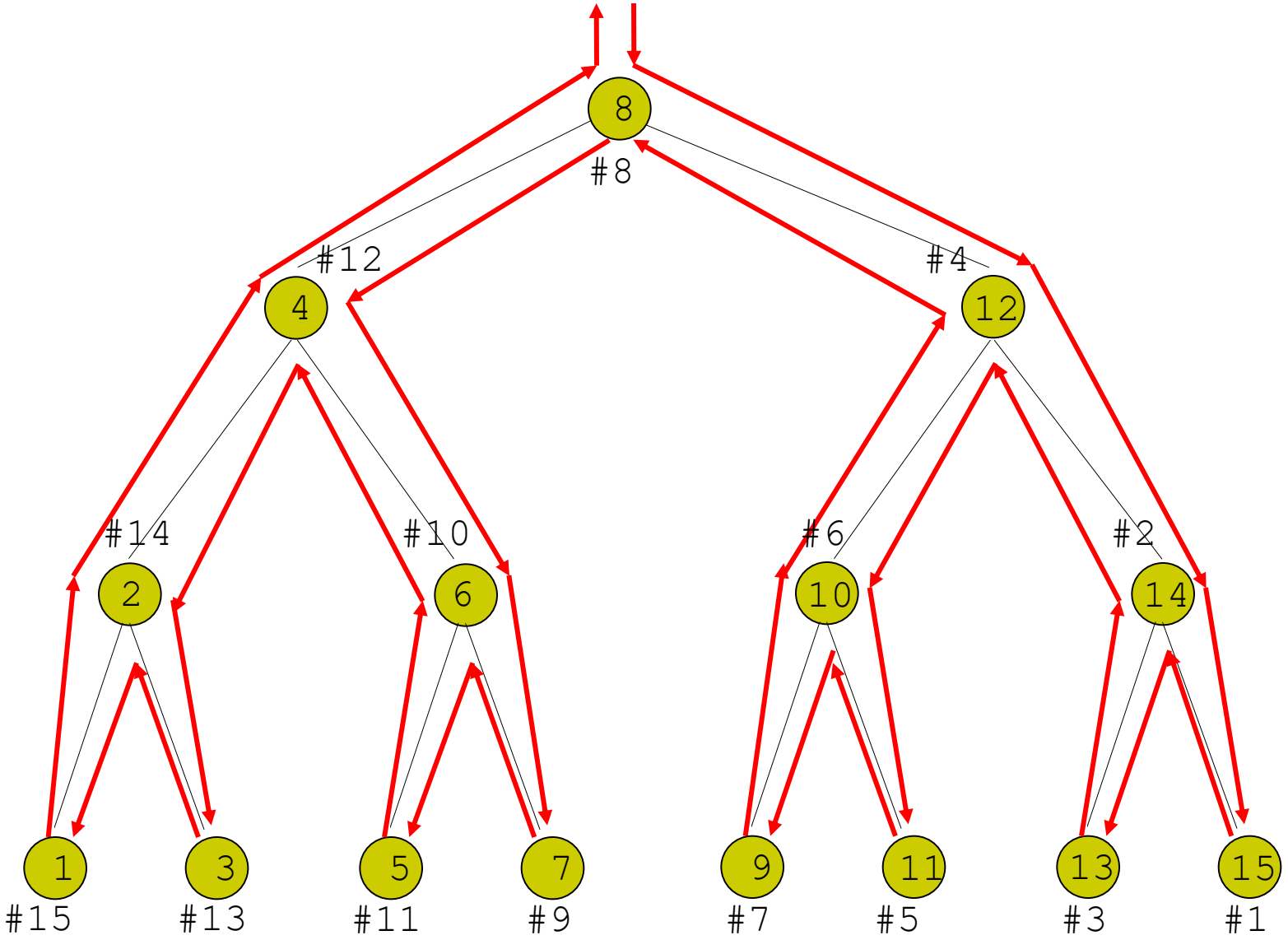
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



με αύξουσα σειρά τιμών

```
void btree_traverse_inc(struct btree *root) {
  if (root != NULL) {
    btree_traverse_inc(root->left);
    printf("%d ",root->key);
    btree_traverse_inc(root->right);
  }
}
```

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

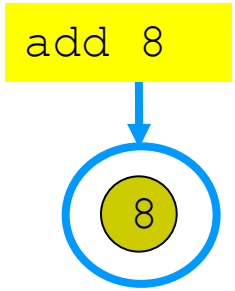


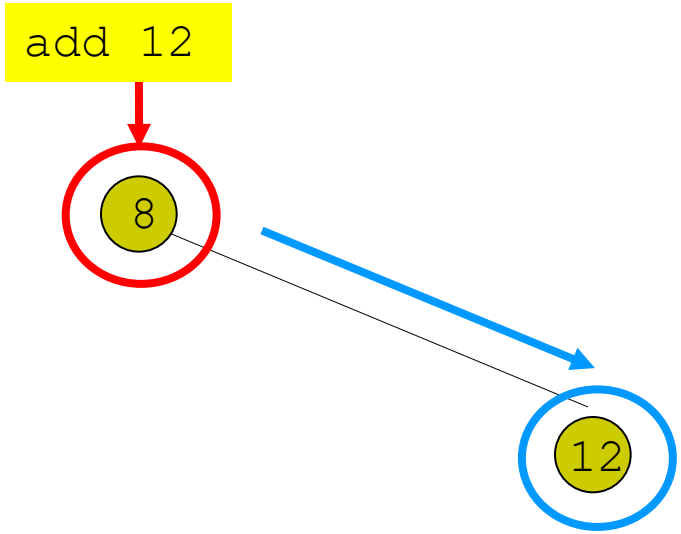
με φθίνουσα σειρά τιμών

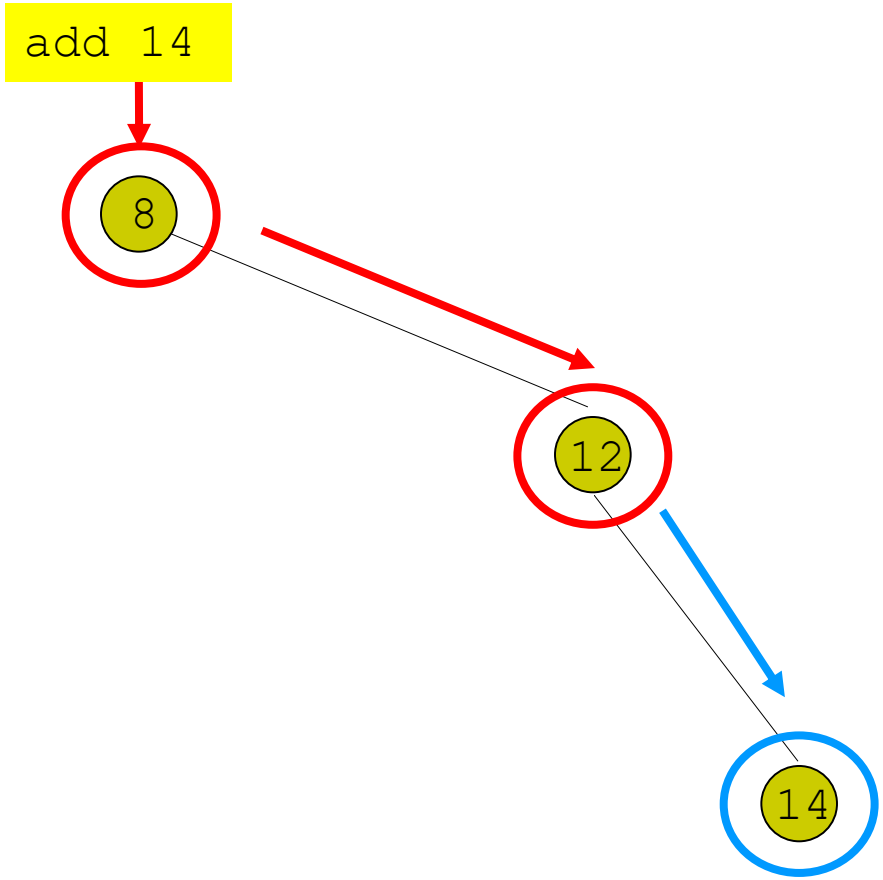
```
void btree_traverse_dec(struct btree *root) {
  if (root != NULL) {
    btree_traverse_dec(root->right);
    printf("%d ",root->key);
    btree_traverse_dec(root->left);
  }
}
```

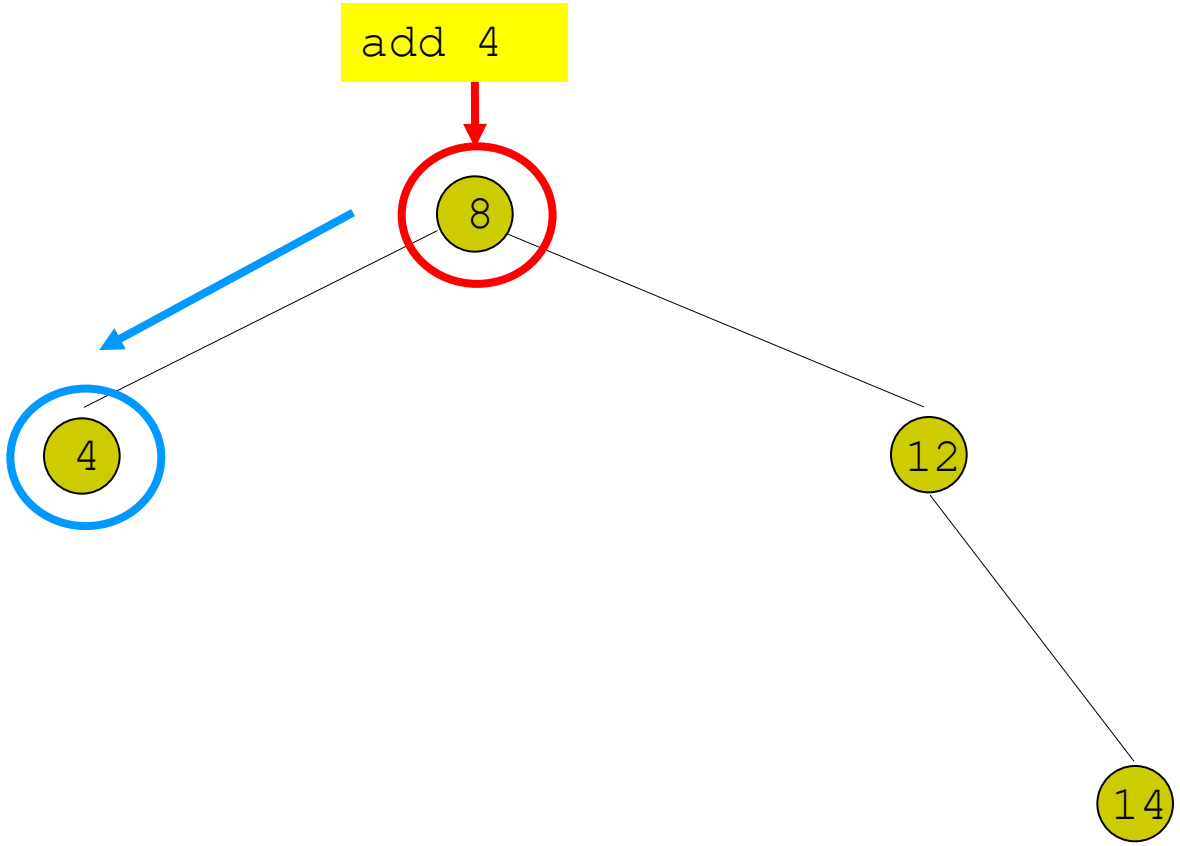
Προσθήκη κόμβου

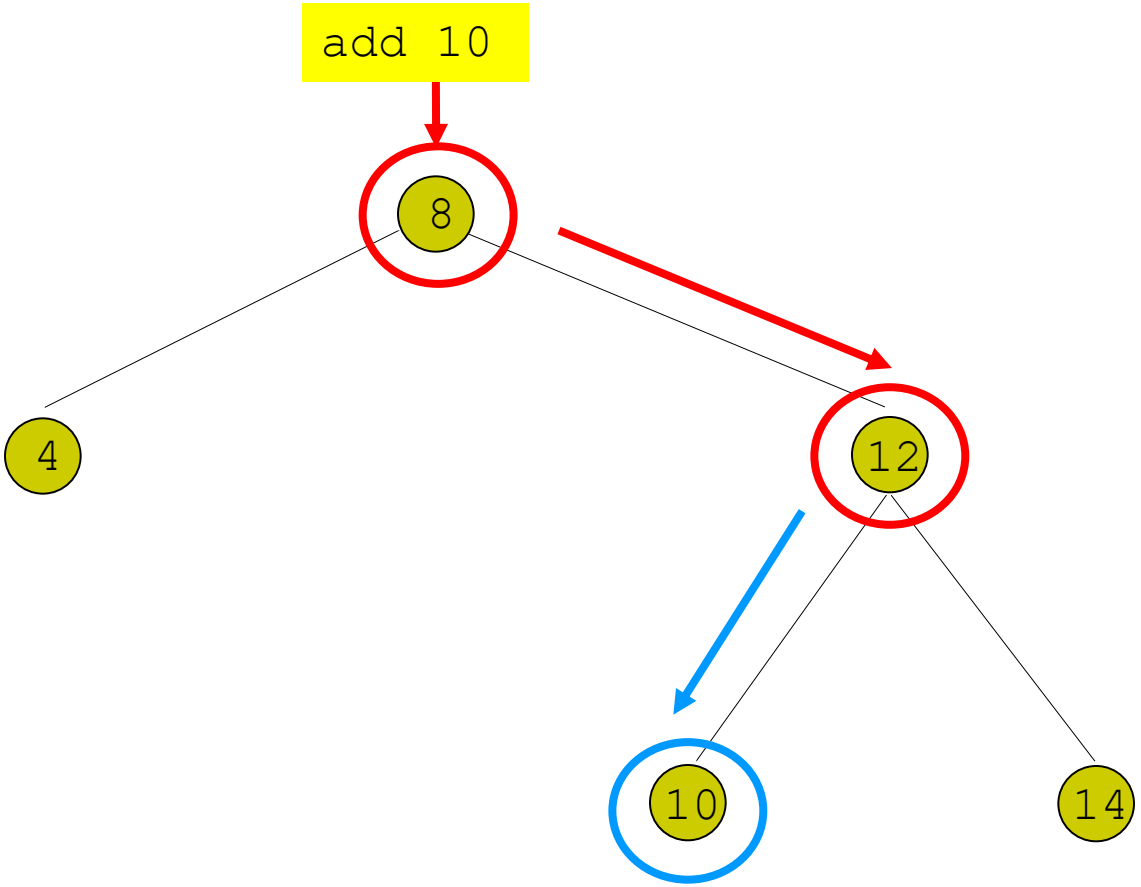
- Αναζήτηση με βάση το κλειδί v του κόμβου
- Αν ο κόμβος **βρεθεί**, γίνεται χειρισμός διπλότυπου
  - αποτυχία προσθήκης ή αντικατάσταση των δεδομένων
- Αν ο κόμβος **δεν βρεθεί**, τότε ο τελευταίος κόμβος που επισκευτήκαμε είναι ο **γονιός** του νέου κόμβου
- Ο νέος κόμβος εισάγεται ως **φύλλο**, **κάτω** από τον κόμβο όπου σταμάτησε η αναζήτηση
- Ως **ρίζα** του **αντίστοιχου** υποδέντρου, σύμφωνα με τη **σύμβαση αναζήτησης**
  - έστω ότι το κλειδί του φύλλου είναι v'
  - αν $v < v'$, ο νέος κόμβος γίνεται η ρίζα του υποδέντρου `left`
  - αν $v > v'$, ο νέος κόμβος γίνεται η ρίζα του υποδέντρου `right`
- Τα υποδέντρα του νέου κόμβου είναι κενά (`NULL`)

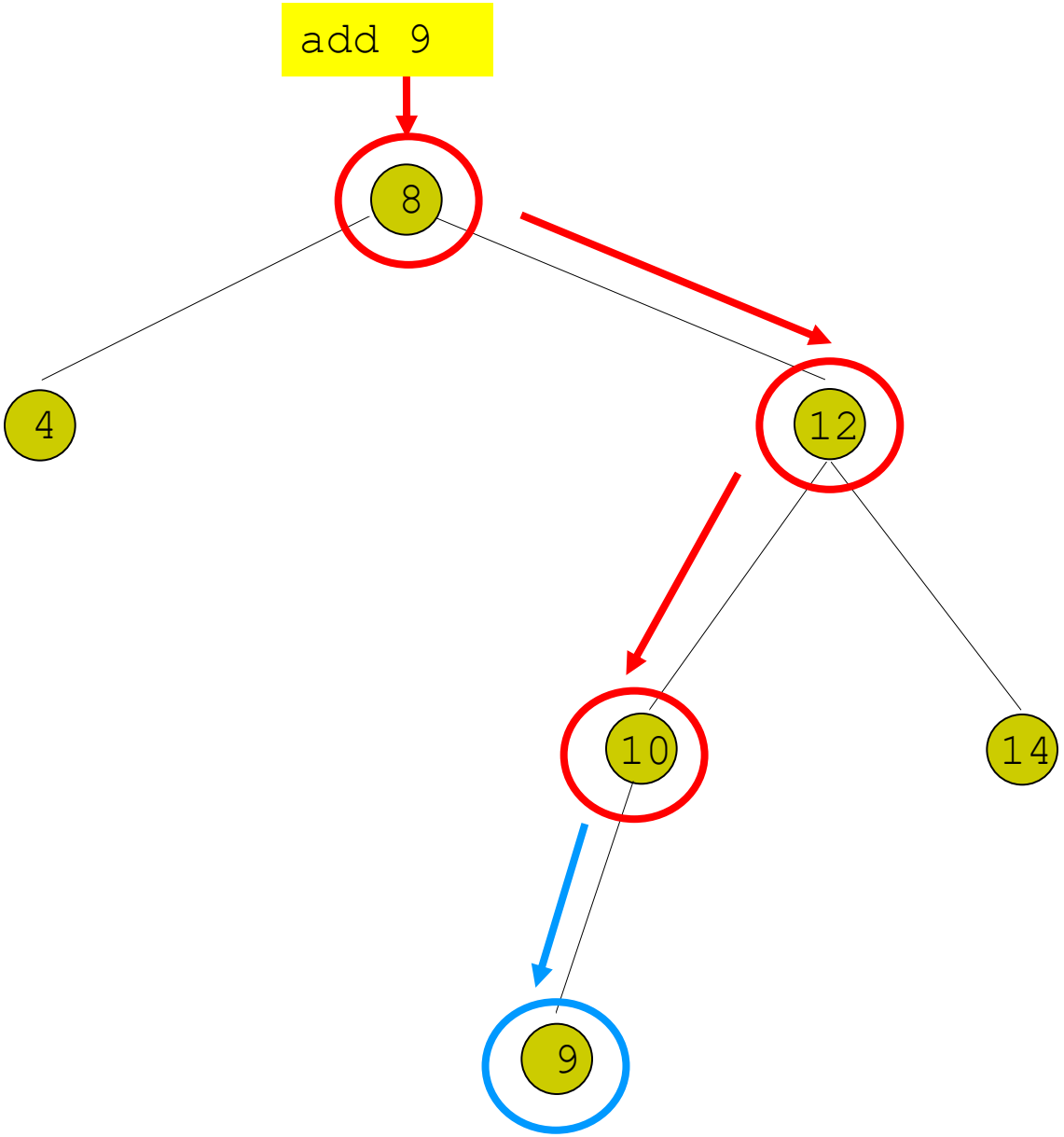

add 8
8

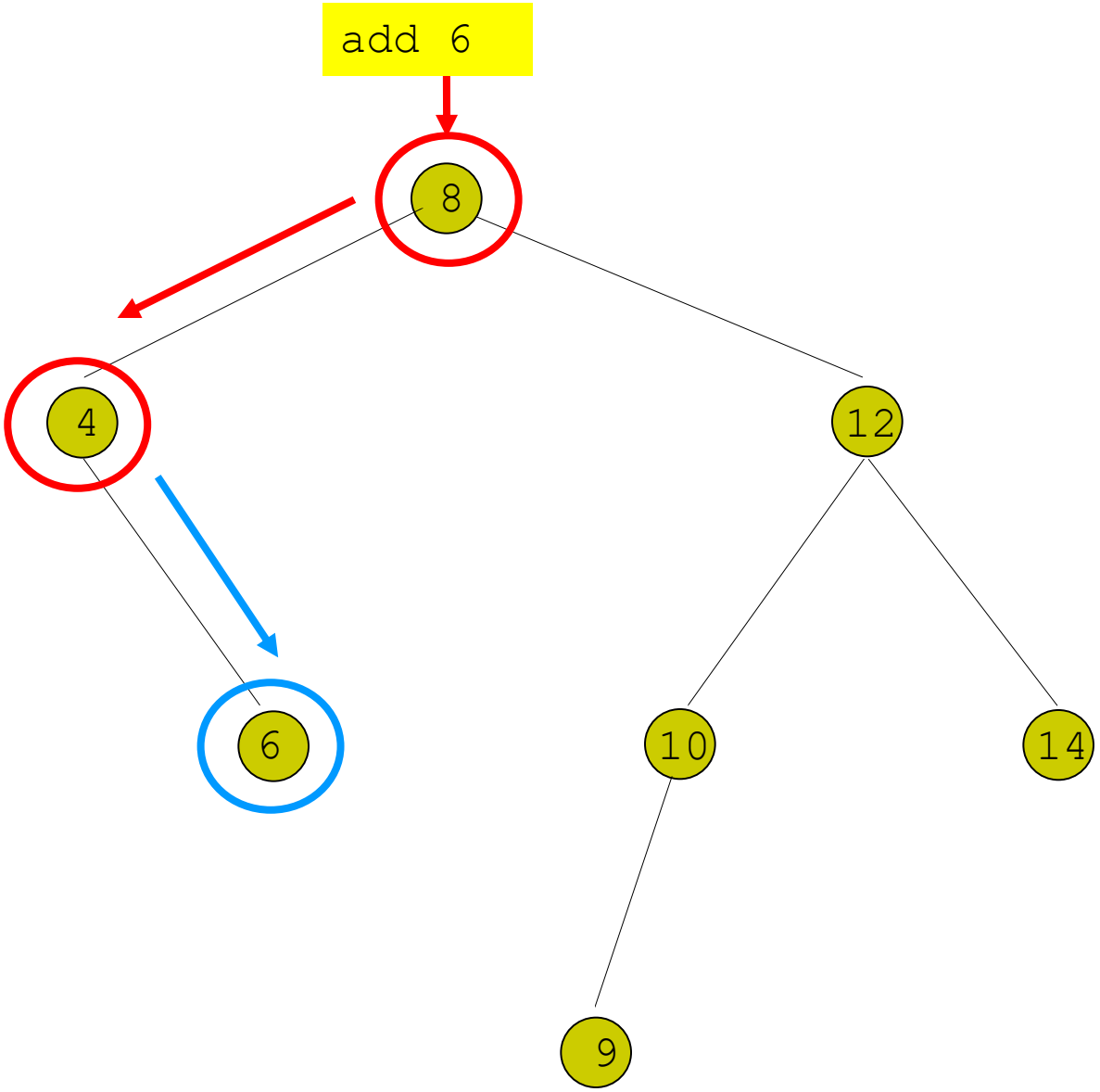

add 12
8
12

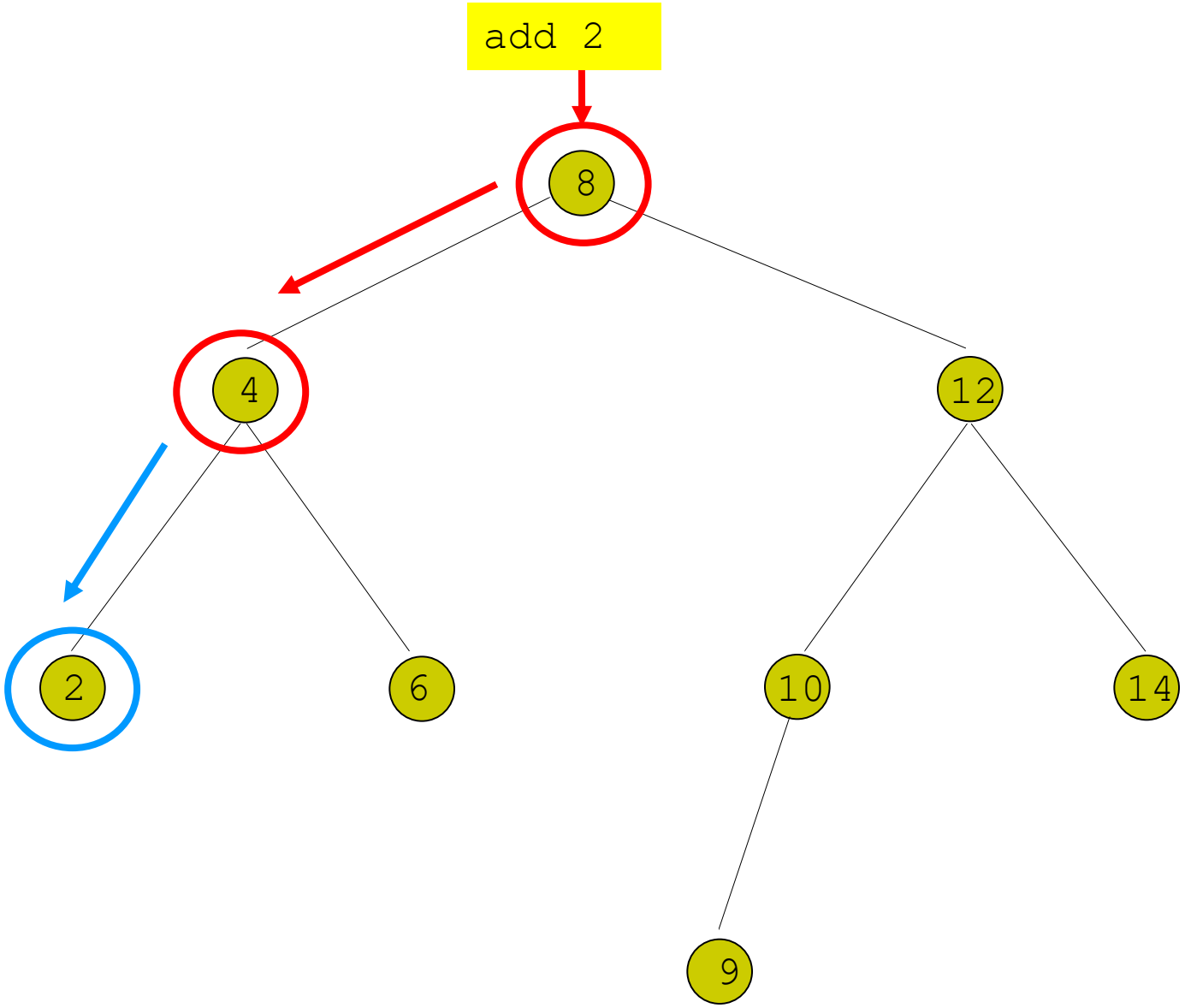

add 14
8
12
14

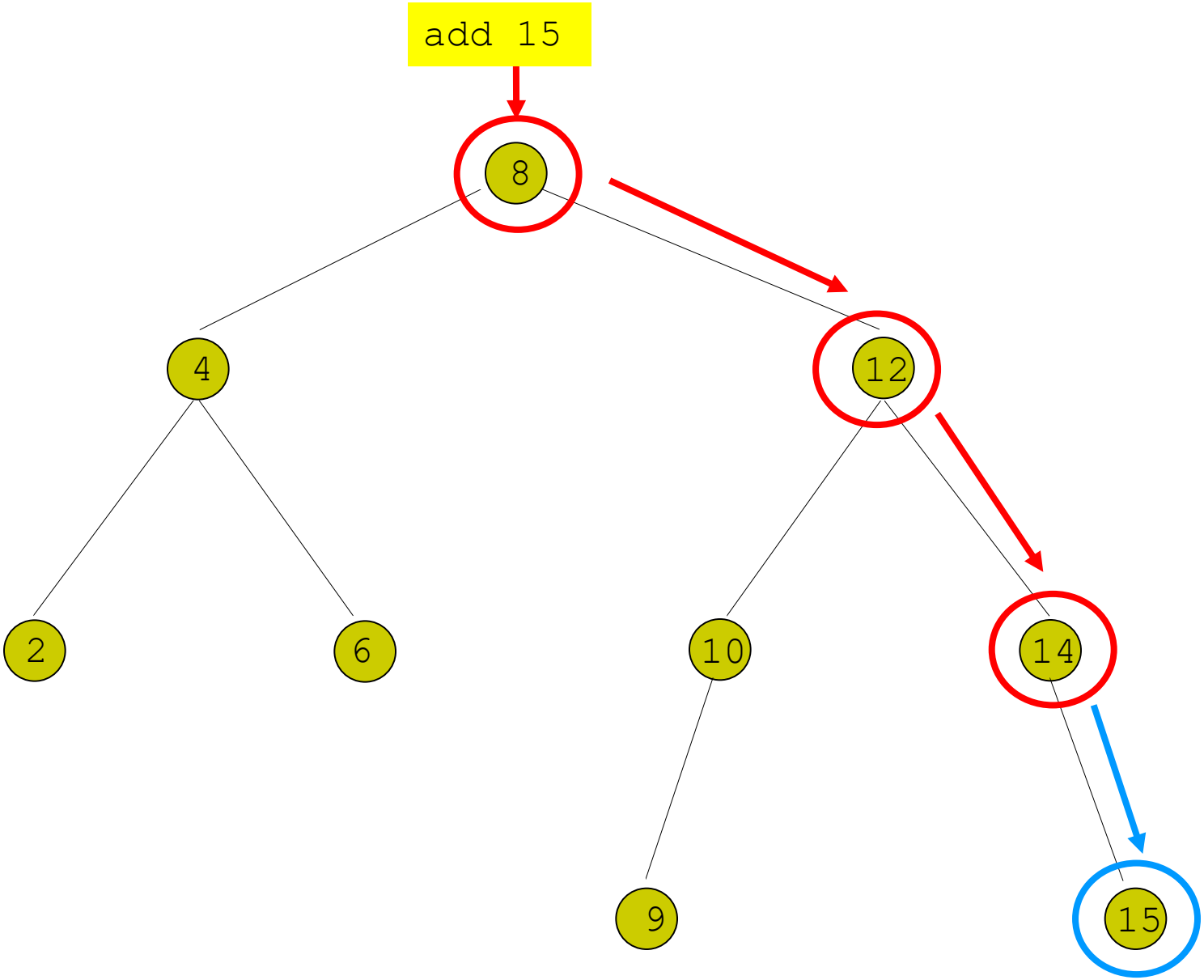

add 4
8
4
12
14

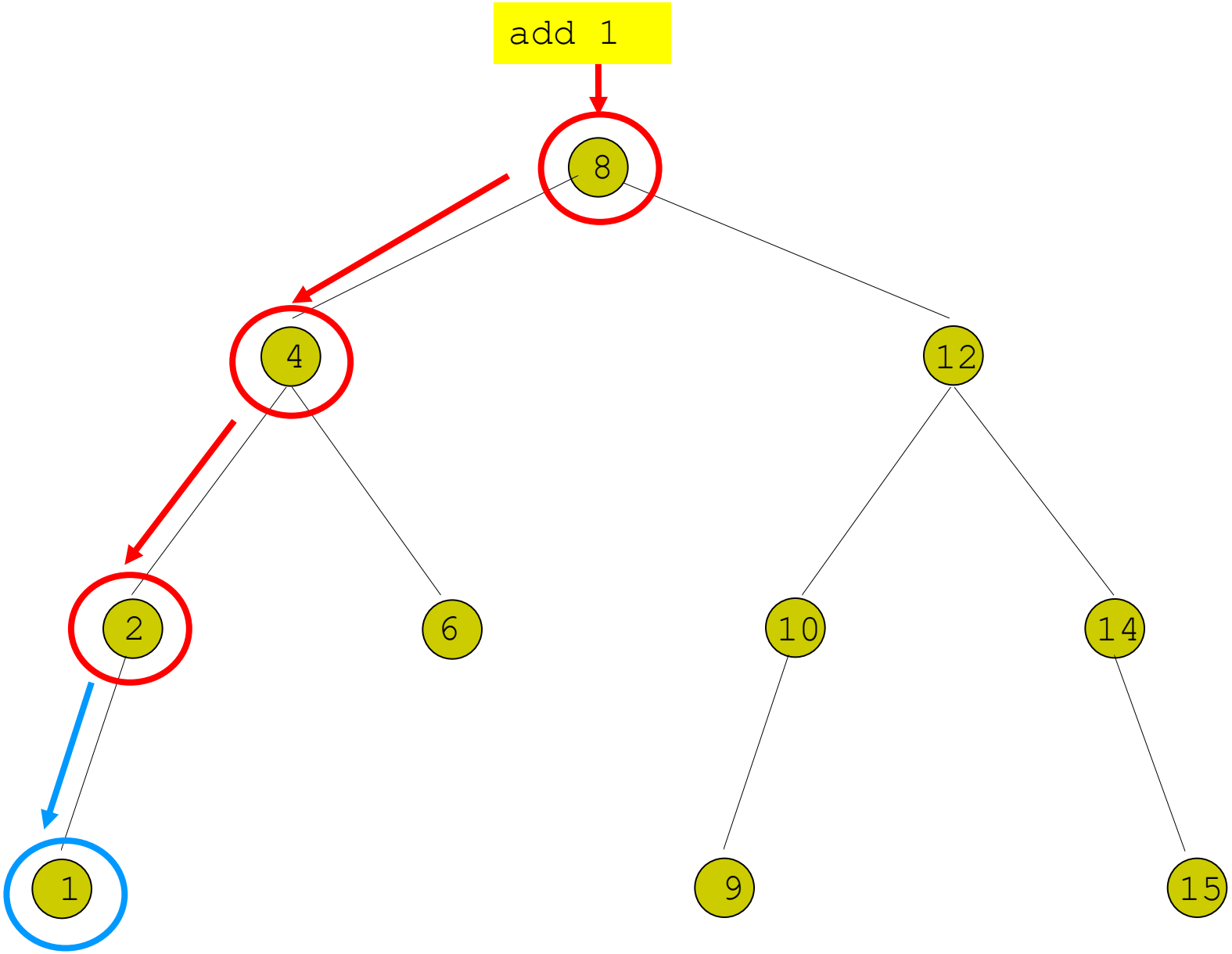

add 10
8
4
12
10
14

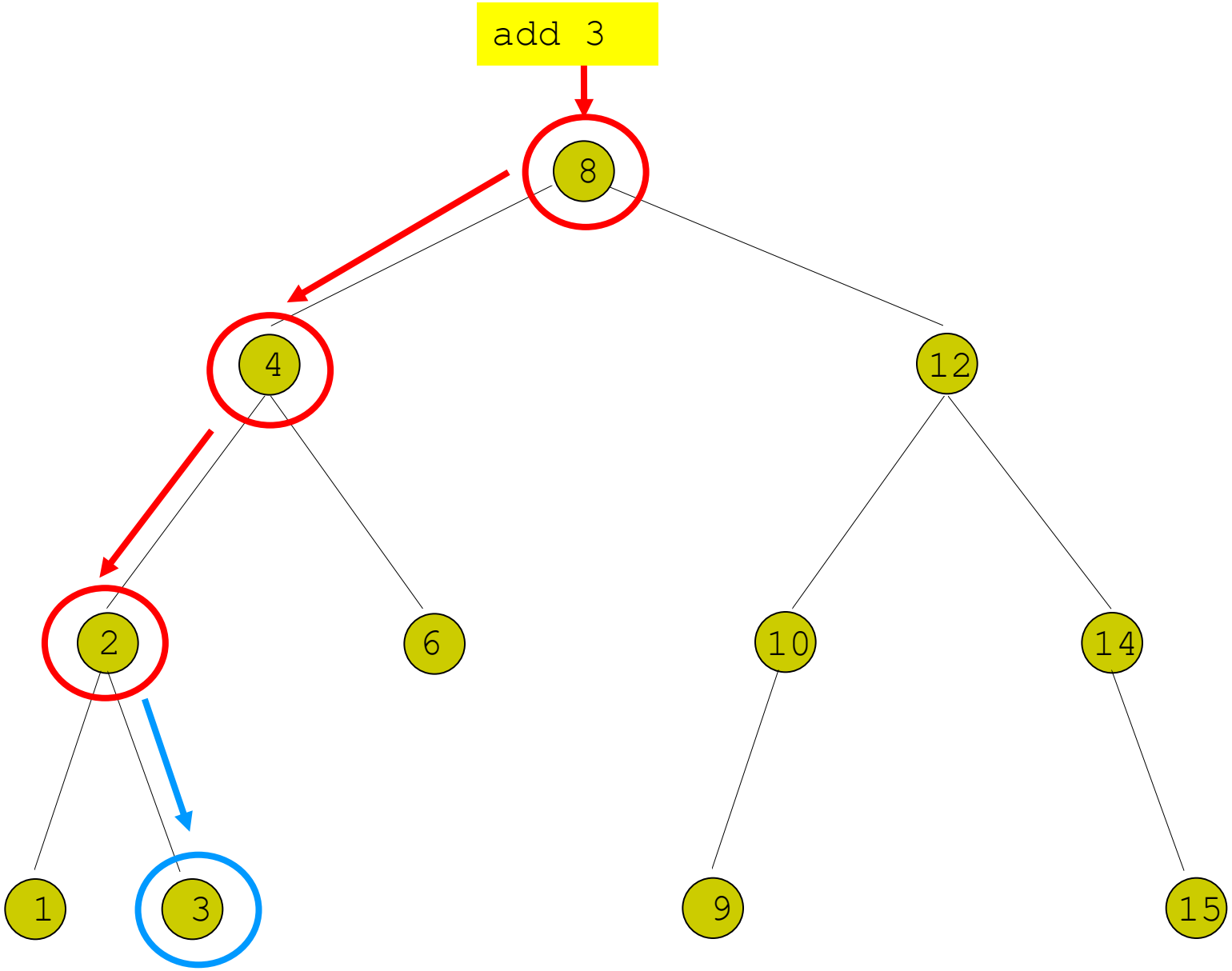

add 9
8
4
12
10
14
9

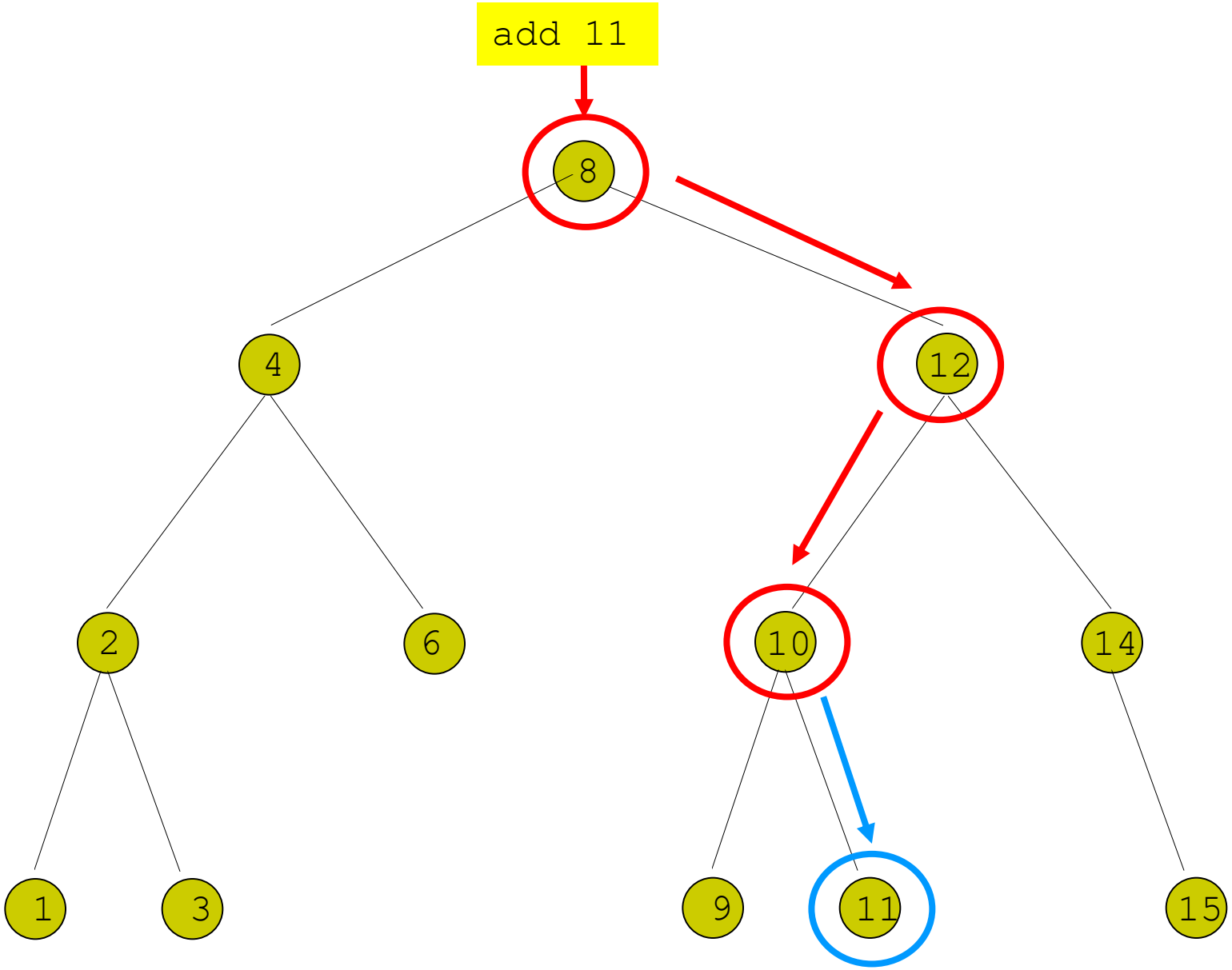

add 6
8
4
12
6
10
14
9

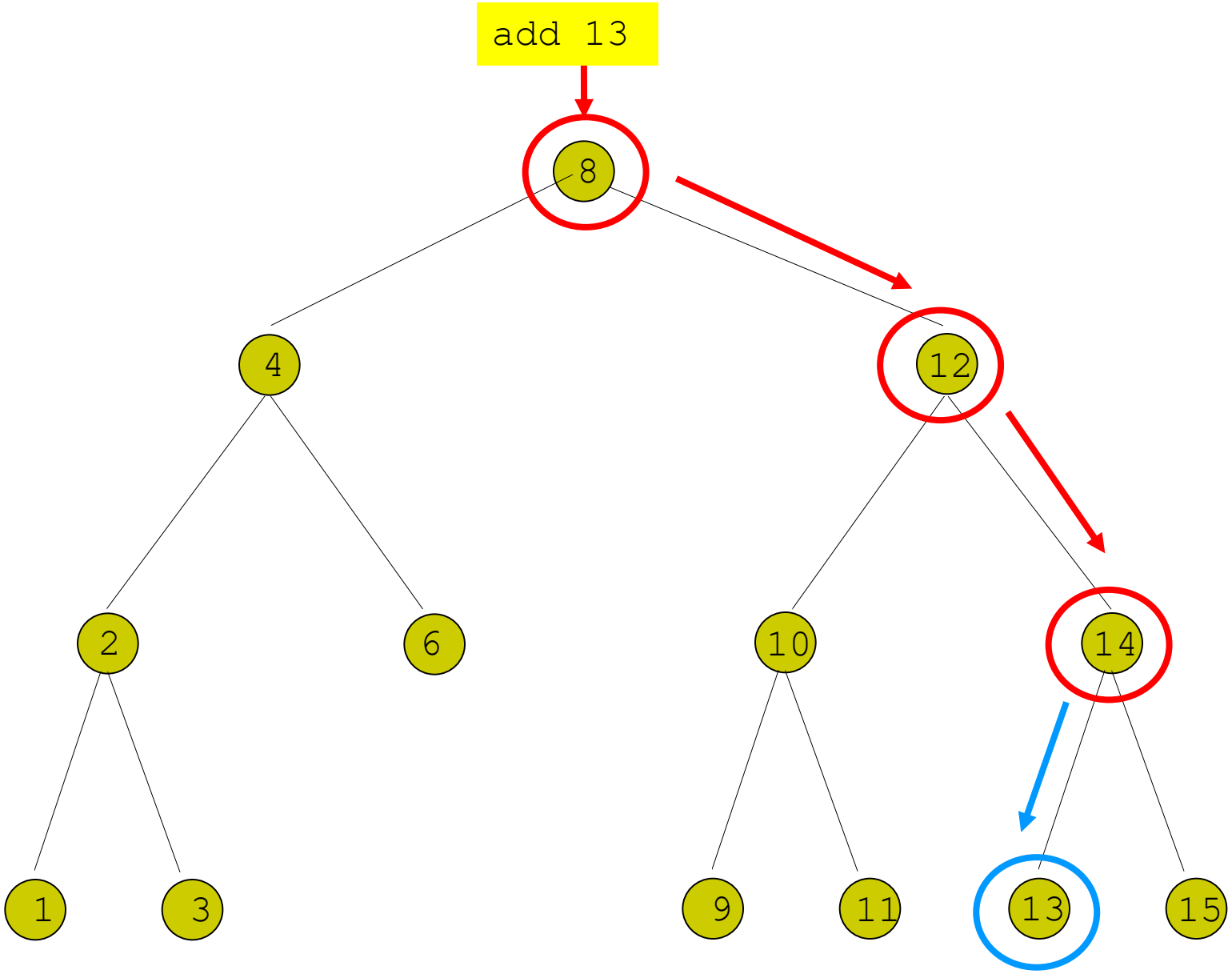

add 2
8
4
12
2
6
10
14
9

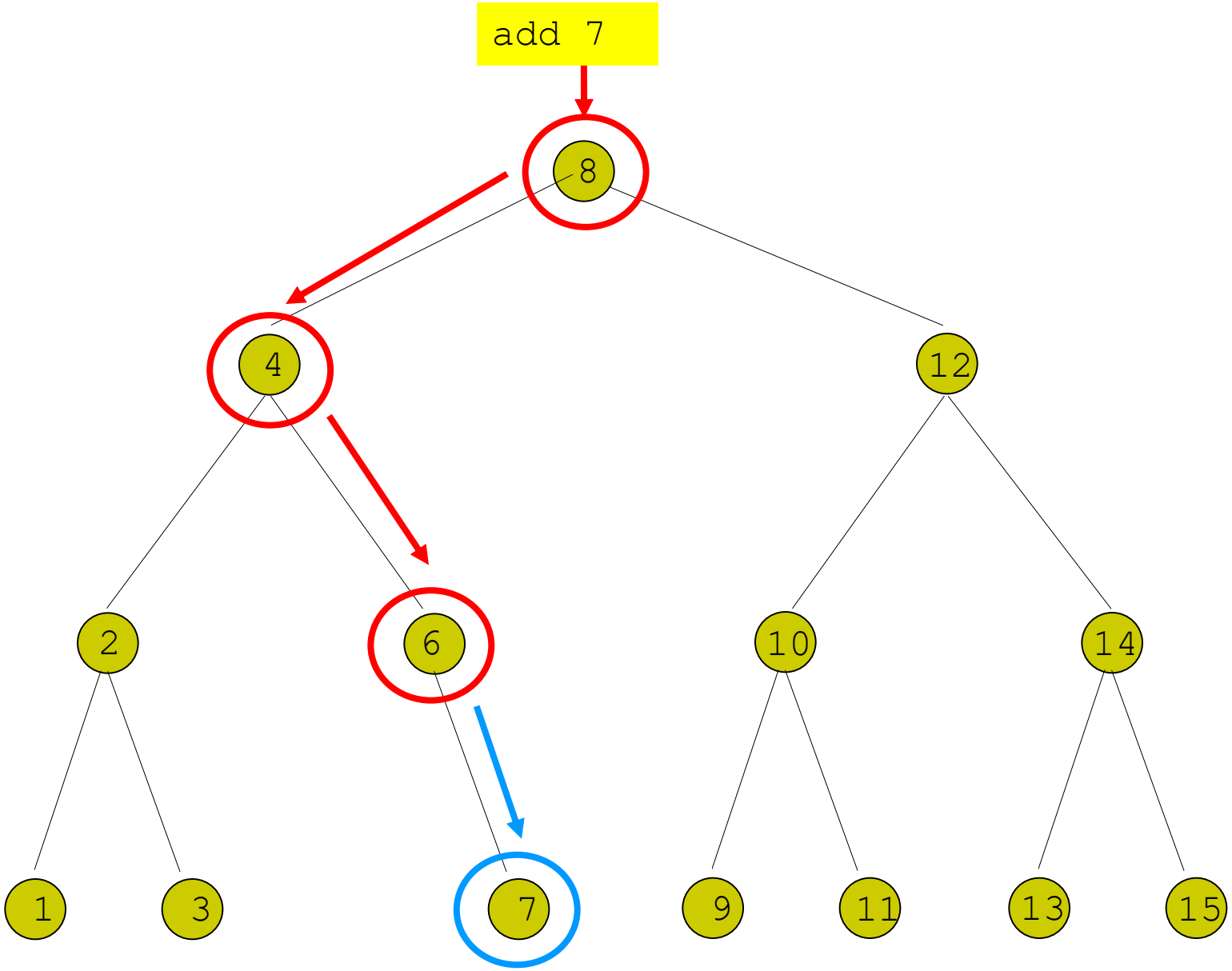

add 15
8
4
12
2
6
10
14
9
15

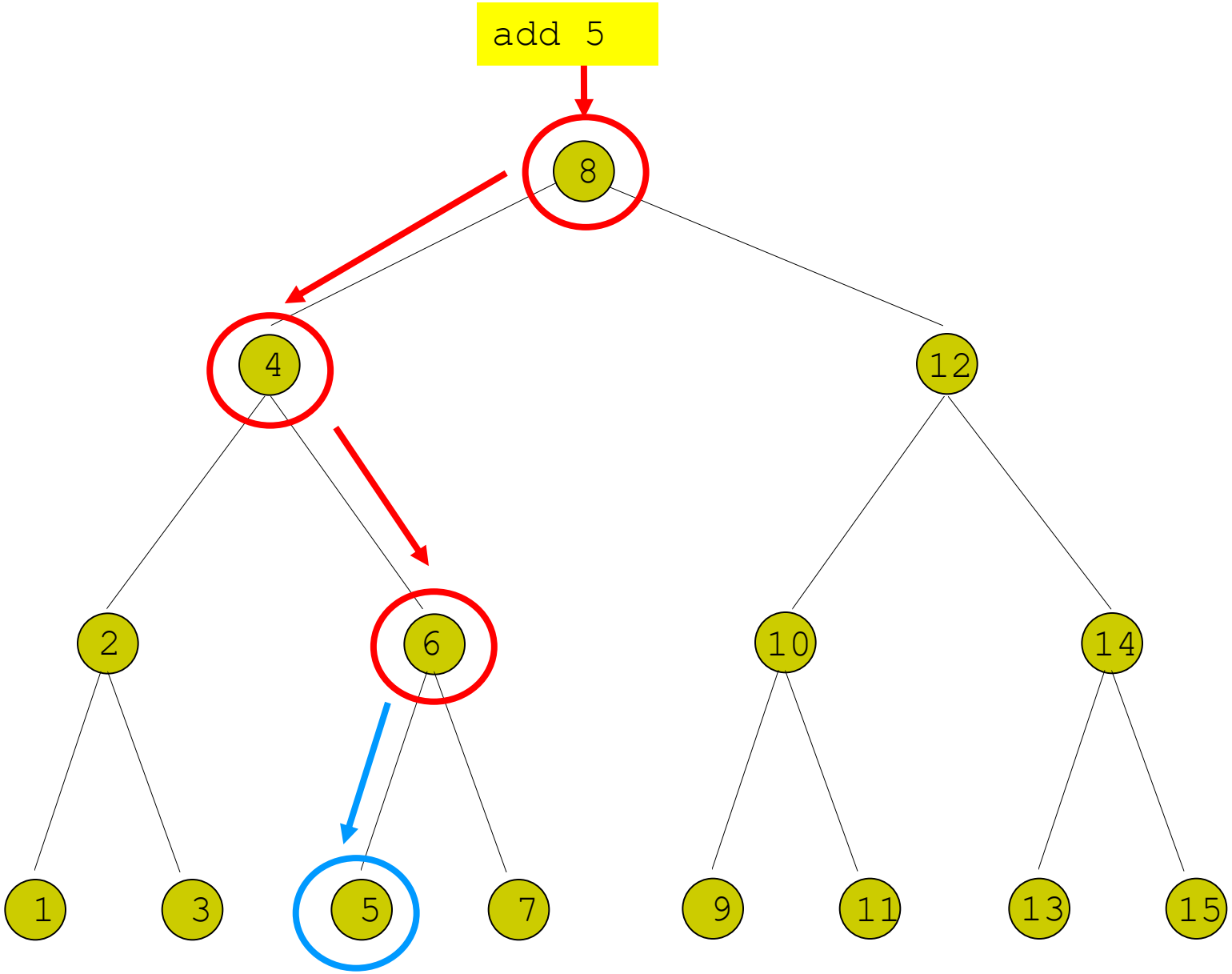

add 1
8
4
12
2
6
10
14
1
9
15


add 3
8
4
12
2
6
10
14
1
3
9
15


add 11
8
4
12
2
6
10
14
1
3
9
11
15


add 13
8
4
12
2
6
10
14
1
3
9
11
13
15


add 7
8
4
12
2
6
10
14
1
3
7
9
11
13
15


add 5
8
4
12
2
6
10
14
1
3
5
7
9
11
13
15

```
/*
 * if a node with <key> is found, return a pointer, else NULL;
 * always return the parent of the node where the search ended
 */
void btree_find0(btree *root, int key,
                 btree **node, btree **parent) {

  btree *curr=root, *curr_parent=NULL;

  while ((curr != NULL) && (curr->key != key)) {
     curr_parent = curr;
     if (key < curr->key) { curr = curr->left; }
     else { curr = curr->right; }
  }

  if ((curr != NULL) && (curr->key == key)) {
    *node = curr;
  }
  else {
    *node = NULL;
  }
  *parent = curr_parent;
}
```

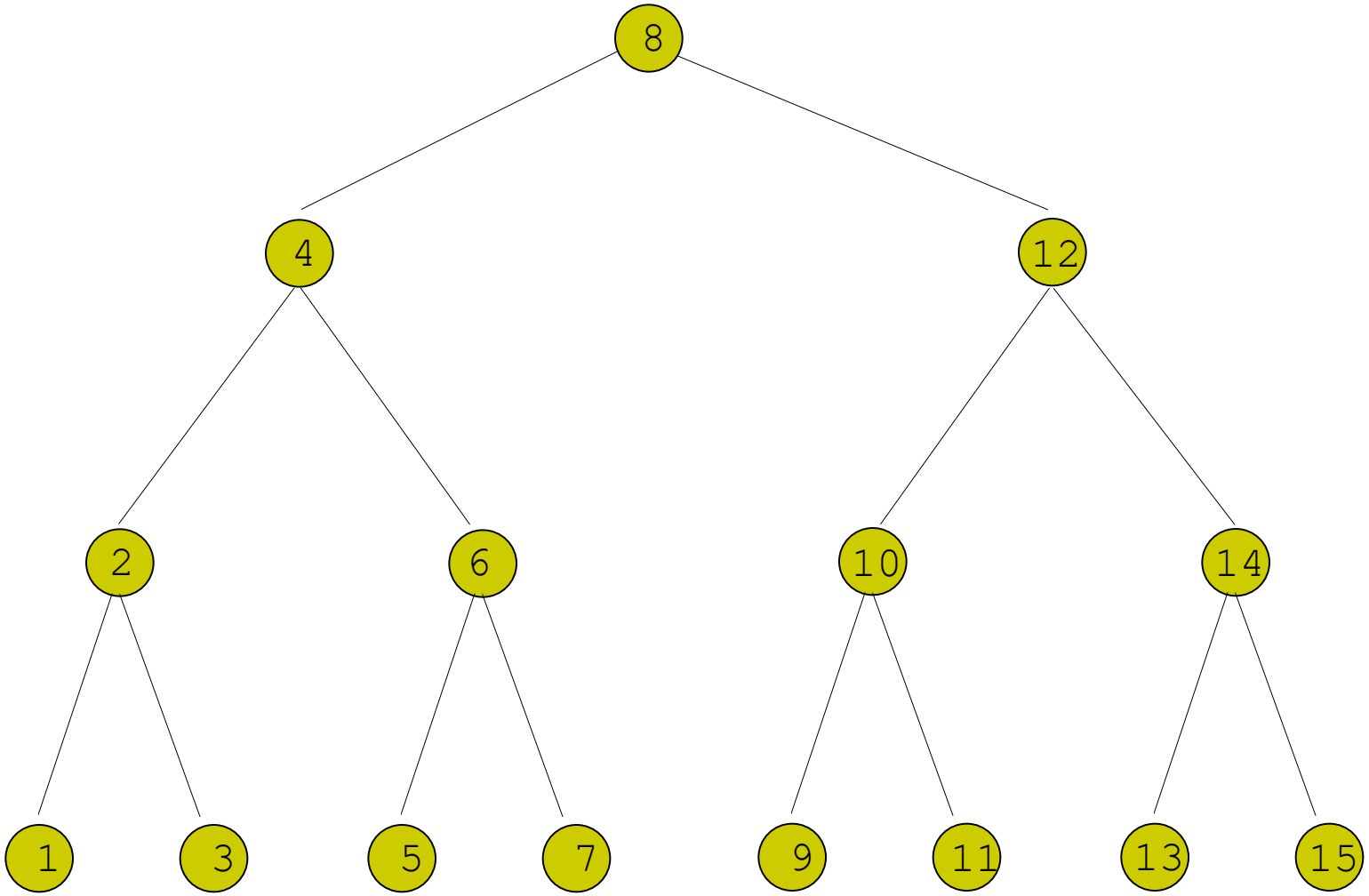
```
int btree_insert(btree **root, btree *node) {
  btree *curr, *curr_parent;

  btree_find0(*root, node->key, &curr, &curr_parent);

  if ((curr != NULL) && (curr->key == node->key)) {
    return(0); // no dublicates allowed
  }

  node->left = NULL;
  node->right = NULL;

  if (curr_parent == NULL) {
    *root = node;
  }
  else if (node->key < curr_parent->key) {
    curr_parent->left = node;
  }
  else {
    curr_parent->right = node;
  }
  return(1);
}
```

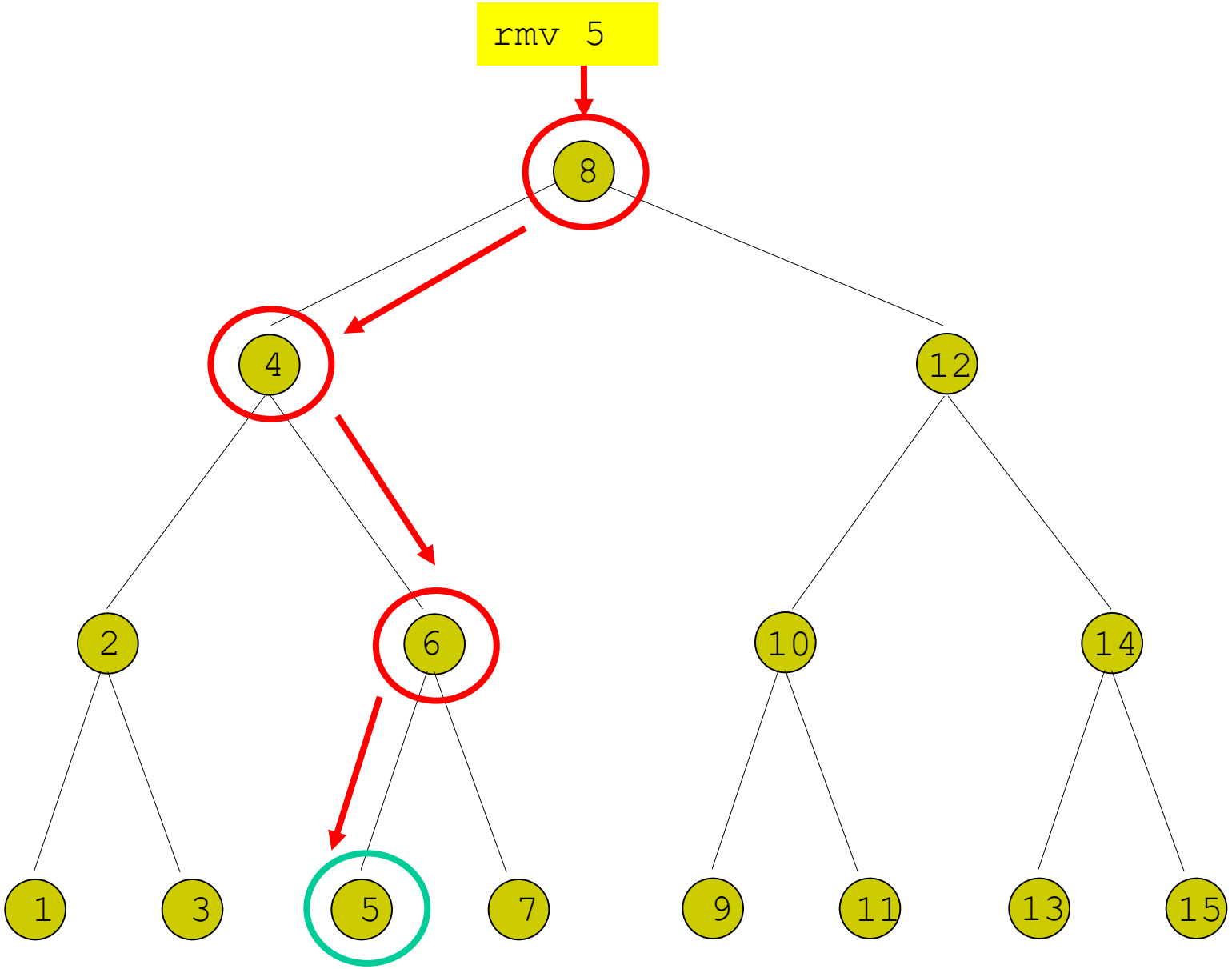
Αφαίρεση κόμβου (απλή περίπτωση)

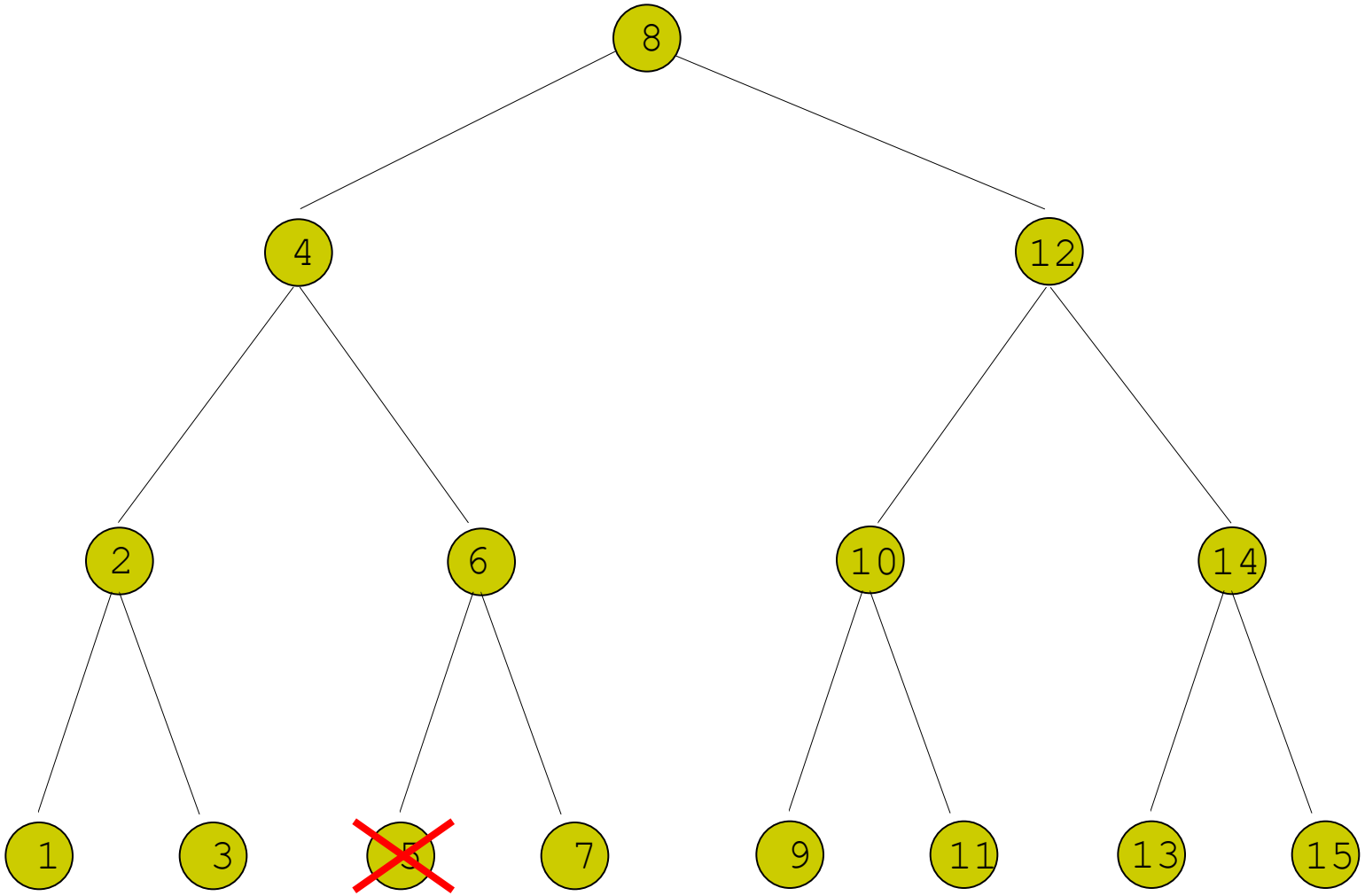
- Αναζήτηση με βάση τον κλειδί `v`
- Αν ο κόμβος δεν βρεθεί, τερματίζουμε

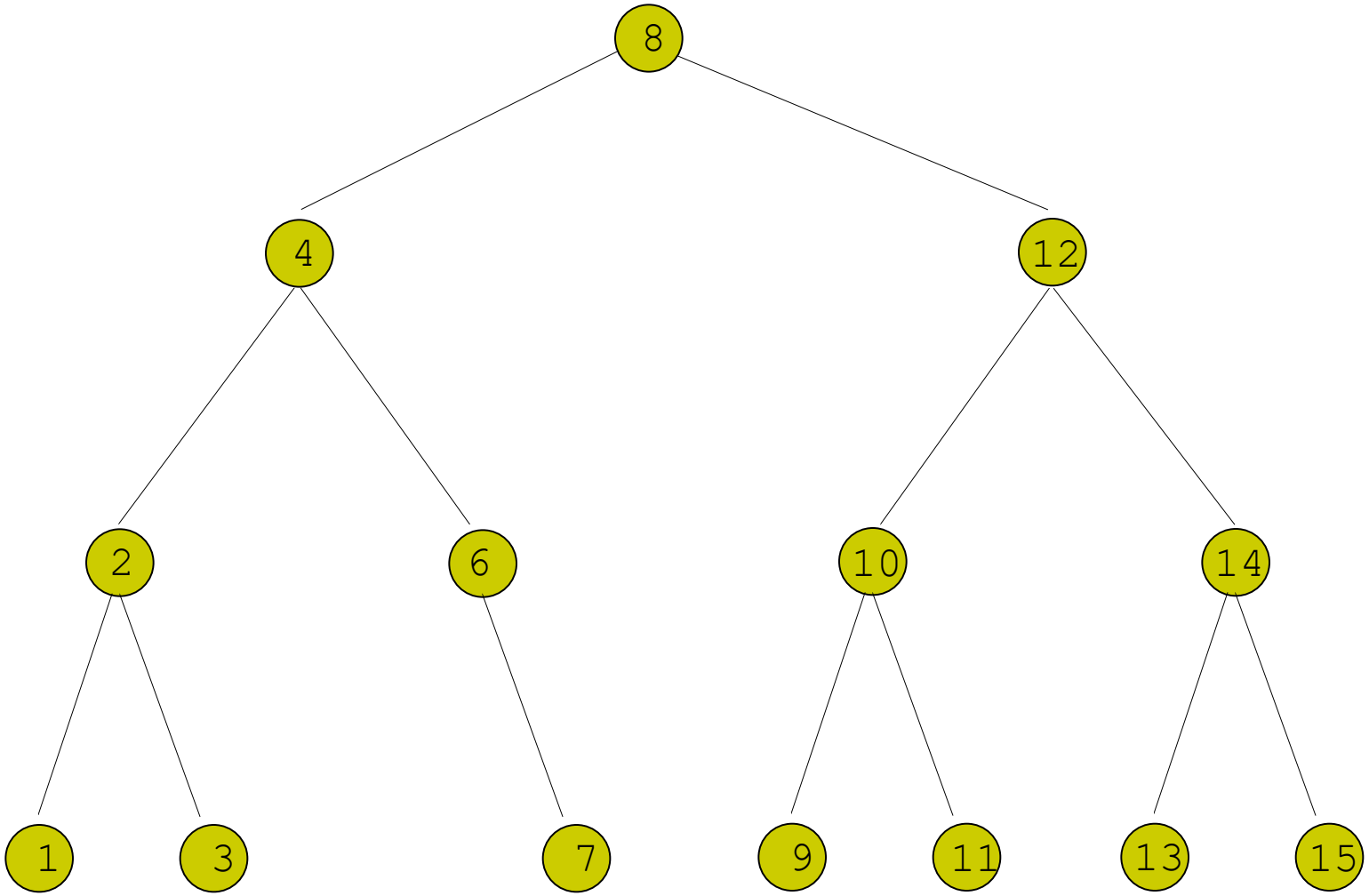
- Έστω ότι ο κόμβος βρέθηκε και είναι **φύλλο**
- Αλλάζουμε τον δείκτη στο **αντίστοιχο υποδέντρο του γονιού** του σε `NULL`

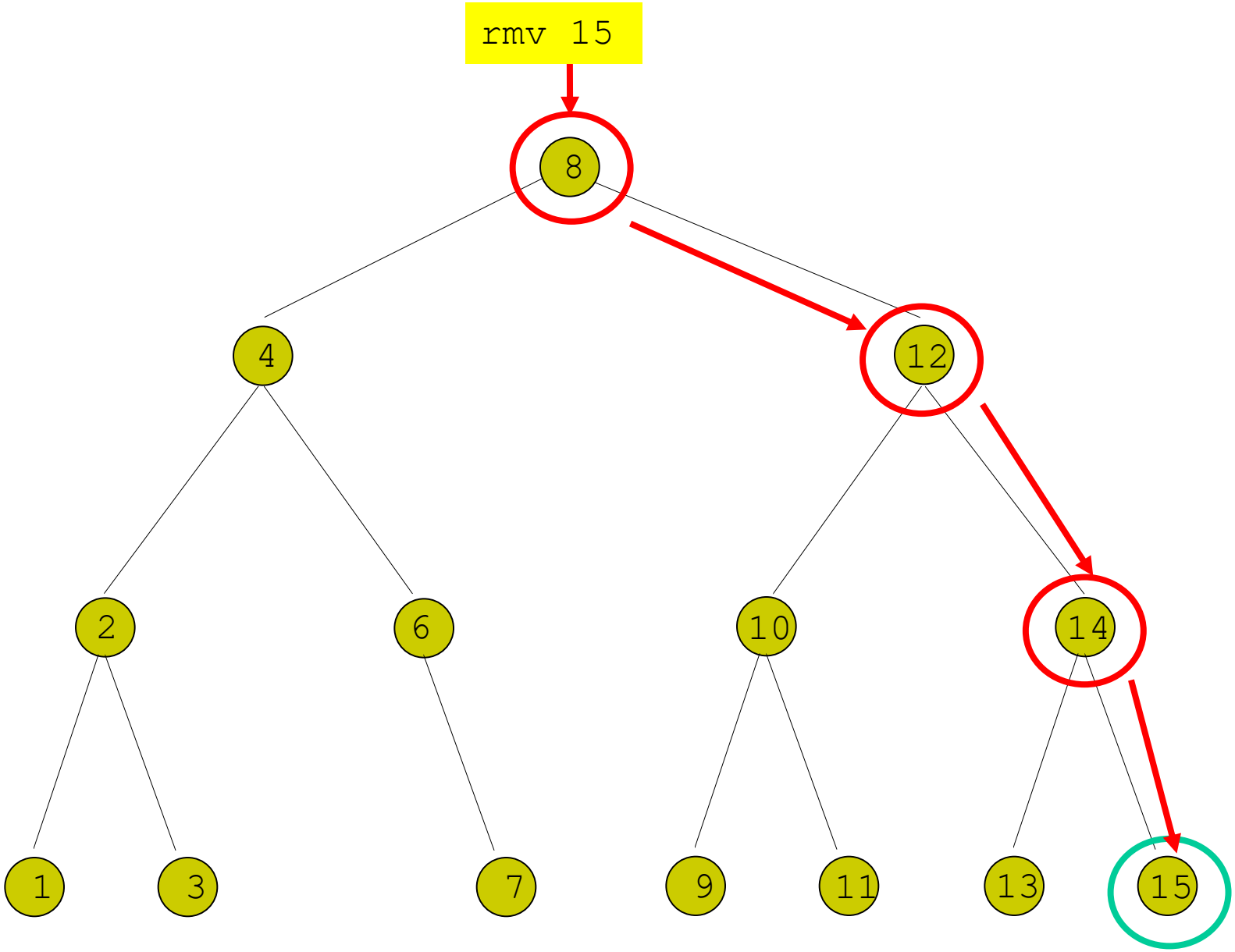
- **Ειδική περίπτωση:** ο κόμβος είναι η **ρίζα** του δέντρου (**δεν** έχει γονιό)
- Αλλάζουμε την ρίζα σε `NULL`

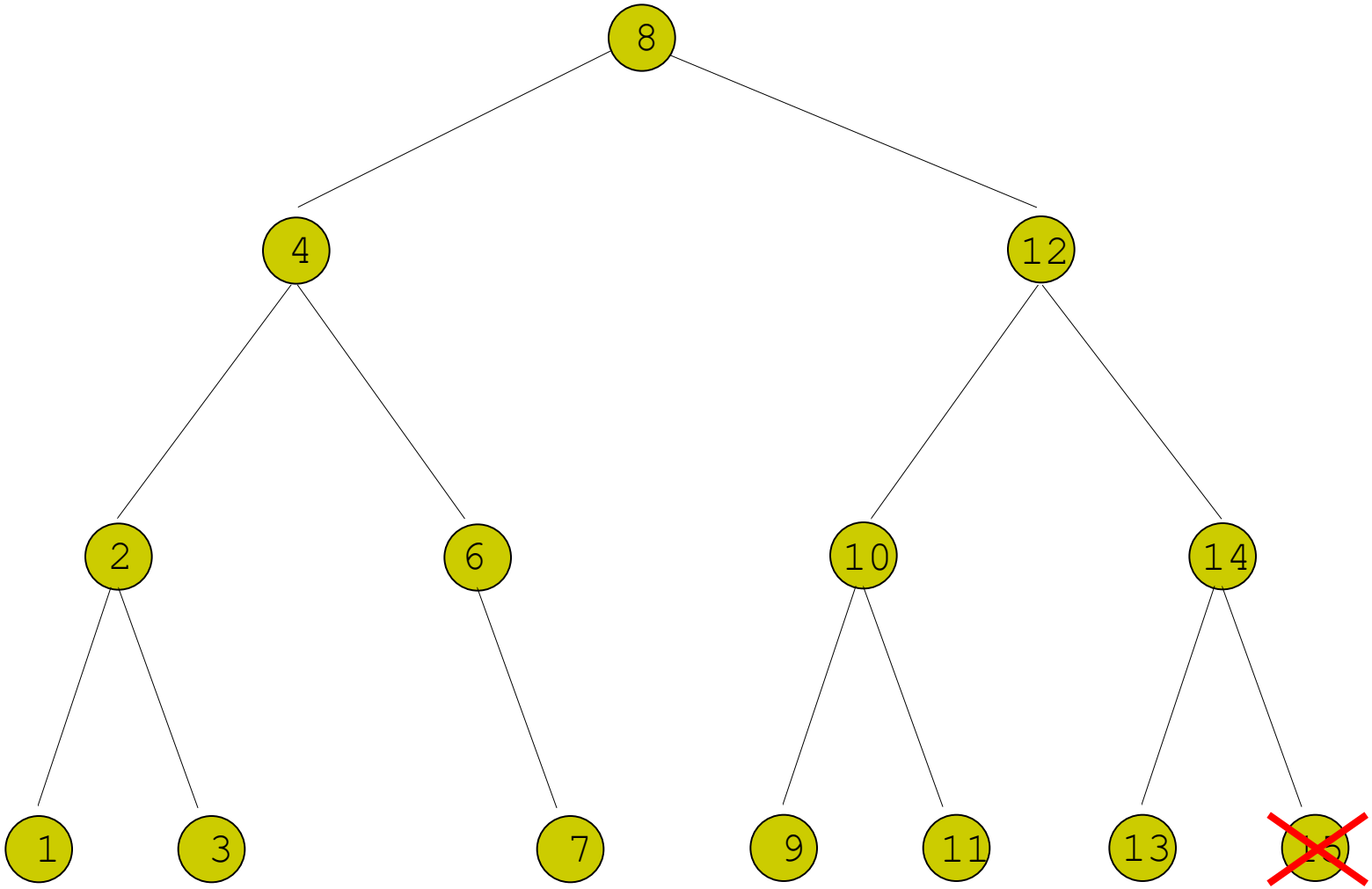

8
4
12
2
6
10
14
1
3
5
7
9
11
13
15

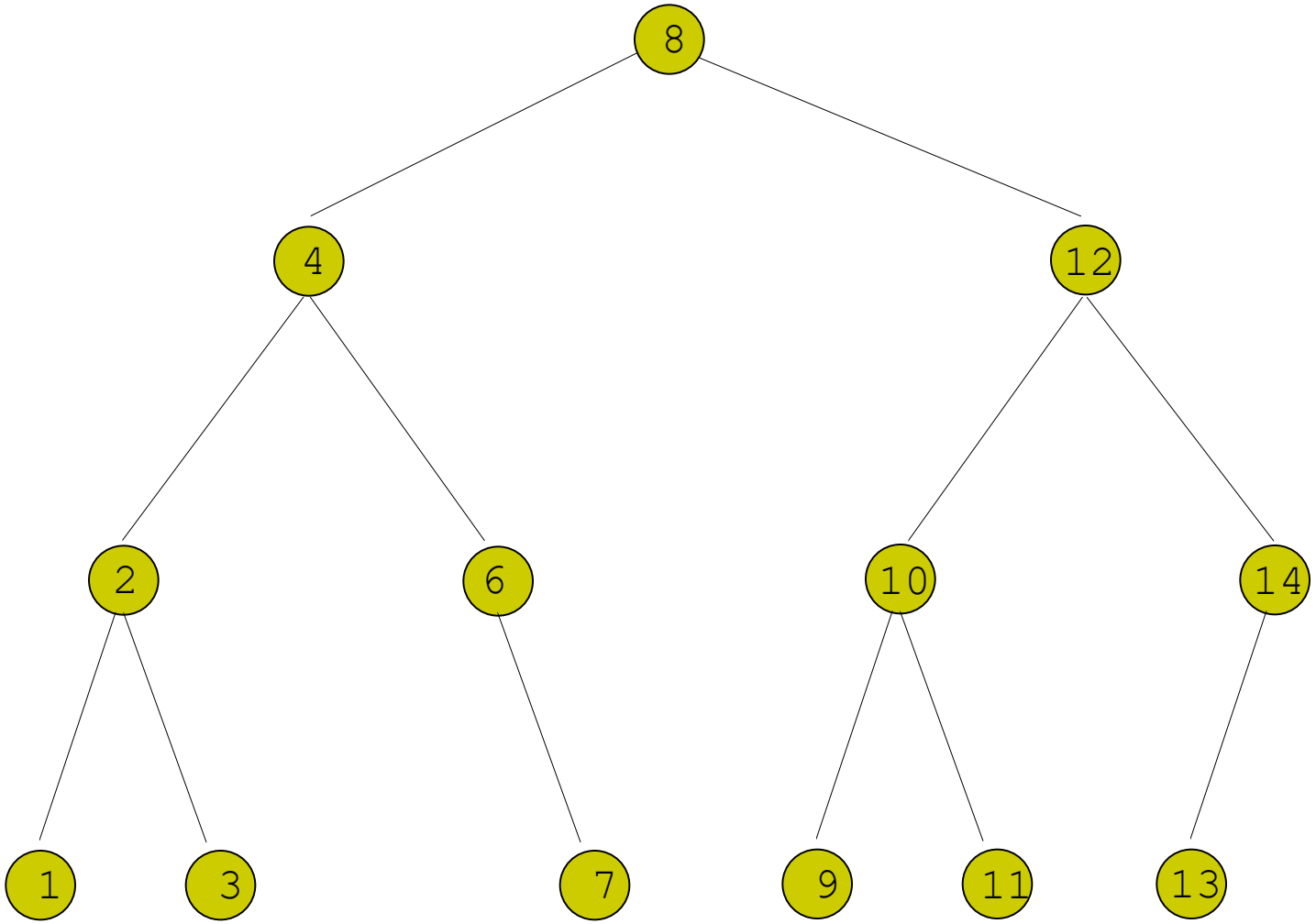

rmv 5
8
4
12
2
6
10
14
1
3
5
7
9
11
13
15


8
4
12
2
6
10
14
1
3
5
7
9
11
13
15


8
4
12
2
6
10
14
1
3
7
9
11
13
15


rmv 15
8
4
12
2
6
10
14
1
3
7
9
11
13
15


8
4
12
2
6
10
14
1
3
7
9
11
13
15

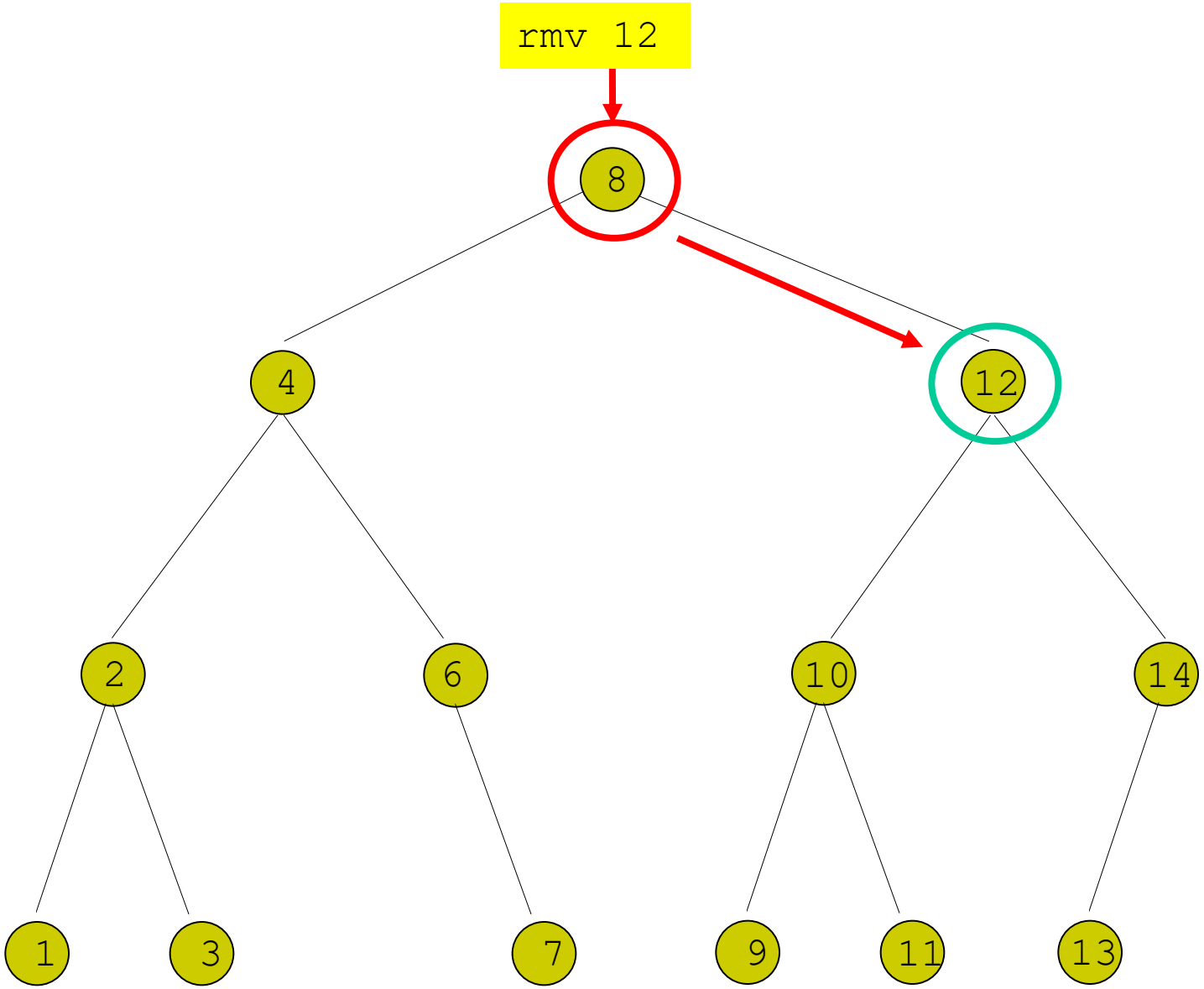

8
4
12
2
6
10
14
1
3
7
9
11
13

```
btree *btree_remove(struct btree **root, int key) {
  struct btree *curr, *curr_parent;

  btree_find0(*root, key, &curr, &curr_parent);
  if (curr == NULL) { return(NULL); }

  if ((curr->left != NULL) || (curr->right != NULL)) {
    return(NULL); /* not a leaf node */
  }

  if (curr_parent == NULL) {
    *root = NULL;
  }
  else if (curr_parent->left == curr) {
    curr_parent->left=NULL;
  }
  else {
    curr_parent->right = NULL;
  }

  return(curr);
}
```

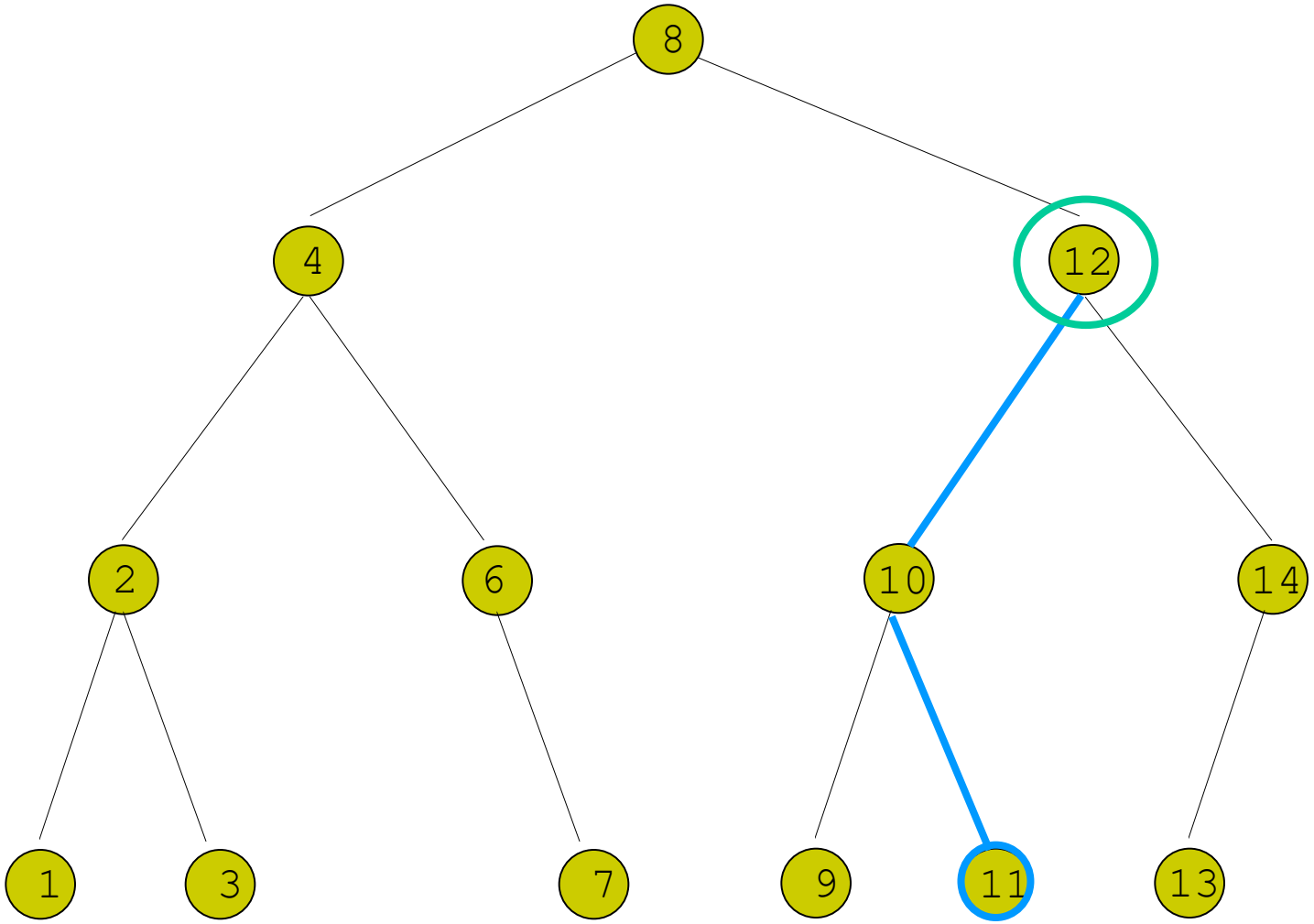
Αφαίρεση κόμβου (γενική περίπτωση)

- Αναζήτηση με βάση τον κλειδί v
- Αν ο κόμβος δεν βρεθεί, τερματίζουμε

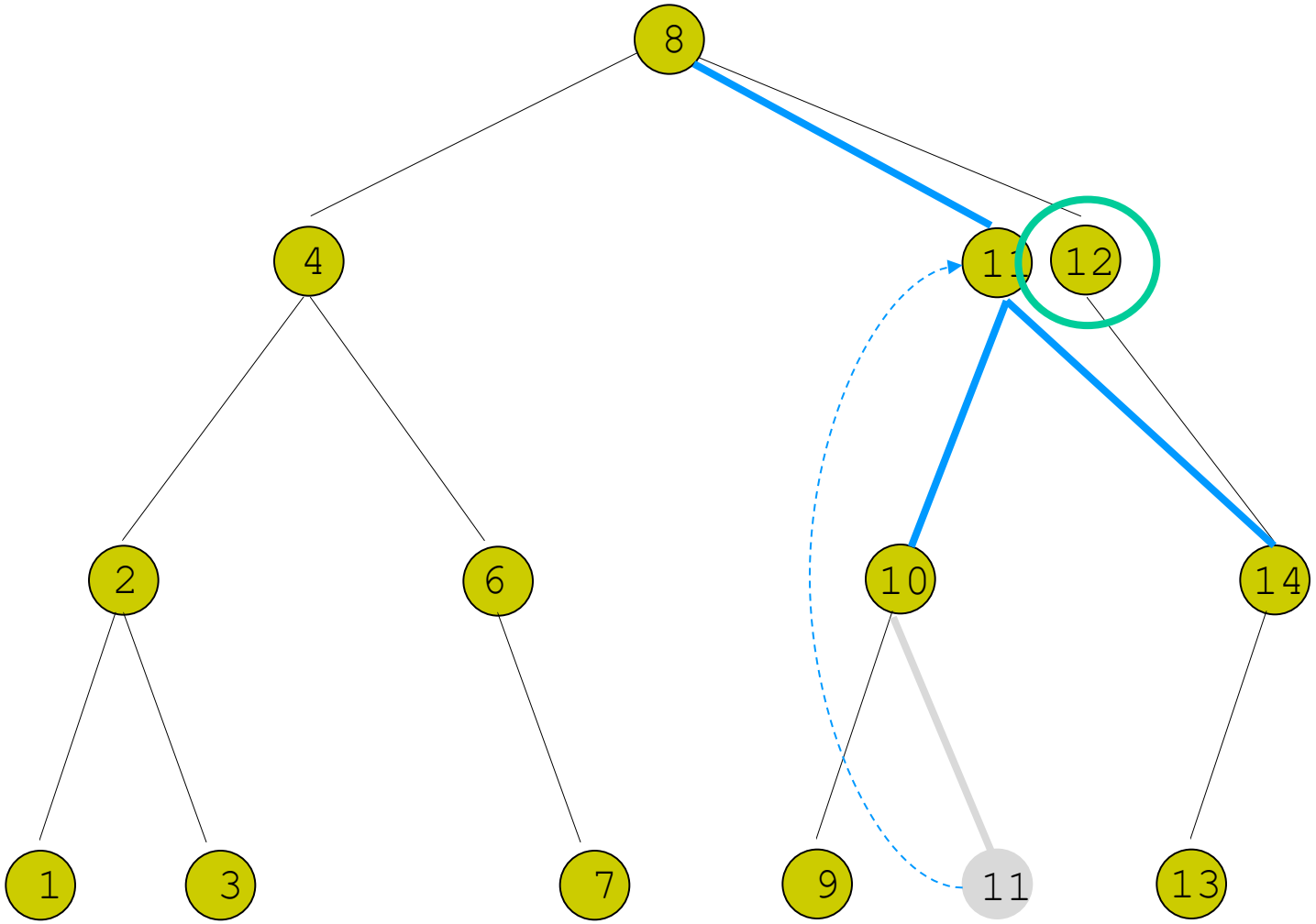
- Έστω ότι ο κόμβος βρέθηκε και **δεν** είναι **φύλλο**
- Τότε είναι η **ρίζα** ενός υποδέντρου
- Πρέπει να **αντικατασταθεί** με κάποιον κόμβο έτσι ώστε να **διατηρείται** η συνθήκη της αναζήτησης
- **Προάγουμε** ως νέα ρίζα του αντίστοιχου υποδέντρου τον **μεγαλύτερο** κόμβο του **αριστερού** υποδέντρου ή τον **μικρότερο** κόμβο του **δεξιού** υποδέντρου του κόμβου προς απομάκρυνση


rmv 12
8
4
12
2
6
10
14
1
3
7
9
11
13

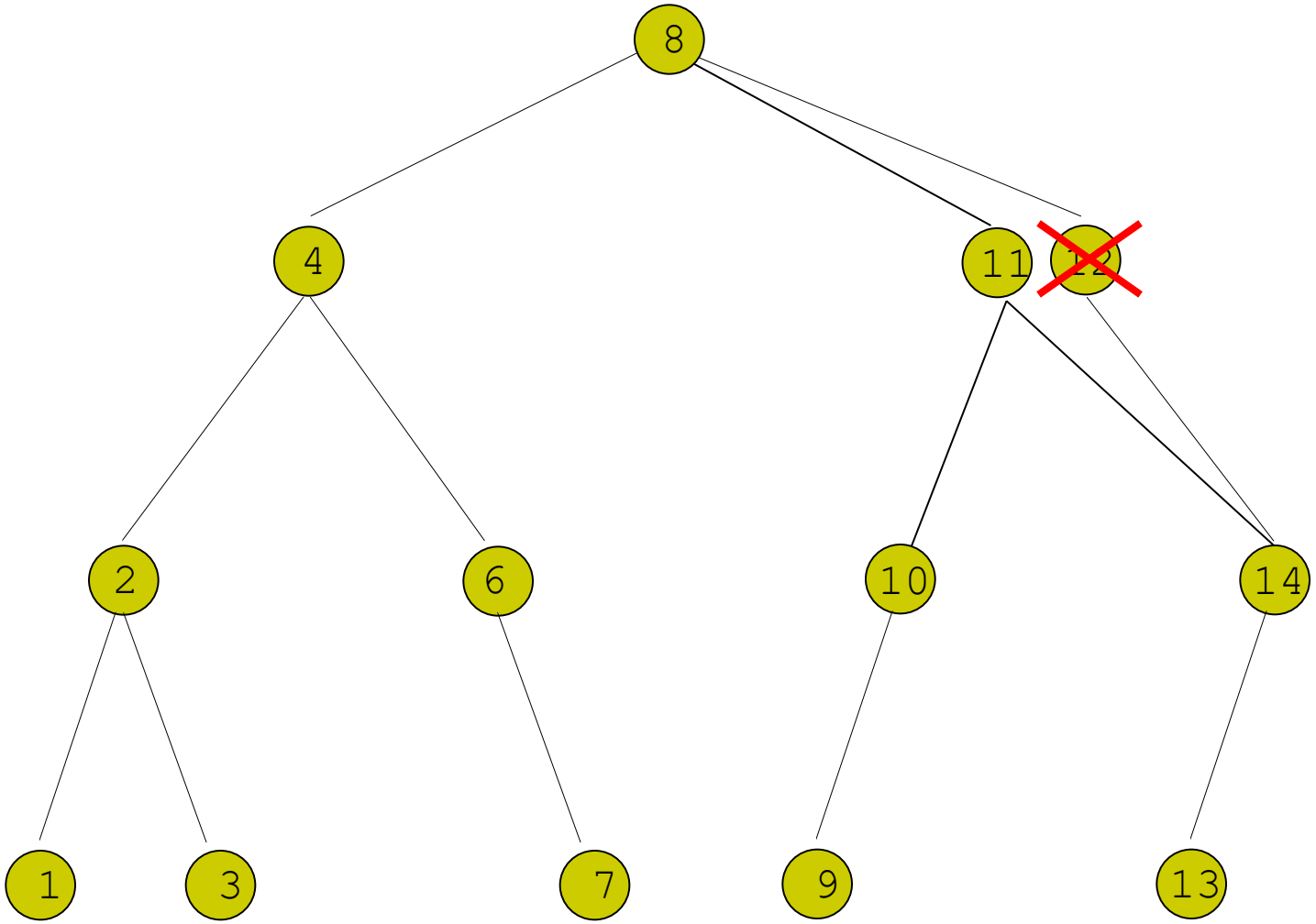
rmv 12

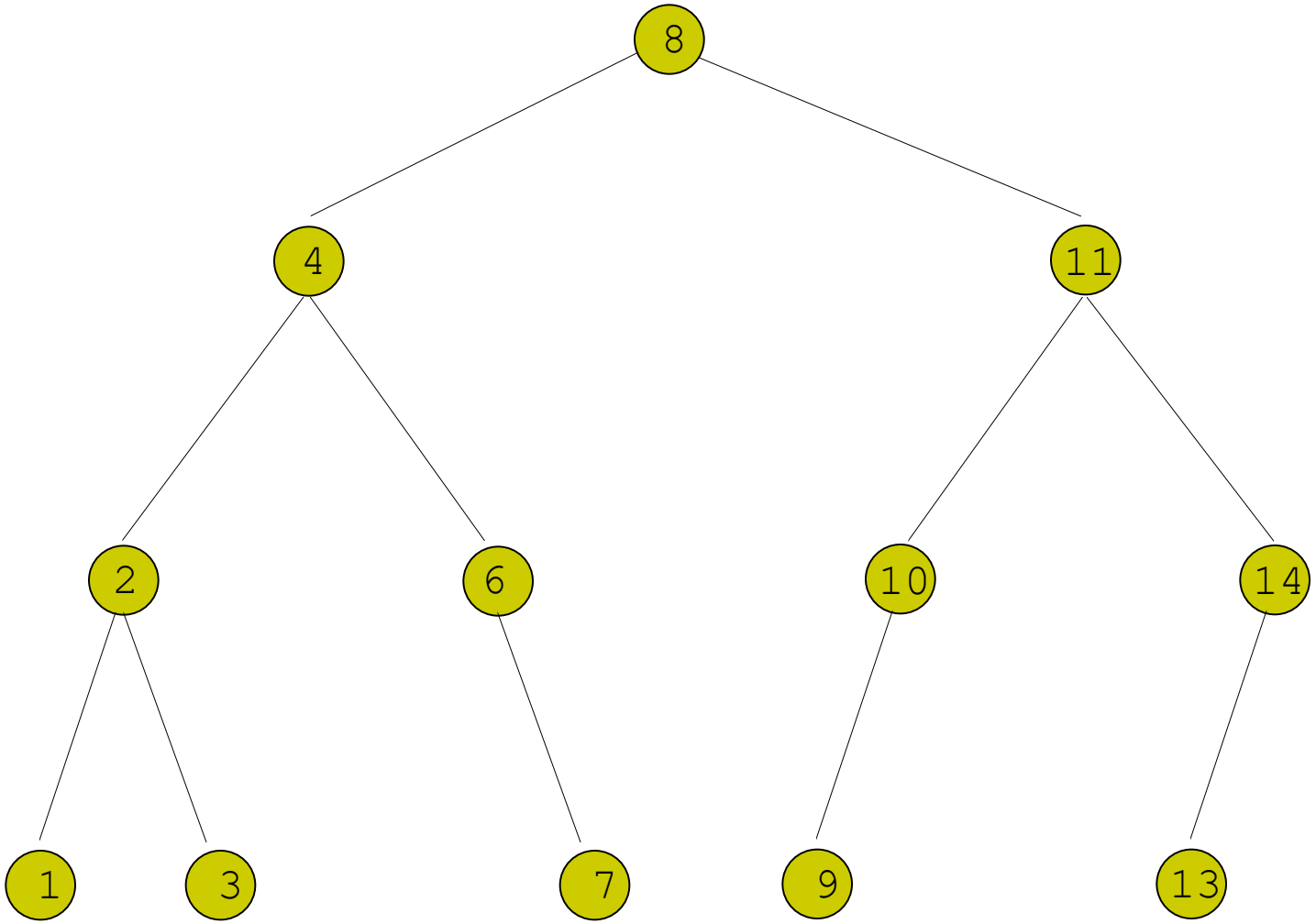


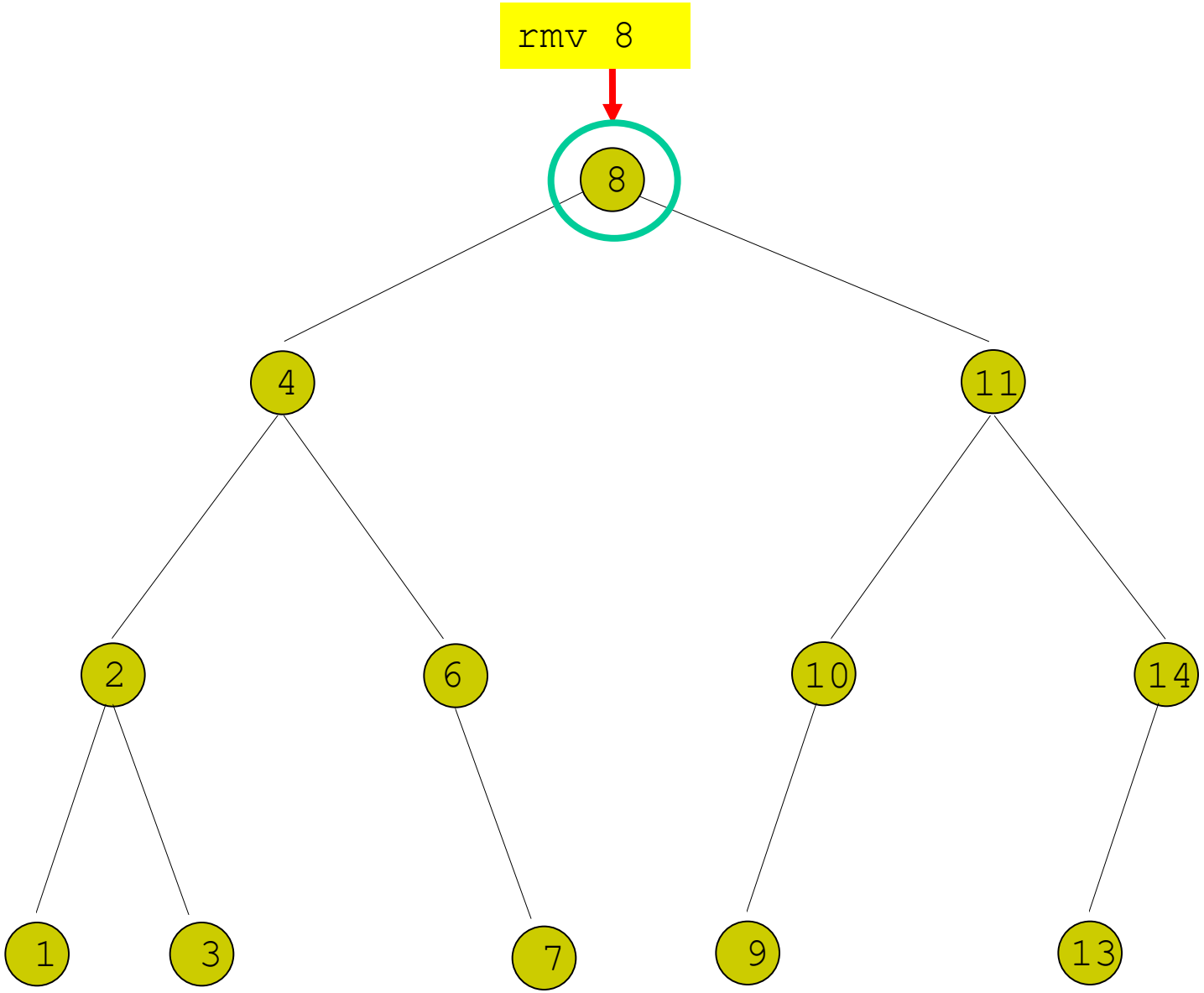
rmv 12

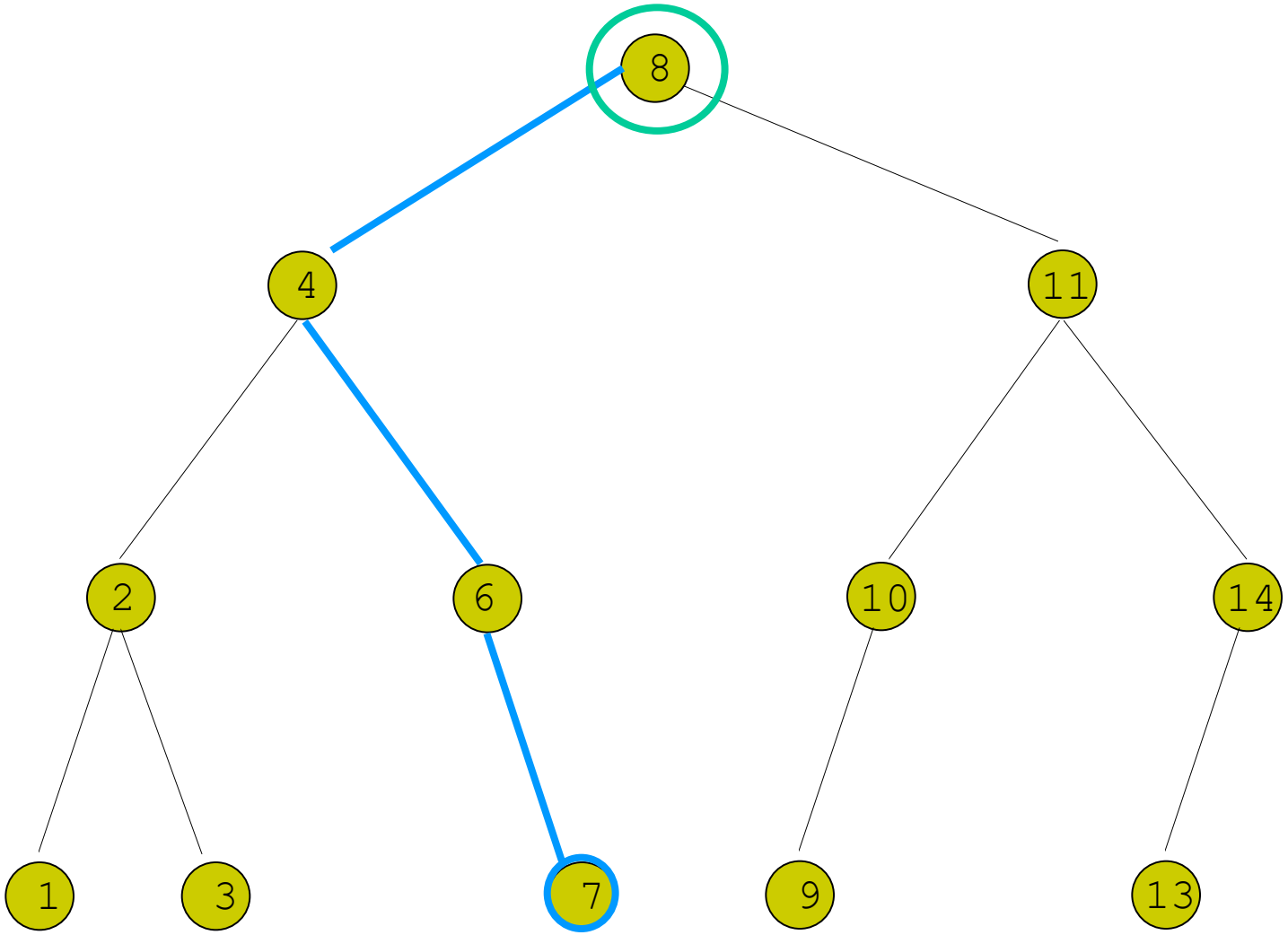


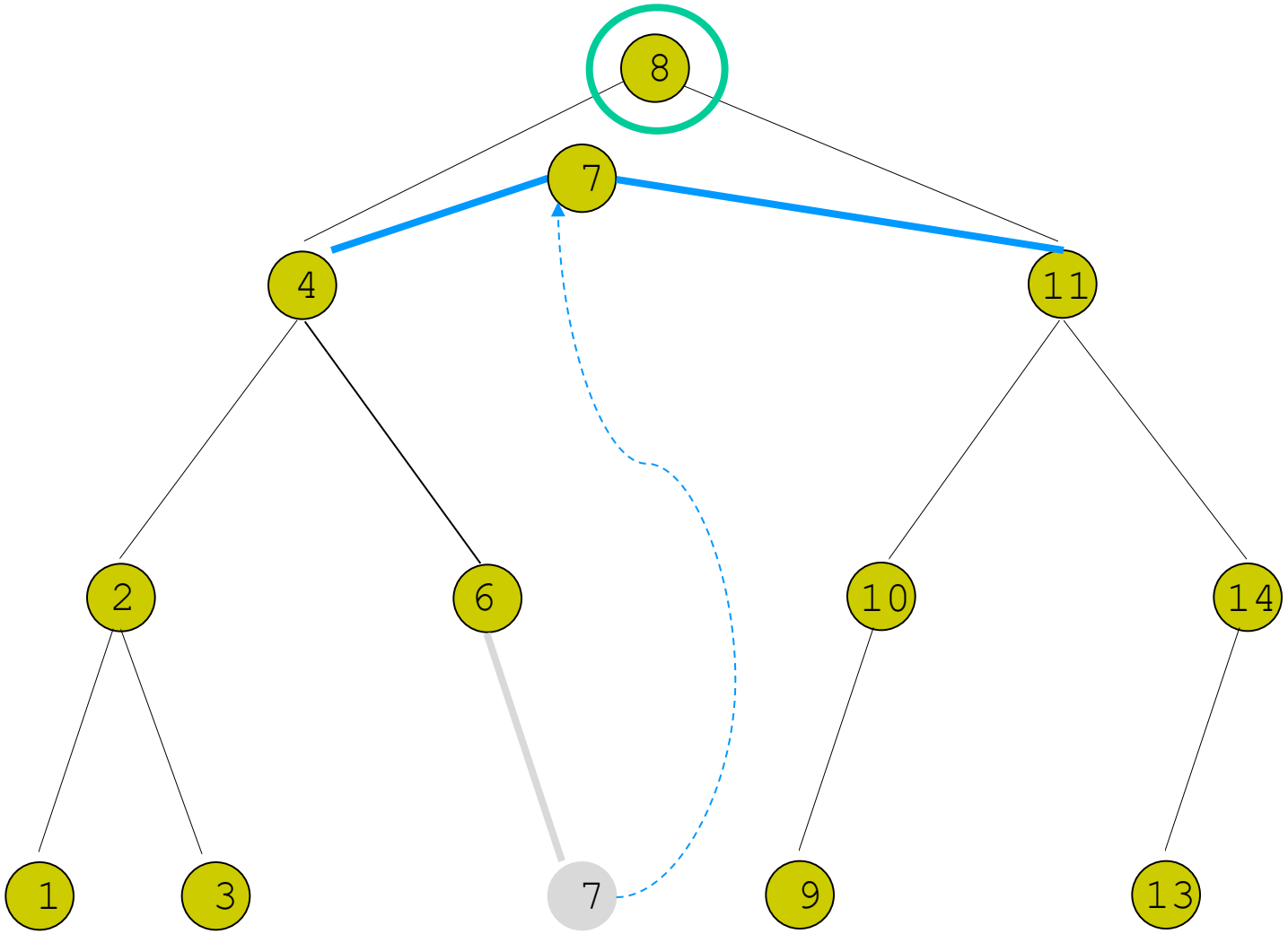
rmv 12

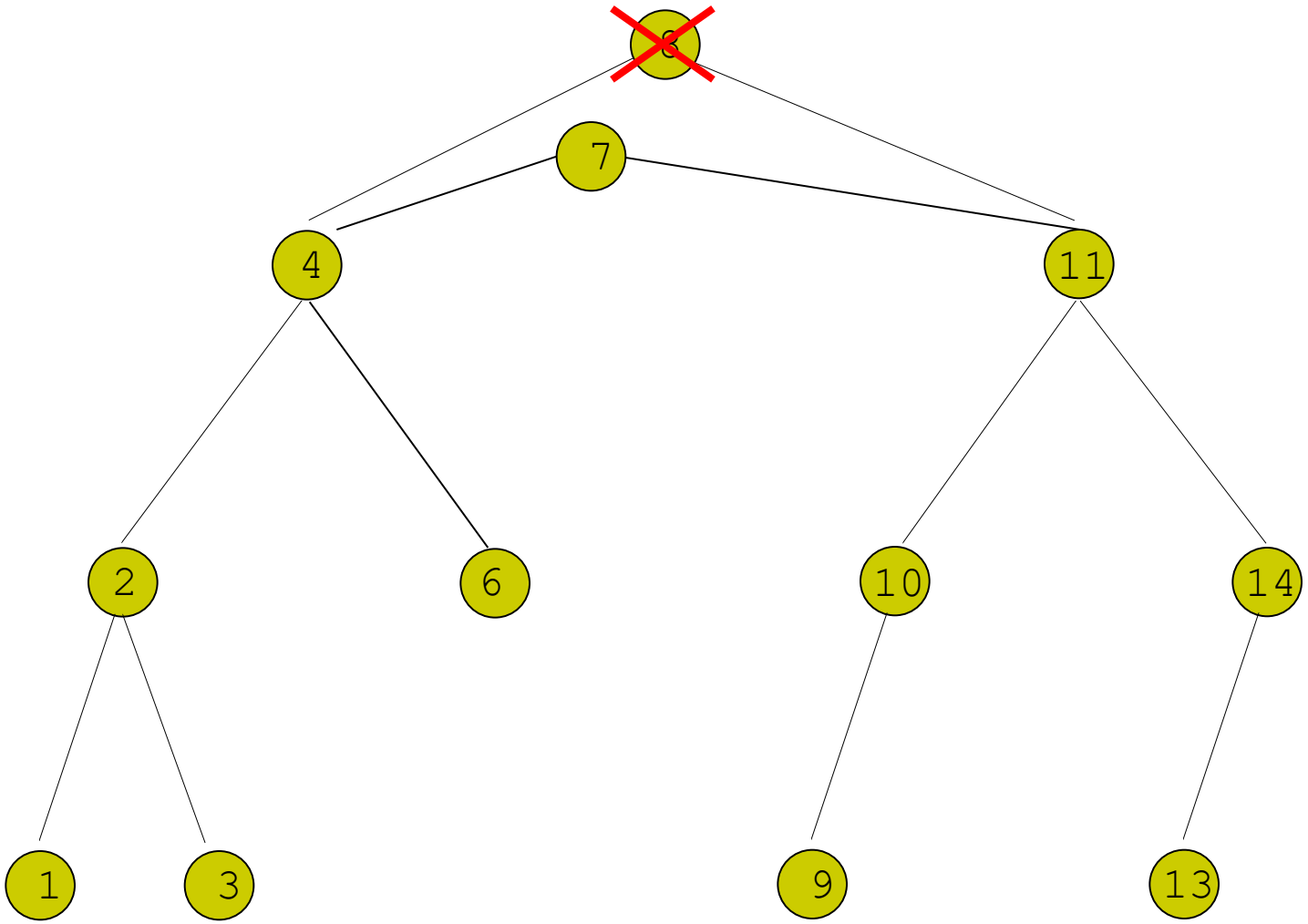


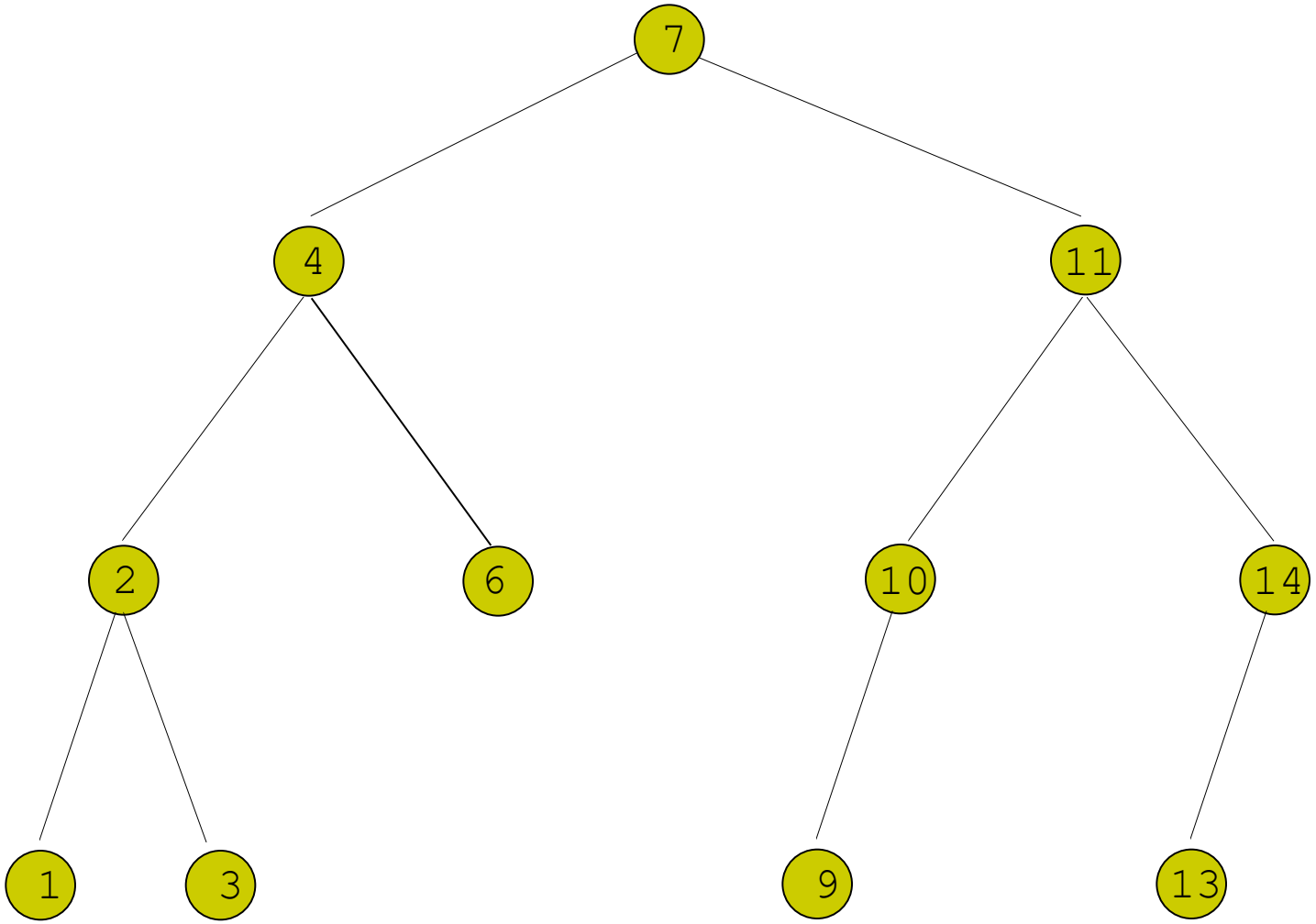

8
4
11
2
6
10
14
1
3
7
9
13


rmv 8
8
4
11
2
6
10
14
1
3
7
9
13


8
4
11
2
6
10
14
1
3
7
9
13


8
7
4
11
2
6
10
14
1
3
7
9
13


8
7
4
11
2
6
10
14
1
3
9
13


7
4
11
2
6
10
14
1
3
9
13

Για όσους έχουν κέφια

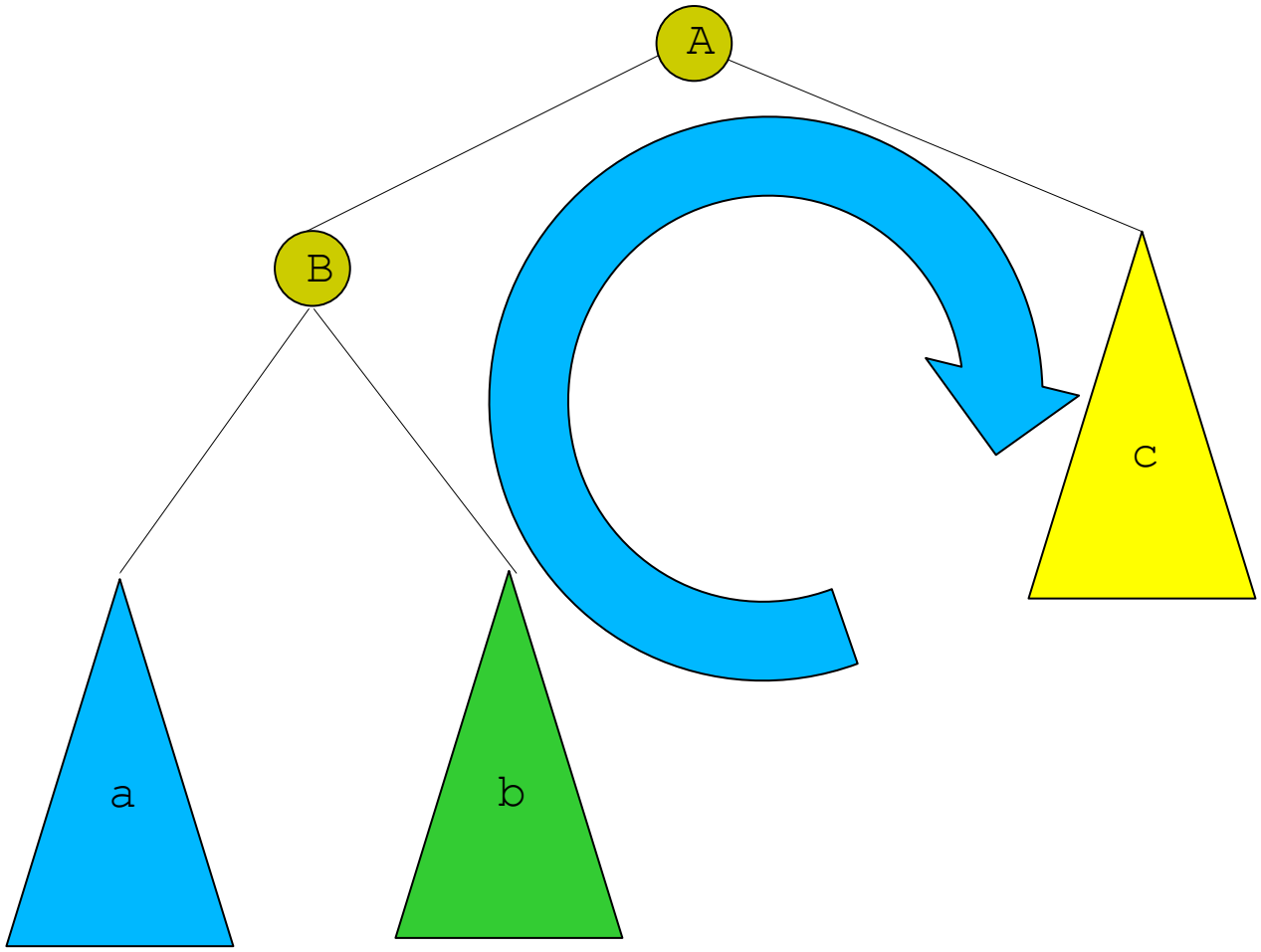
- Προσπαθήστε να βρείτε **μόνοι σας** τα βήματα της απομάκρυνσης κόμβου στην γενική περίπτωση
  - ο κόμβος **δεν** είναι φύλο, άρα σίγουρα έχει τουλάχιστον ένα υποδέντρο (αριστερό ή δεξί ή και τα δύο)
- Γράψτε κώδικα για την **αναζήτηση** στο **αριστερό** (και, ανάποδα, στο δεξί) υποδέντρο του κόμβου προς αφαίρεση, έτσι ώστε να βρεθεί ο **μεγαλύτερος** (και, ανάποδα, ο μικρότερος) κόμβος «αντικαταστάτης»
- Σκεφτείτε τις **περιπτώσεις** που υπάρχουν για αυτά τα υποδέντρα και τις αντίστοιχες **ενέργειες** που απαιτούνται για να γίνει σωστά η αντικατάσταση

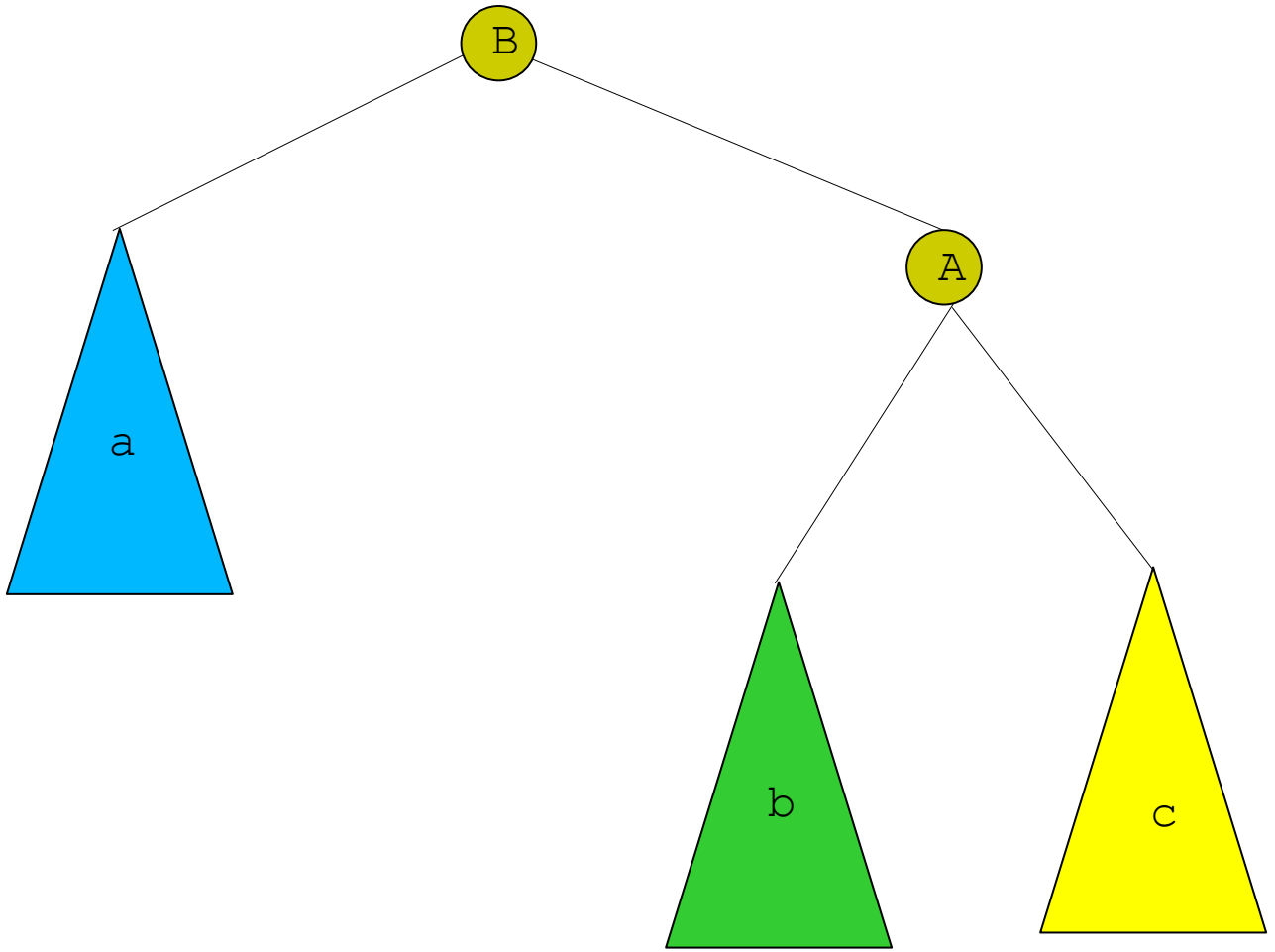
Εξισορρόπηση

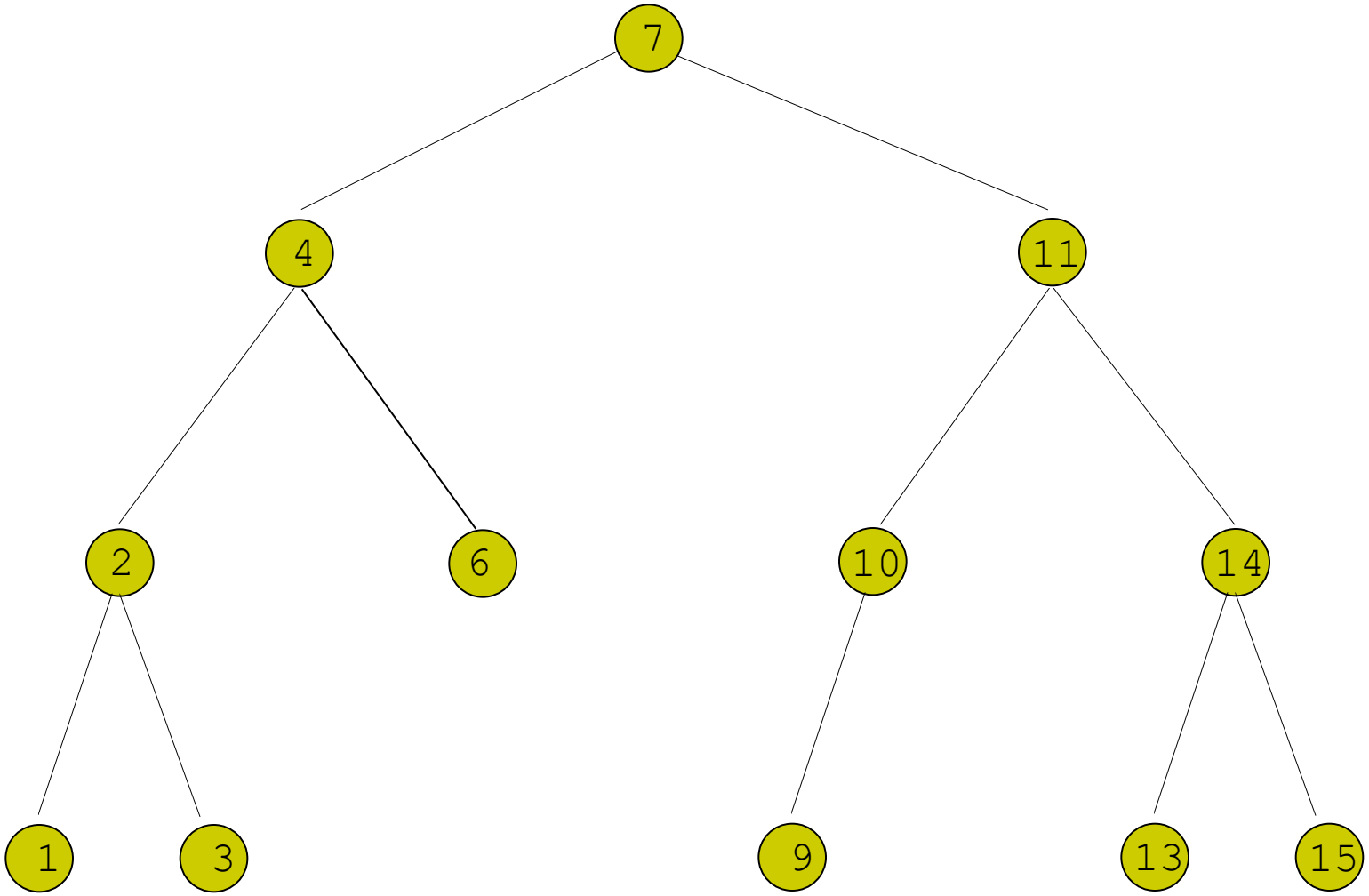
- Το δέντρο μπορεί να χάσει την «ισορροπία» του
  - λόγω της (τυχαίας) σειράς με την οποίας προστίθενται στοιχεία
  - λόγω της (τυχαίας) σειράς με την οποία αφαιρούνται στοιχεία

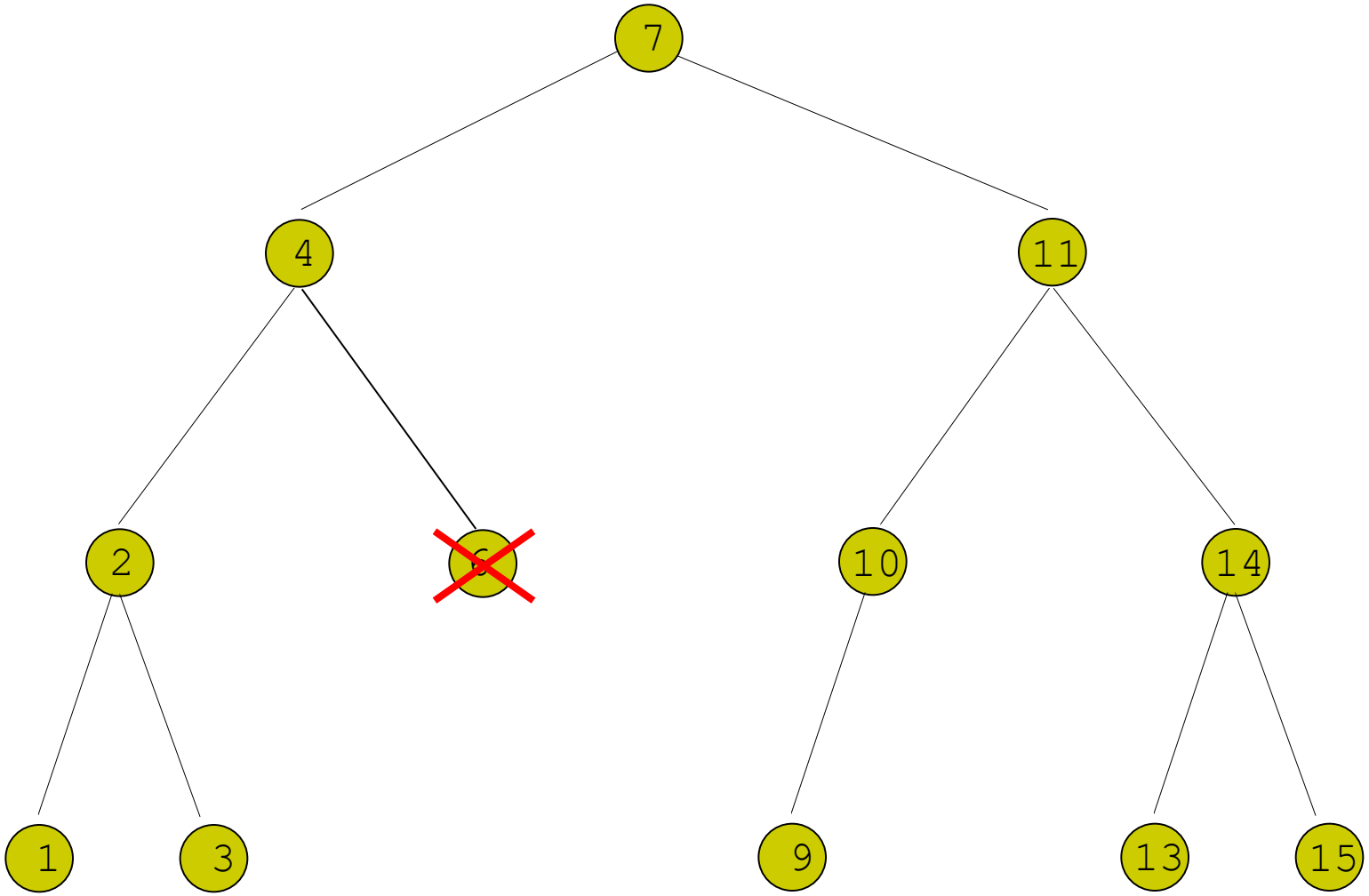
- **Ακραία περίπτωση:** το δέντρο **«εκφυλίζεται»** σε μια (ταξινομημένη) λίστα
  - πολυπλοκότητα αναζήτησης γίνεται $O(\mathrm{N})$ αντί για $O(\log_2 \mathrm{N})$

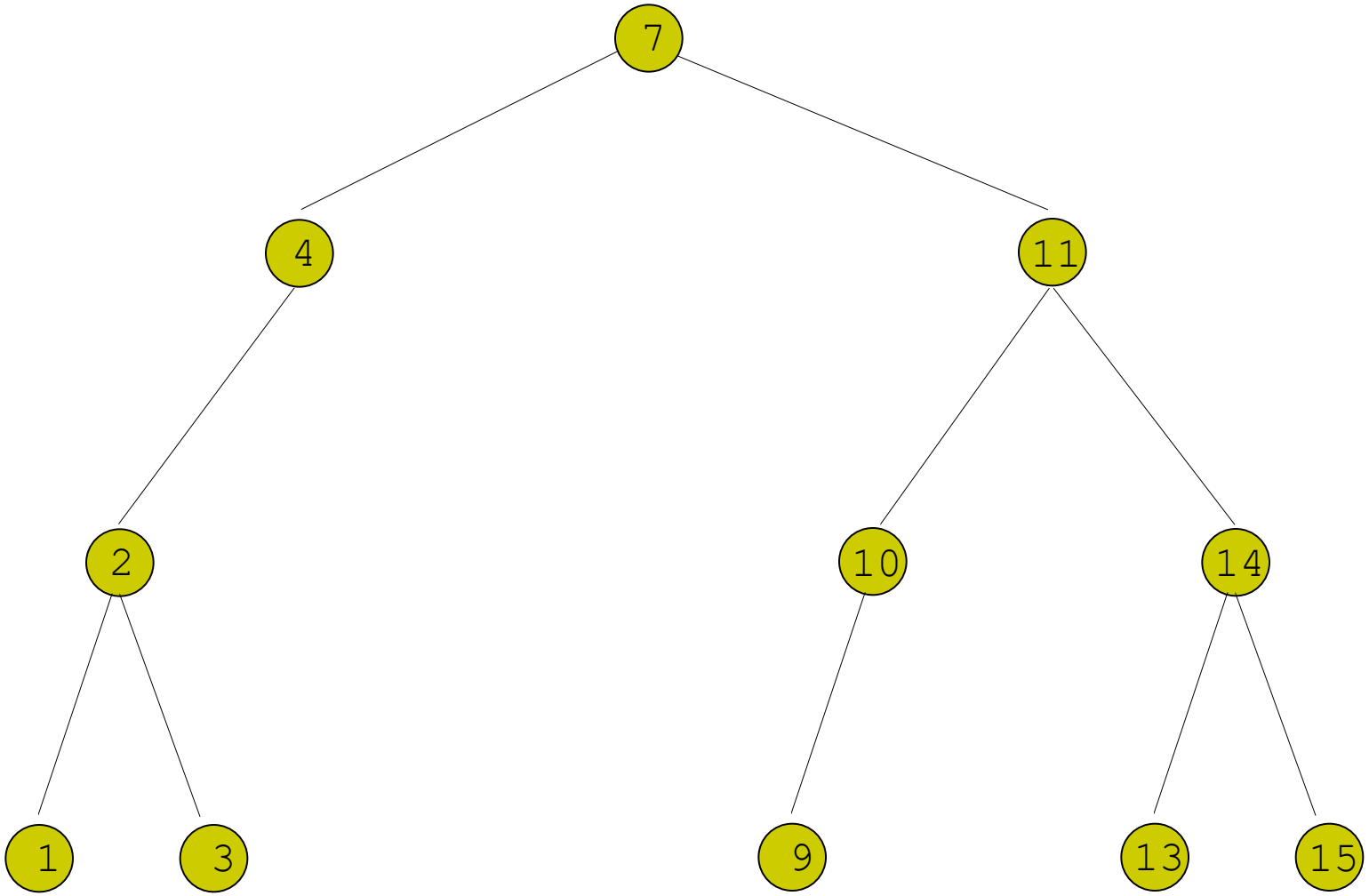
- **Λύση:** πρέπει να γίνονται ενέργειες **εξισορρόπησης** του δέντρου (balancing)
  - περιοδικά ή κάθε φορά που γίνεται εισαγωγή/απομάκρυνση

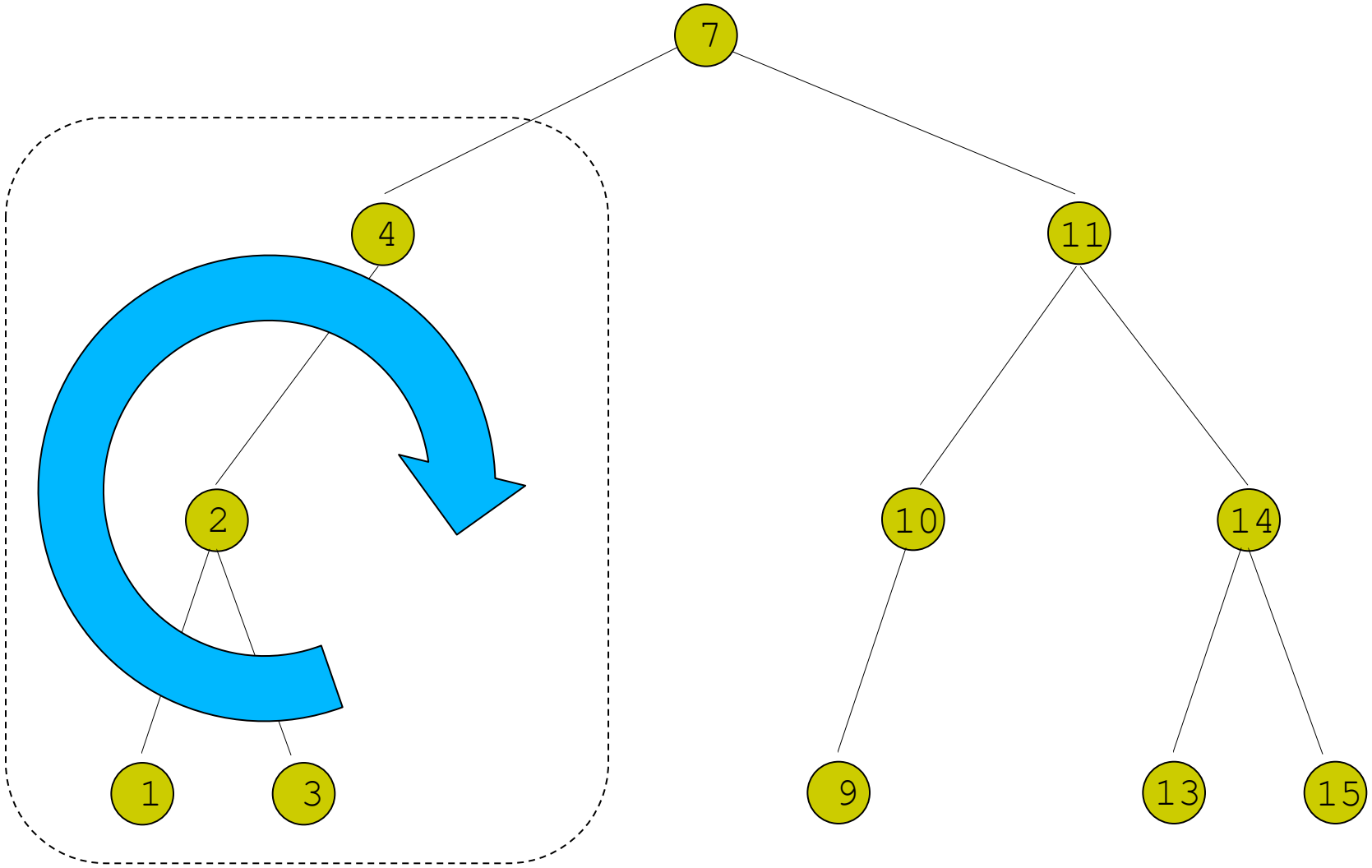

A
B
C
a
b

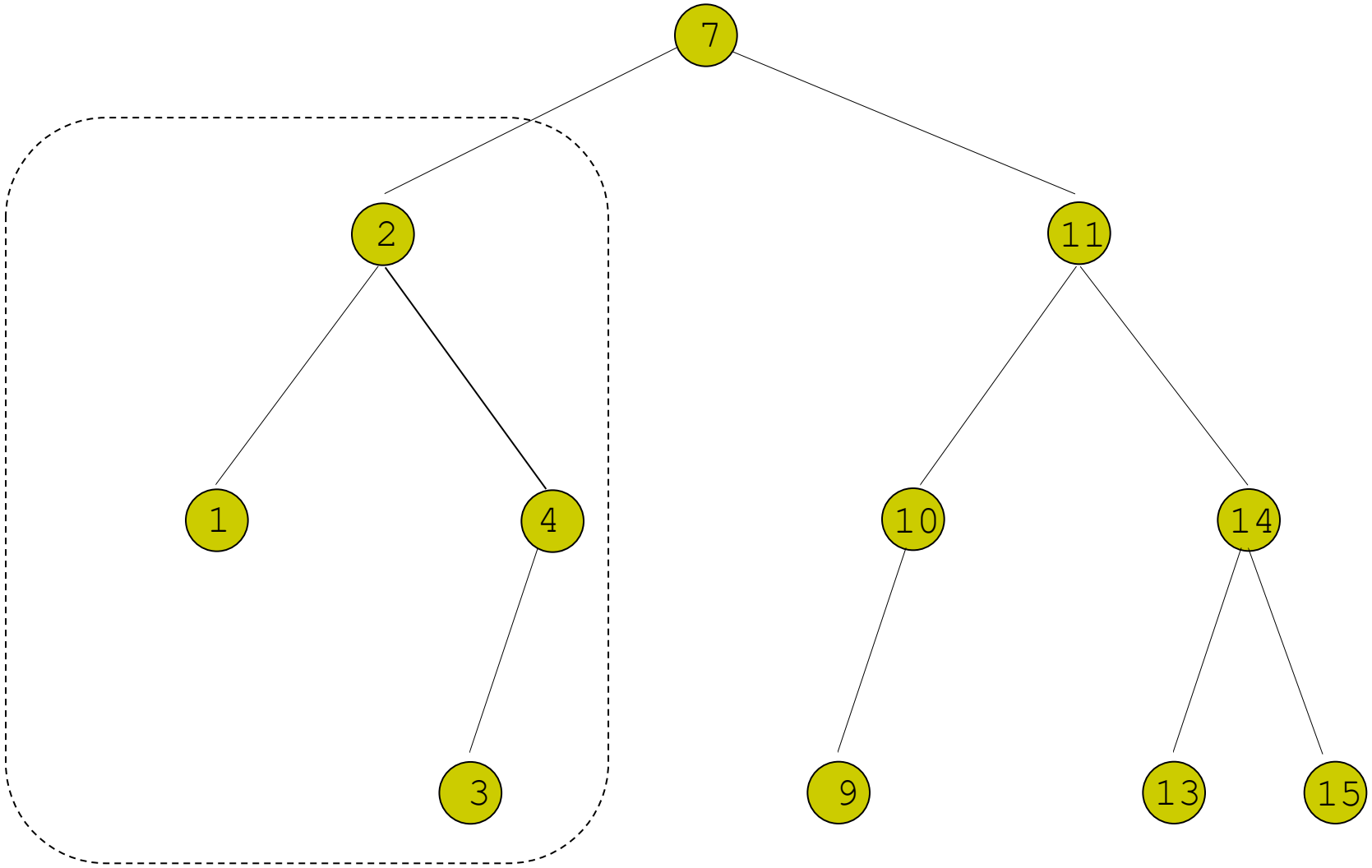

B
A
a
b
c

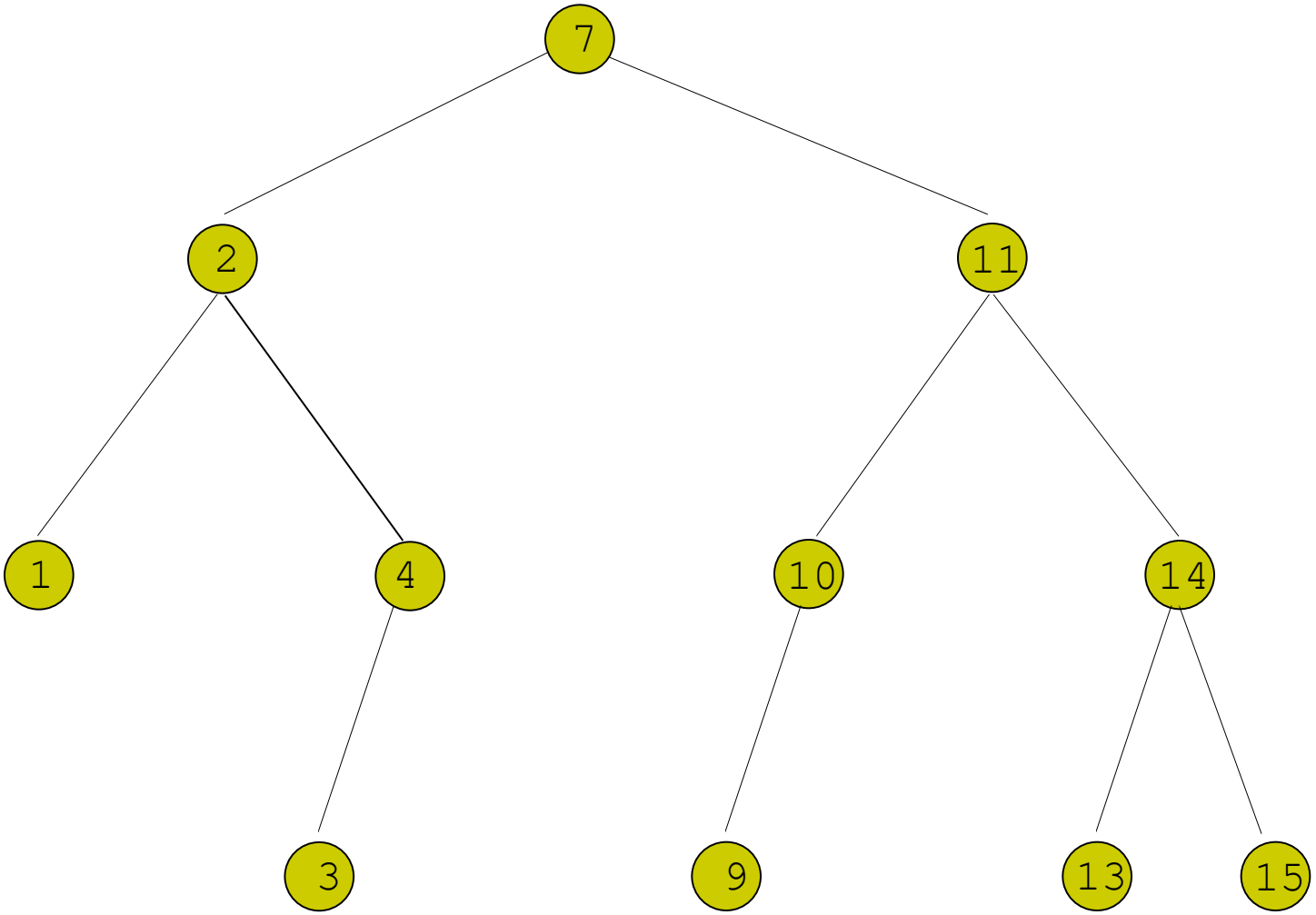

7
4
11
2
6
10
14
1
3
9
13
15

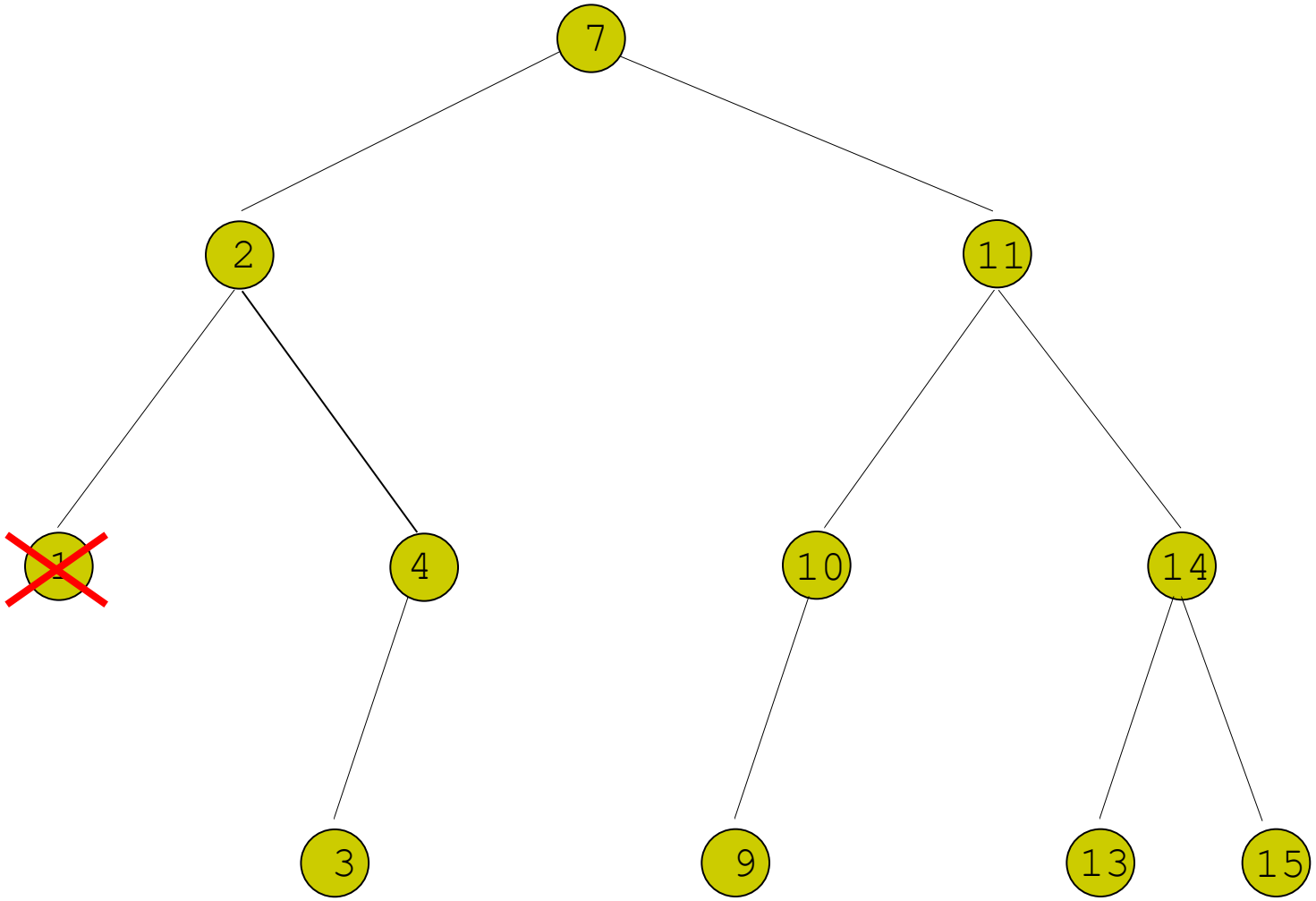

7
4
11
2
6
10
14
1
3
9
13
15

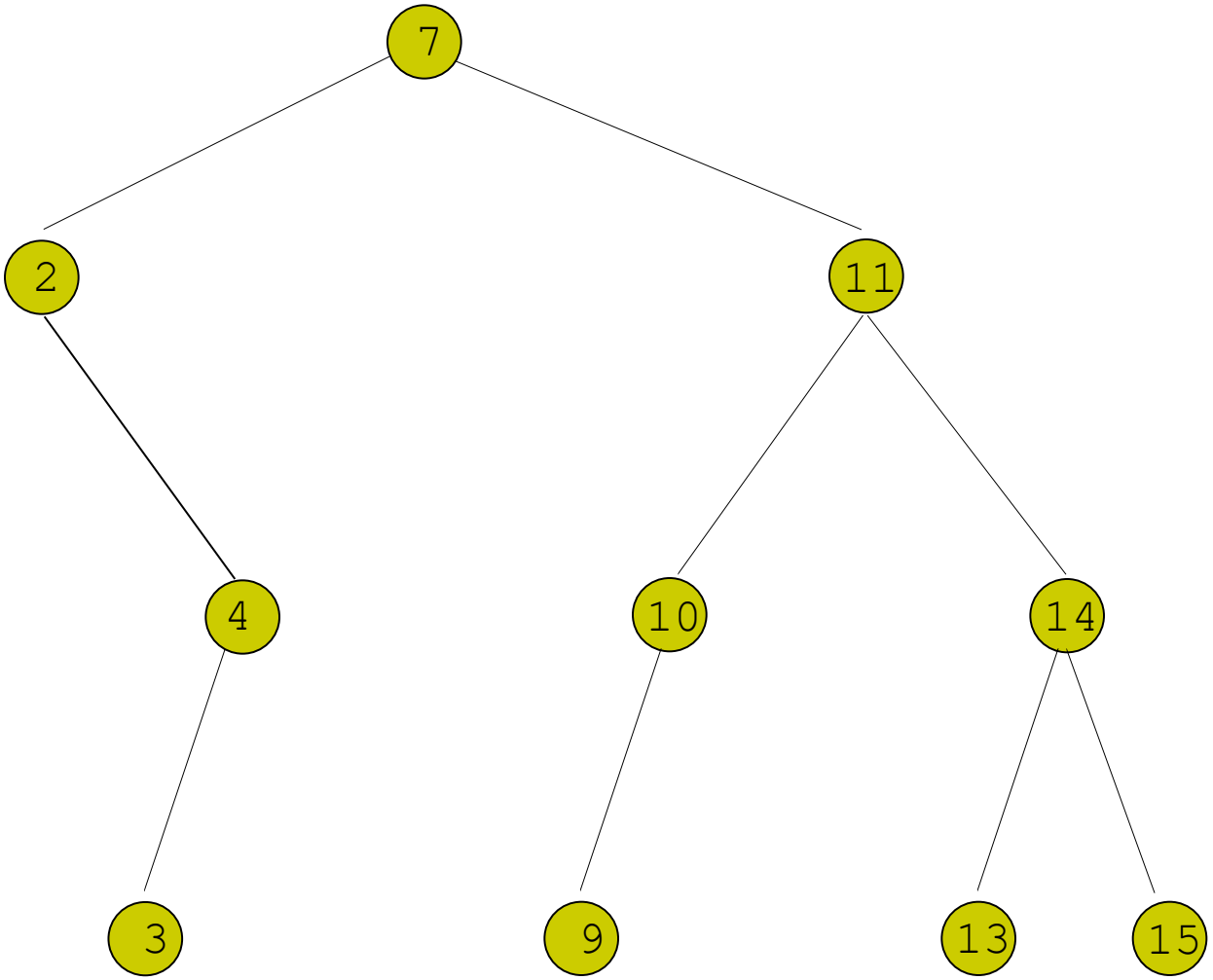

7
4
11
2
10
14
1
3
9
13
15

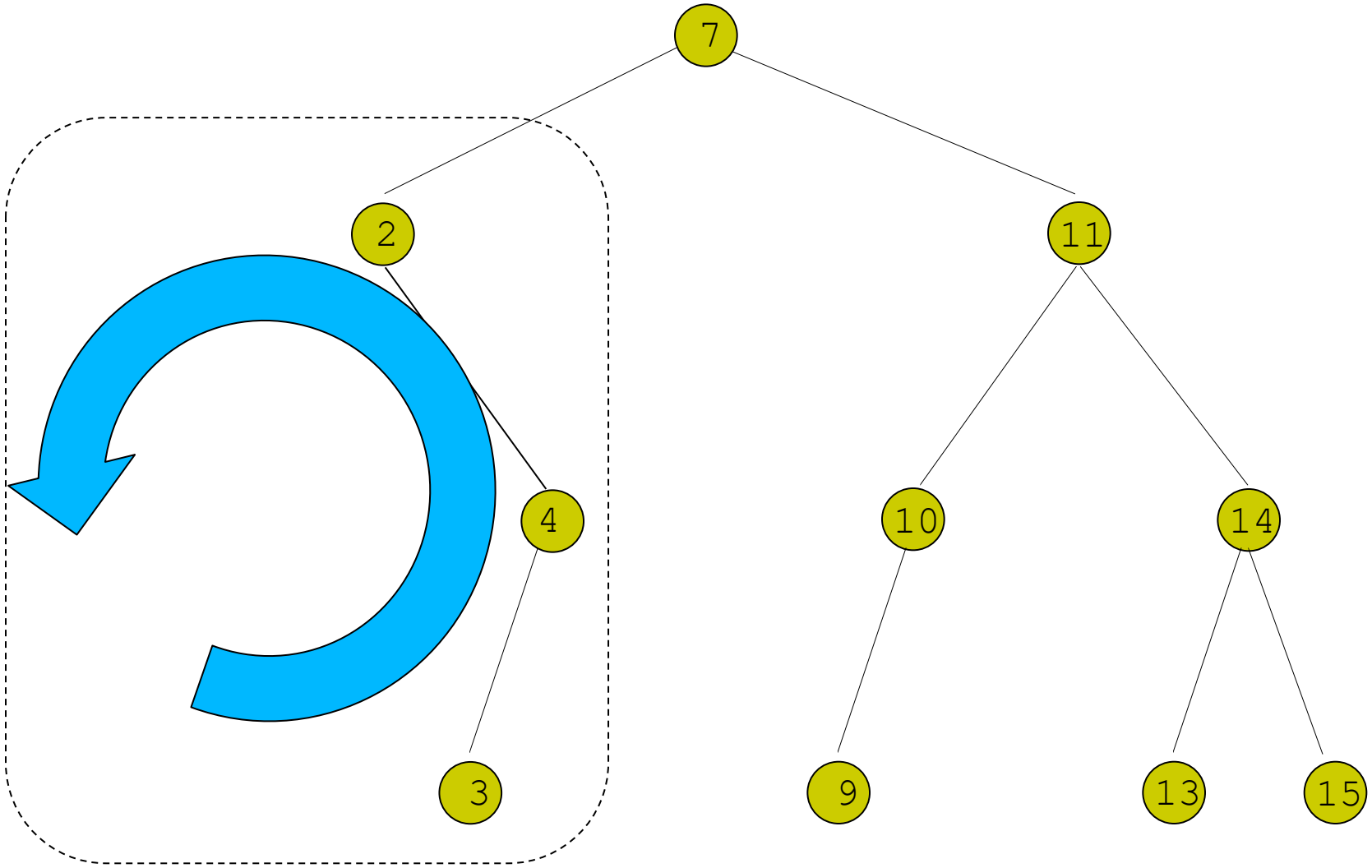

7
4
11
2
10
14
1
3
9
13
15

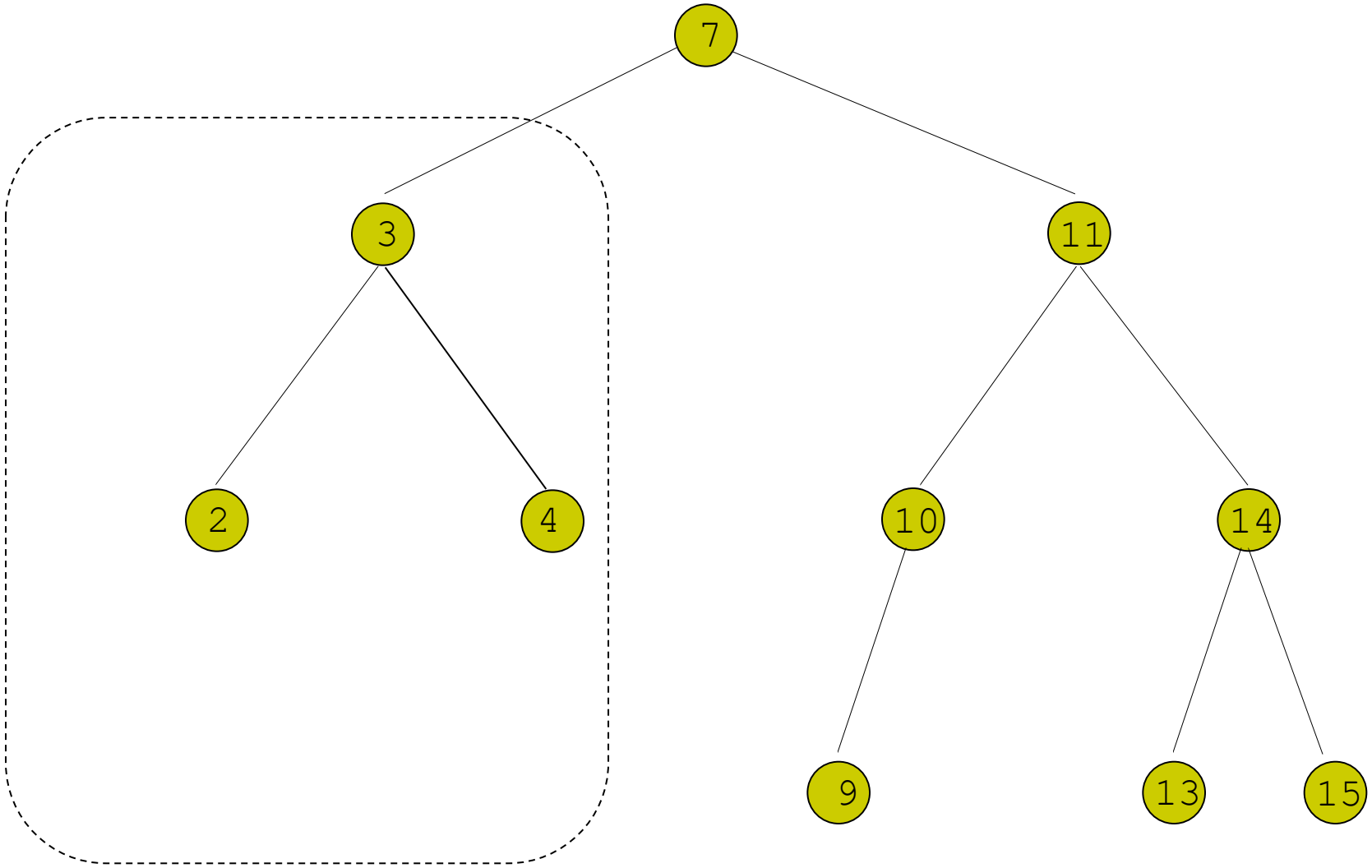

7
2
11
1
4
10
14
3
9
13
15

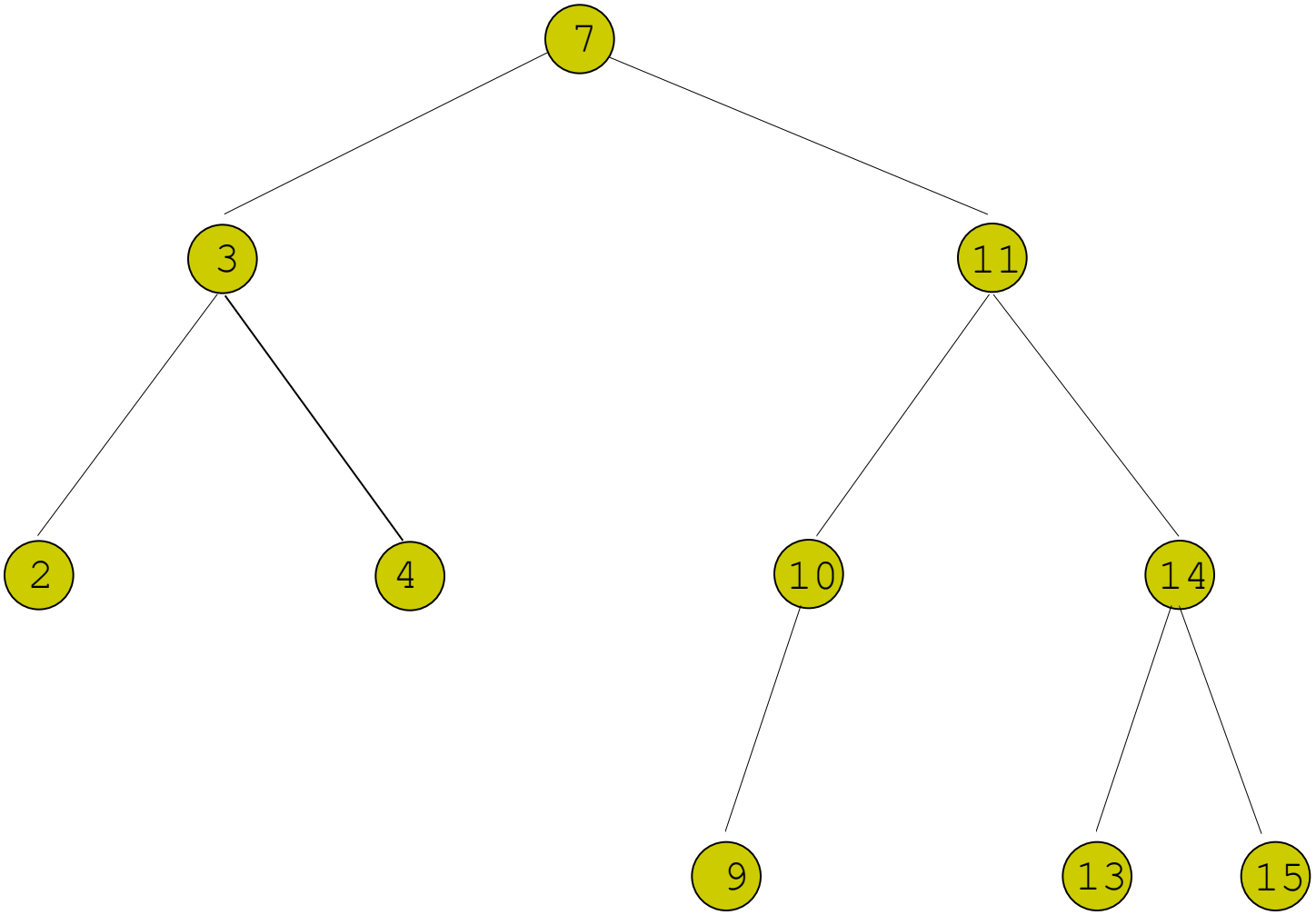

7
2
11
1
4
10
14
3
9
13
15

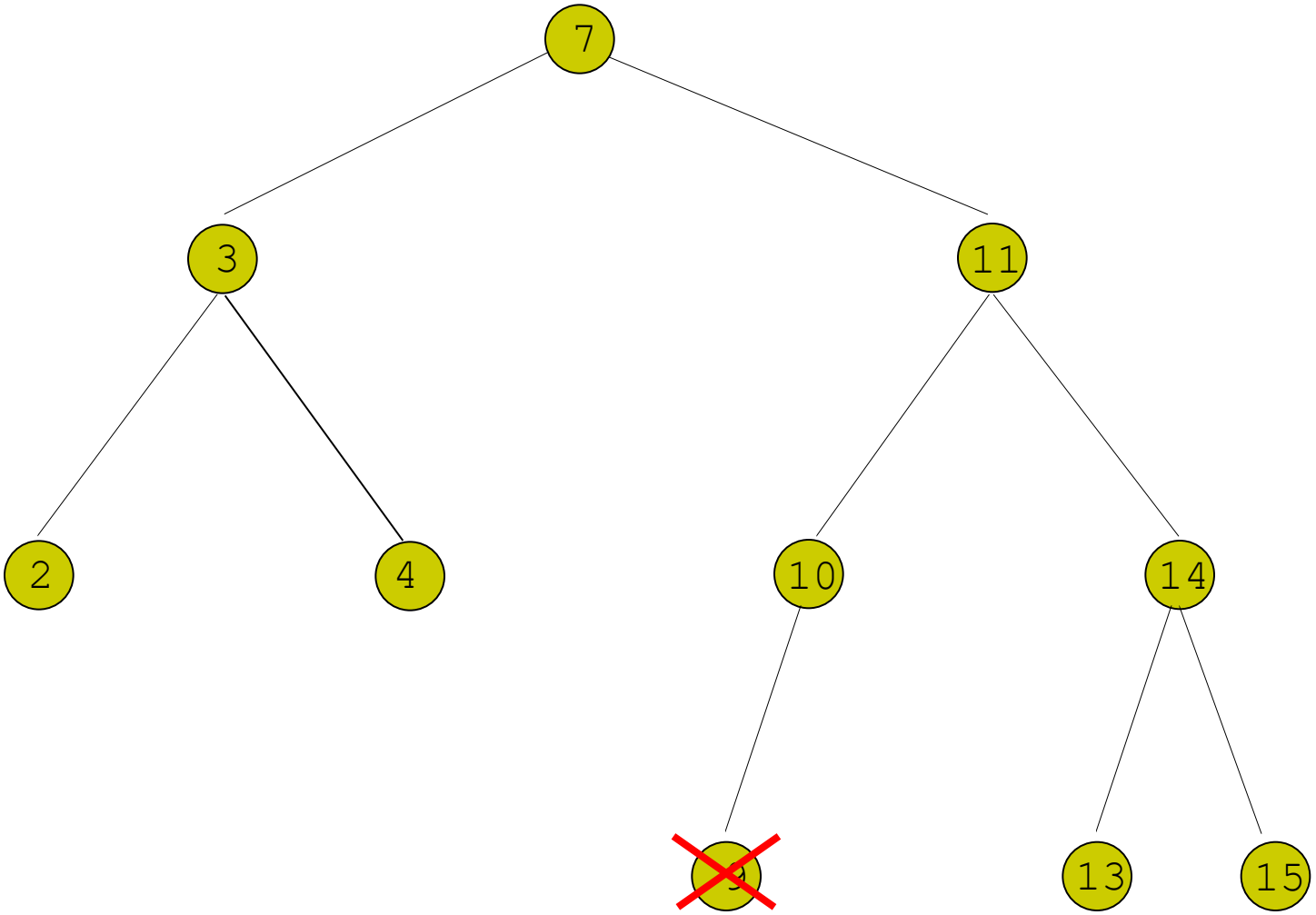

7
2
11
1
4
10
14
3
9
13
15

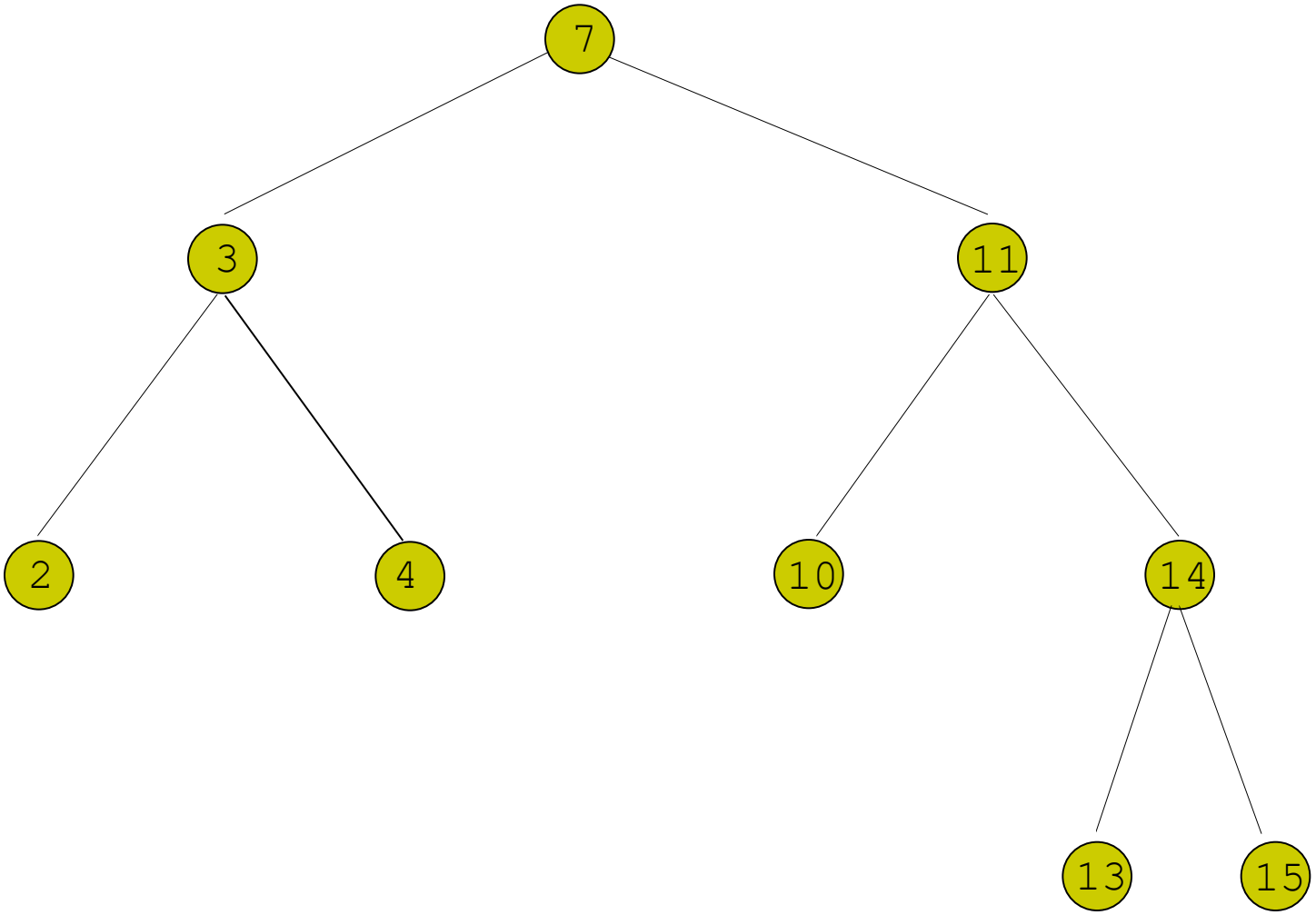

7
2
11
4
10
14
3
9
13
15

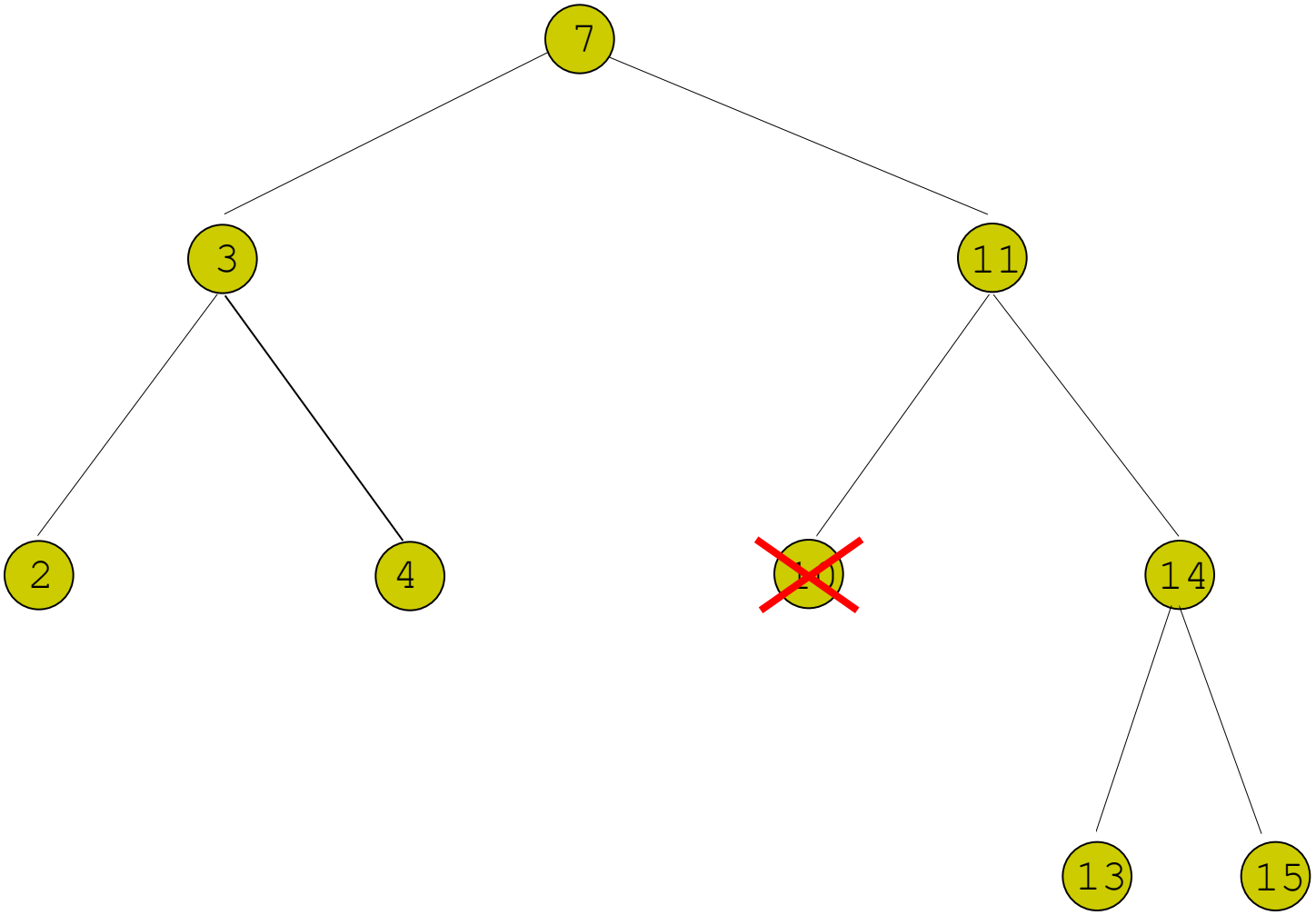

7
2
11
4
10
14
3
9
13
15

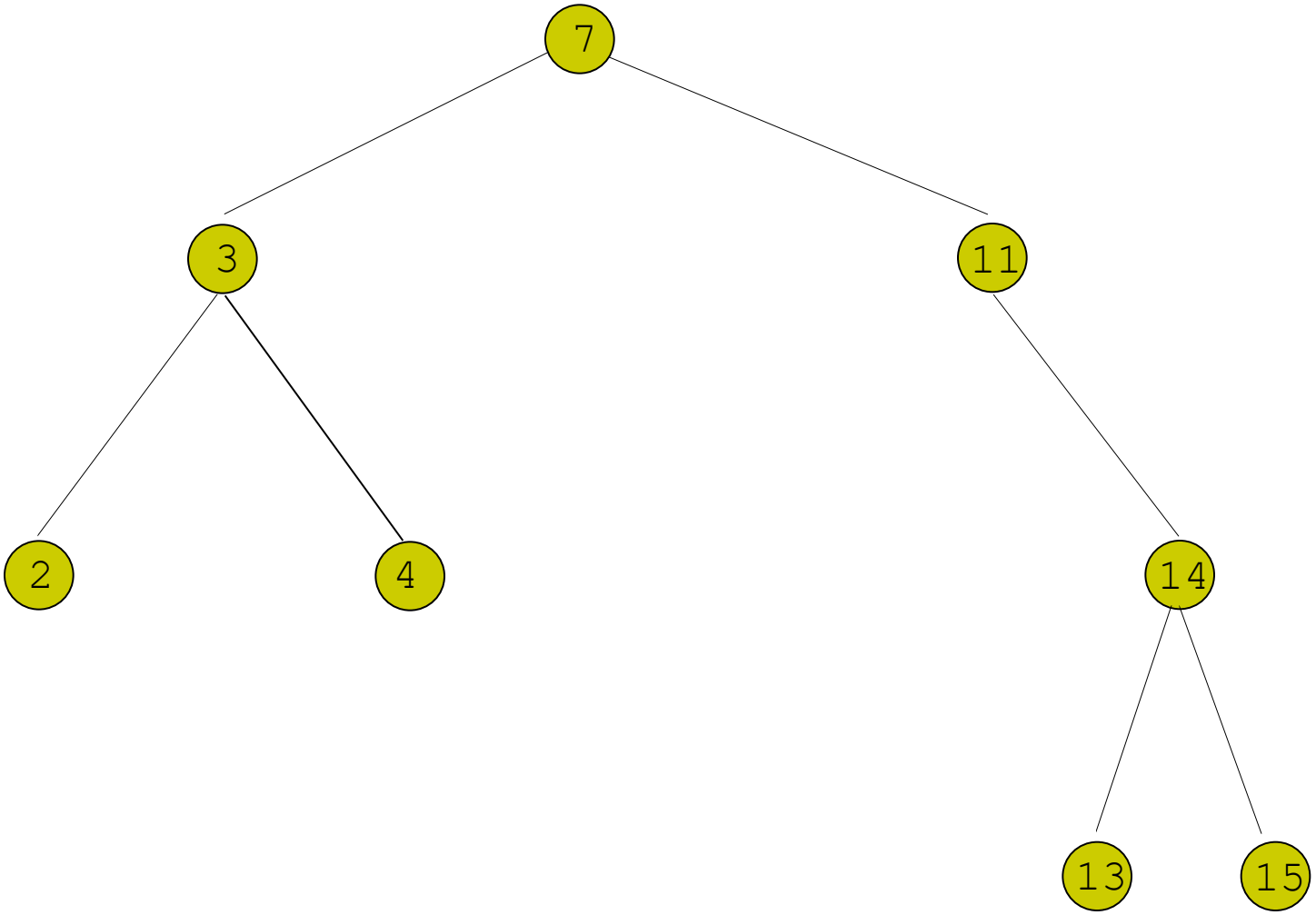

7
3
11
2
4
10
14
9
13
15

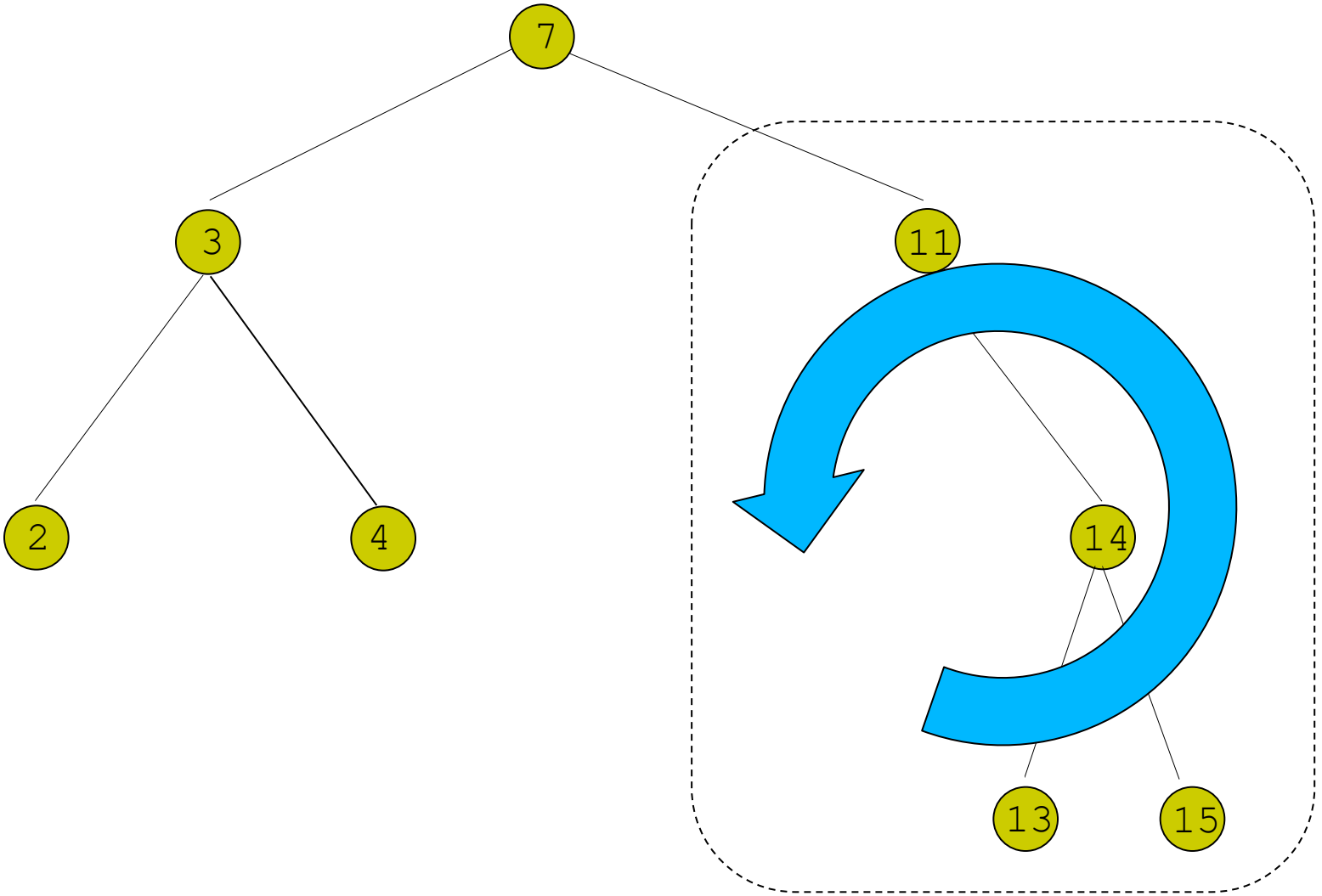

7
3
11
2
4
10
14
9
13
15

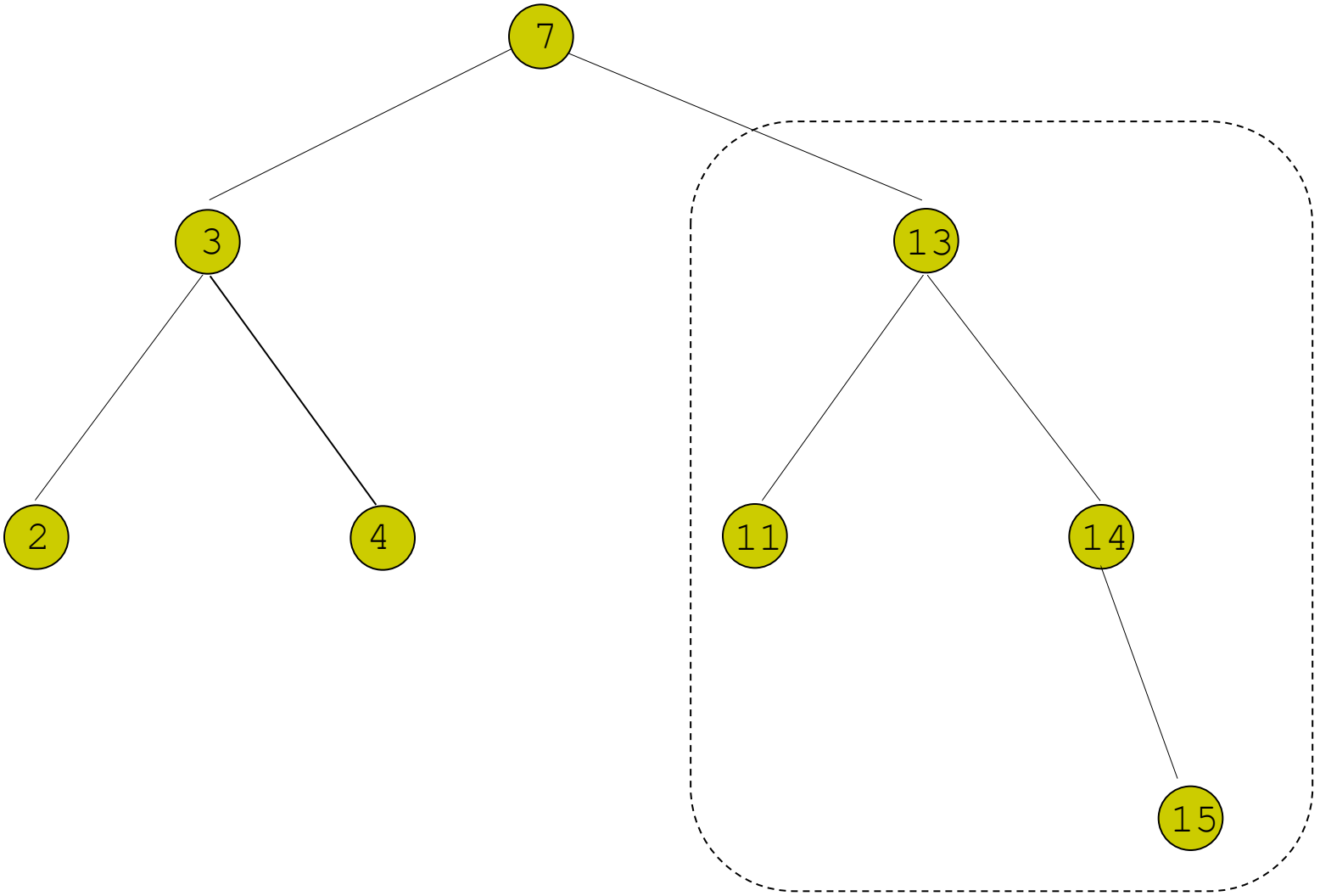

7
3
11
2
4
10
14
9
13
15

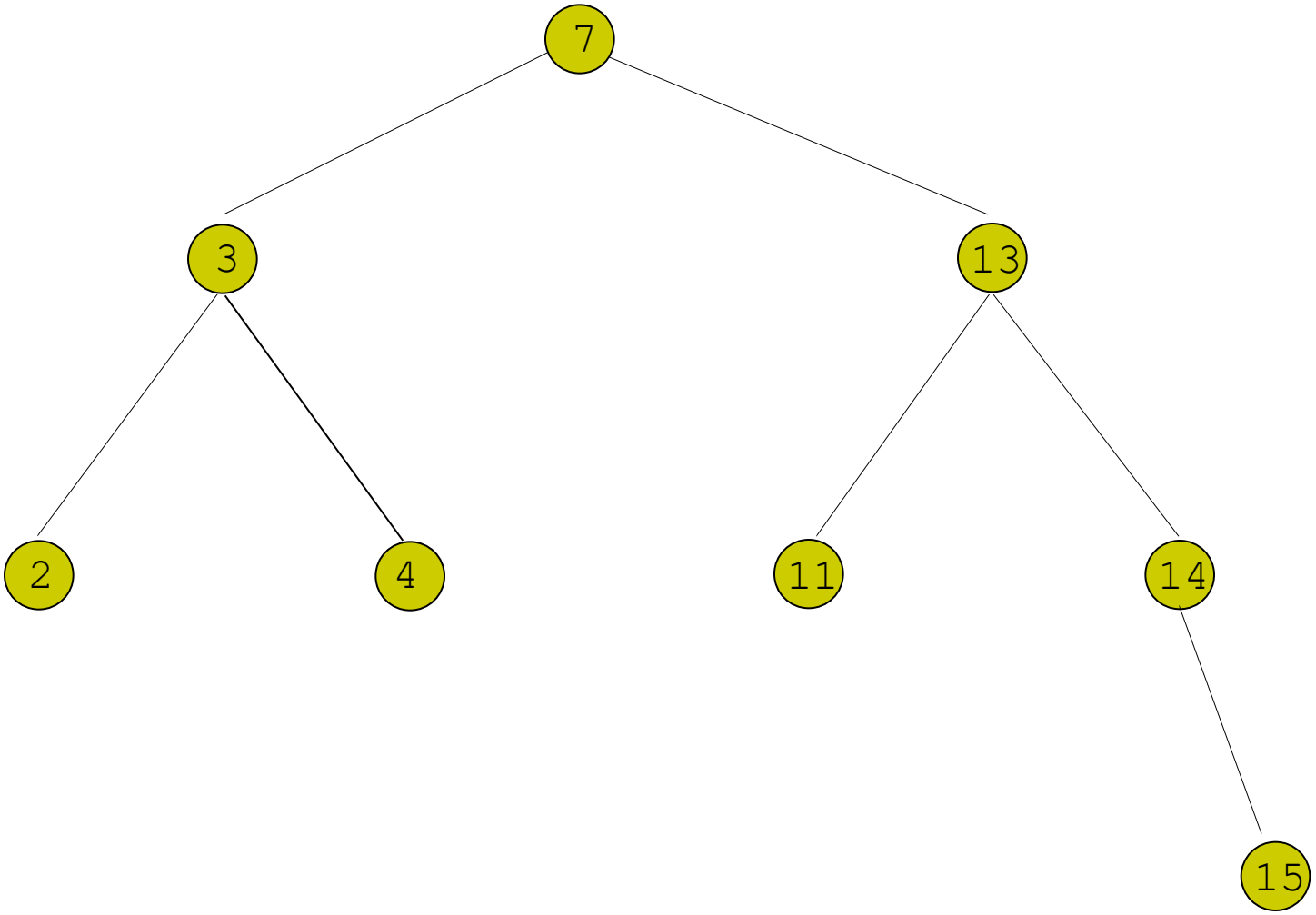

7
3
11
2
4
10
14
13
15

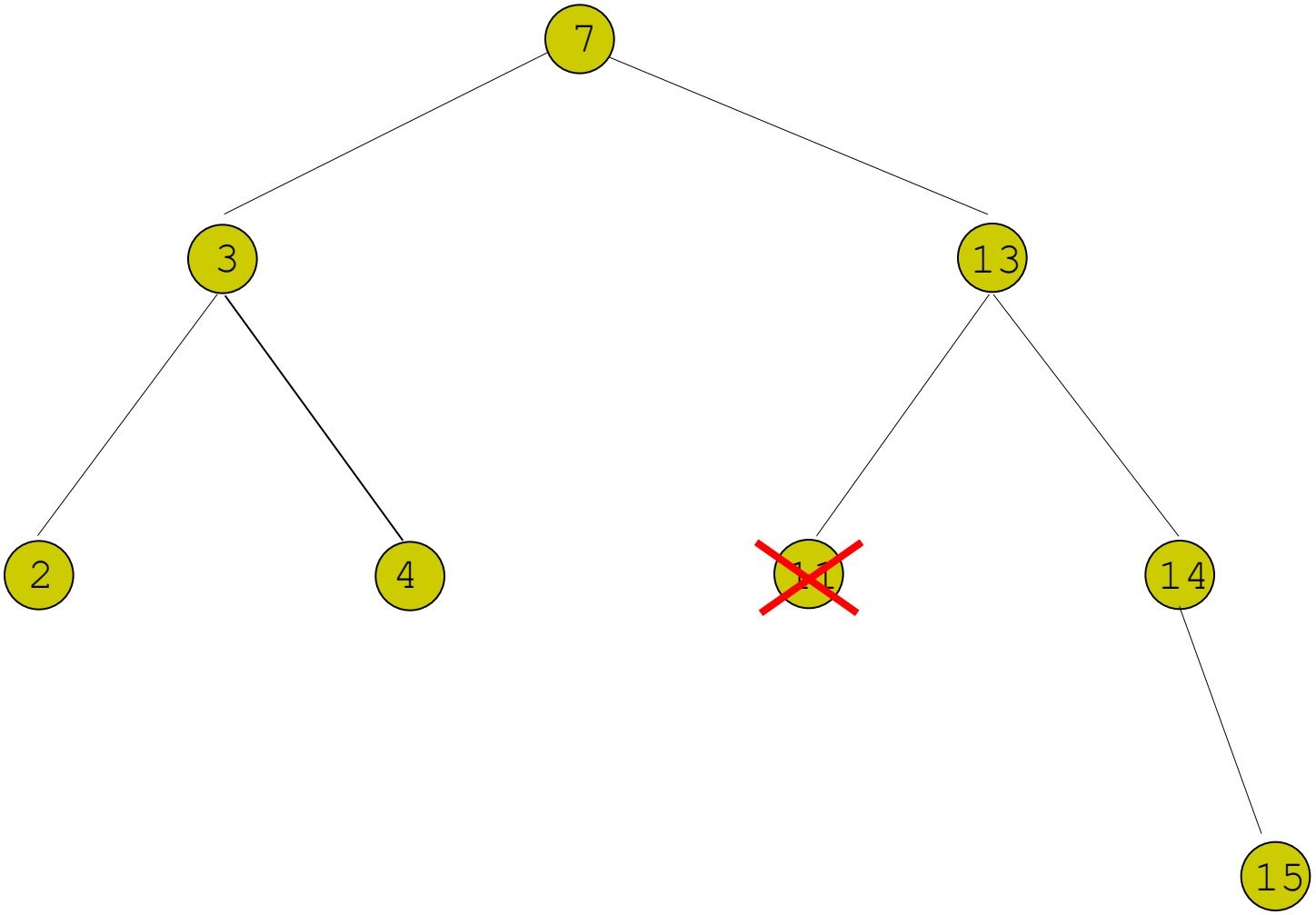

7
3
11
2
4
10
14
13
15

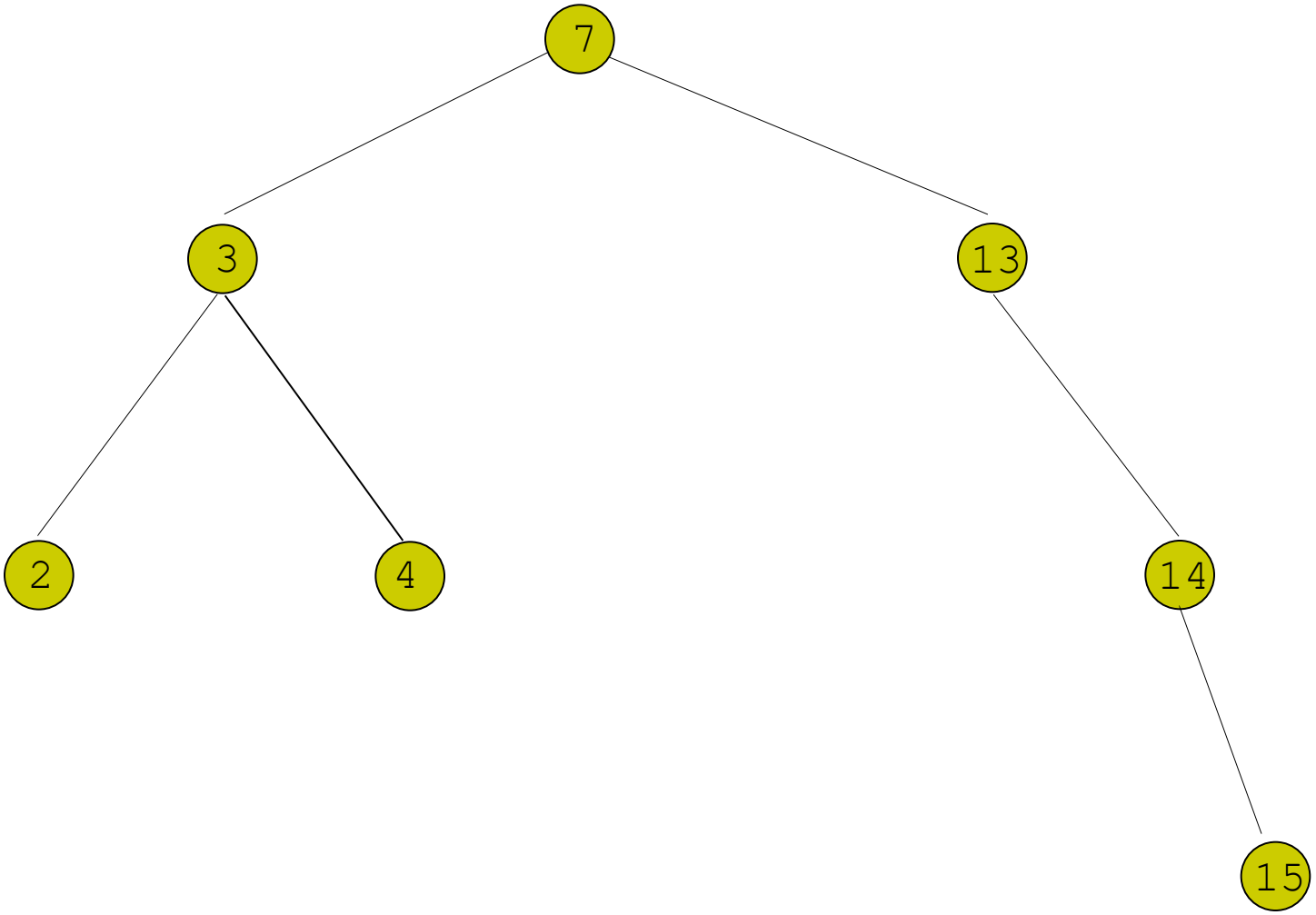

7
3
11
2
4
14
13
15

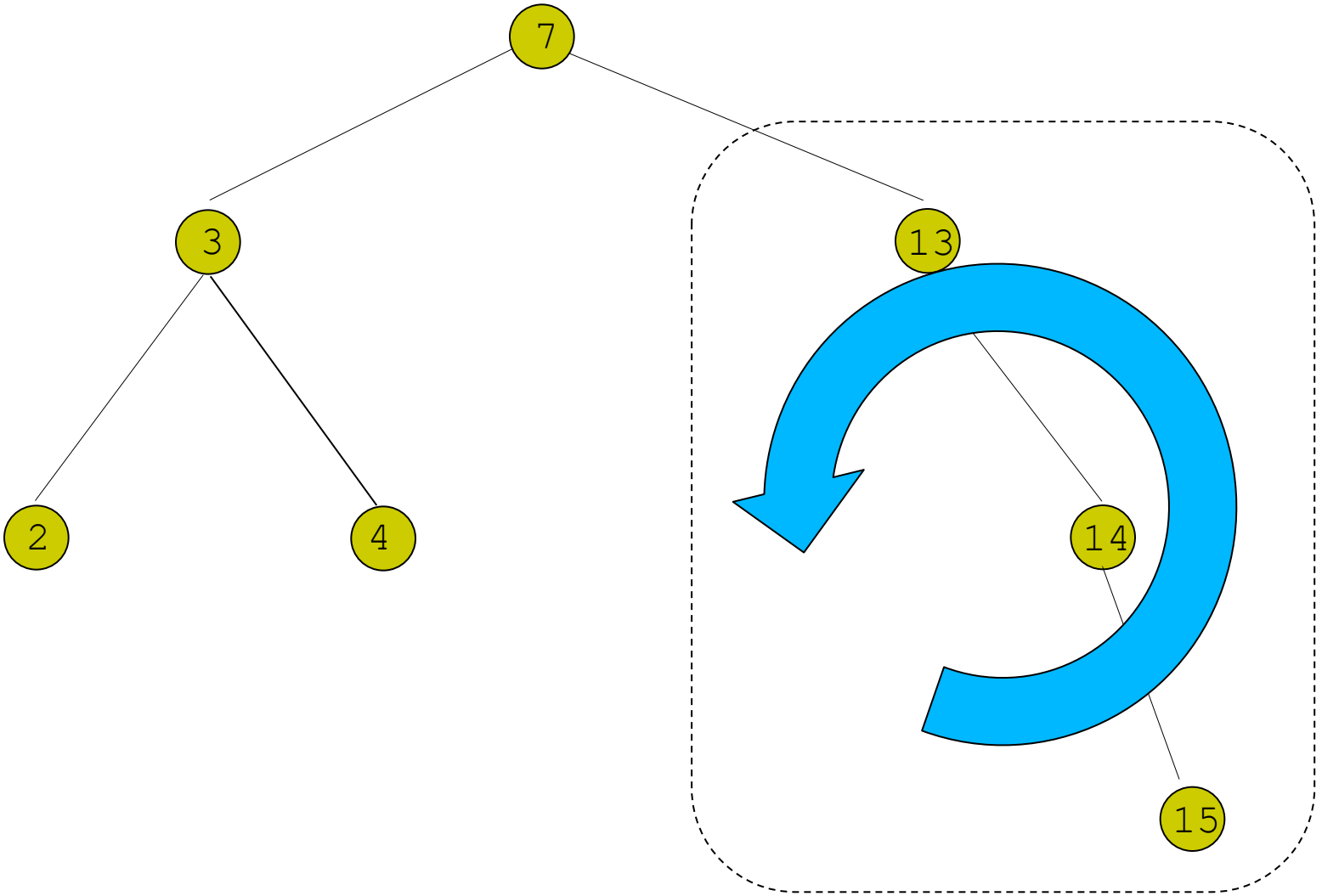

7
3
11
2
4
14
13
15

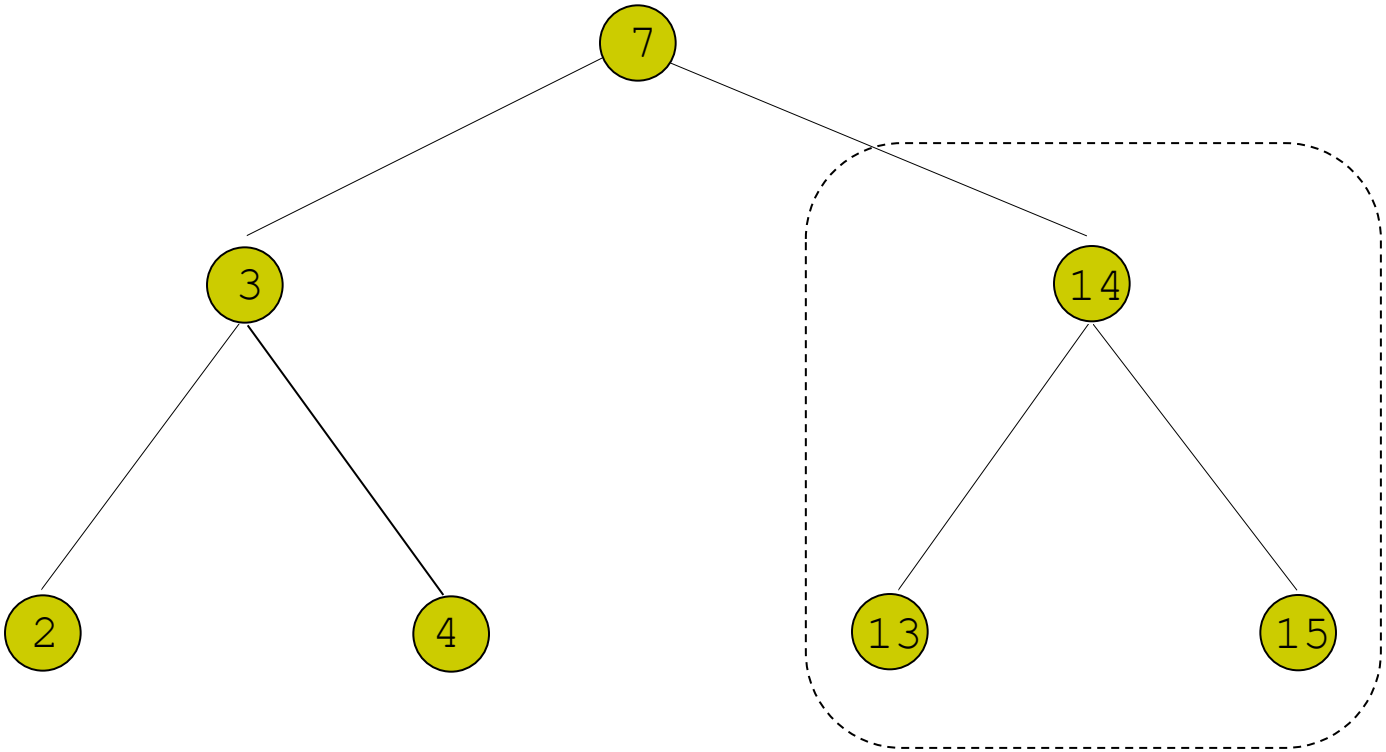

7
3
13
2
4
11
14
15

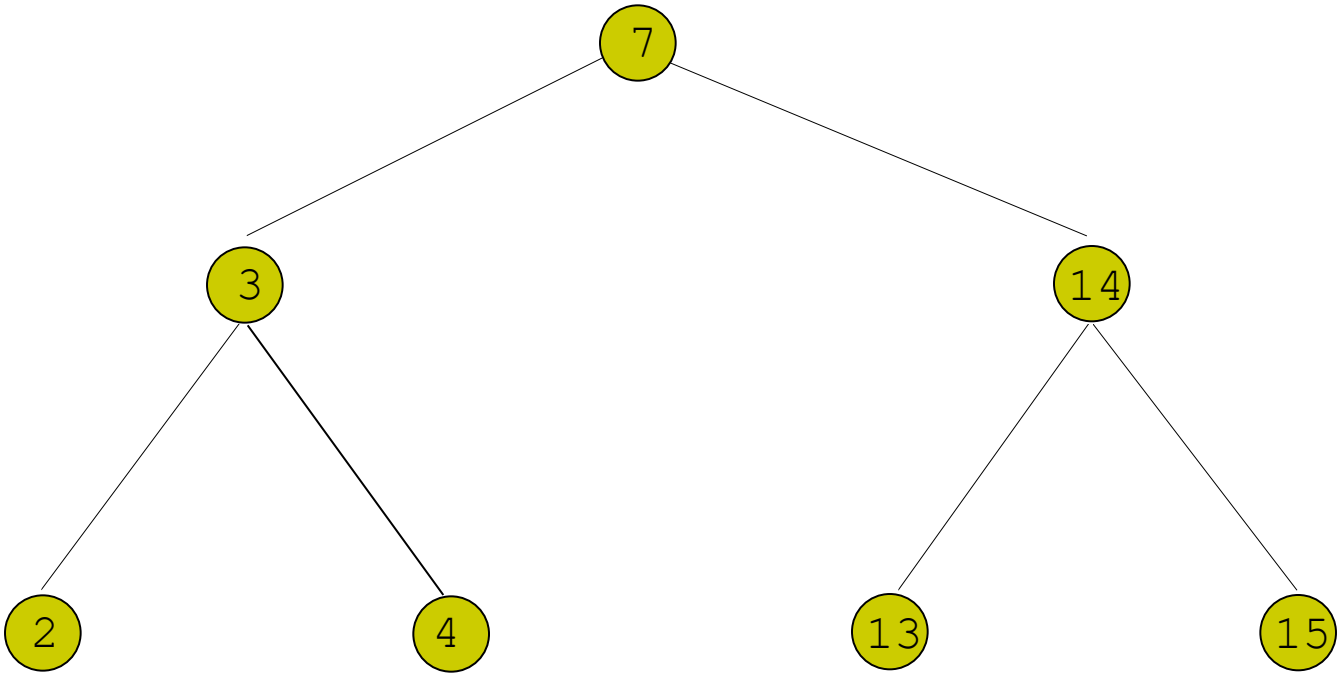

7
3
13
2
4
11
14
15


7
3
13
2
4
11
14
15


7
3
13
2
4
14
15


7
3
13
2
4
14
15


7
3
14
2
4
13
15


7
3
14
2
4
13
15

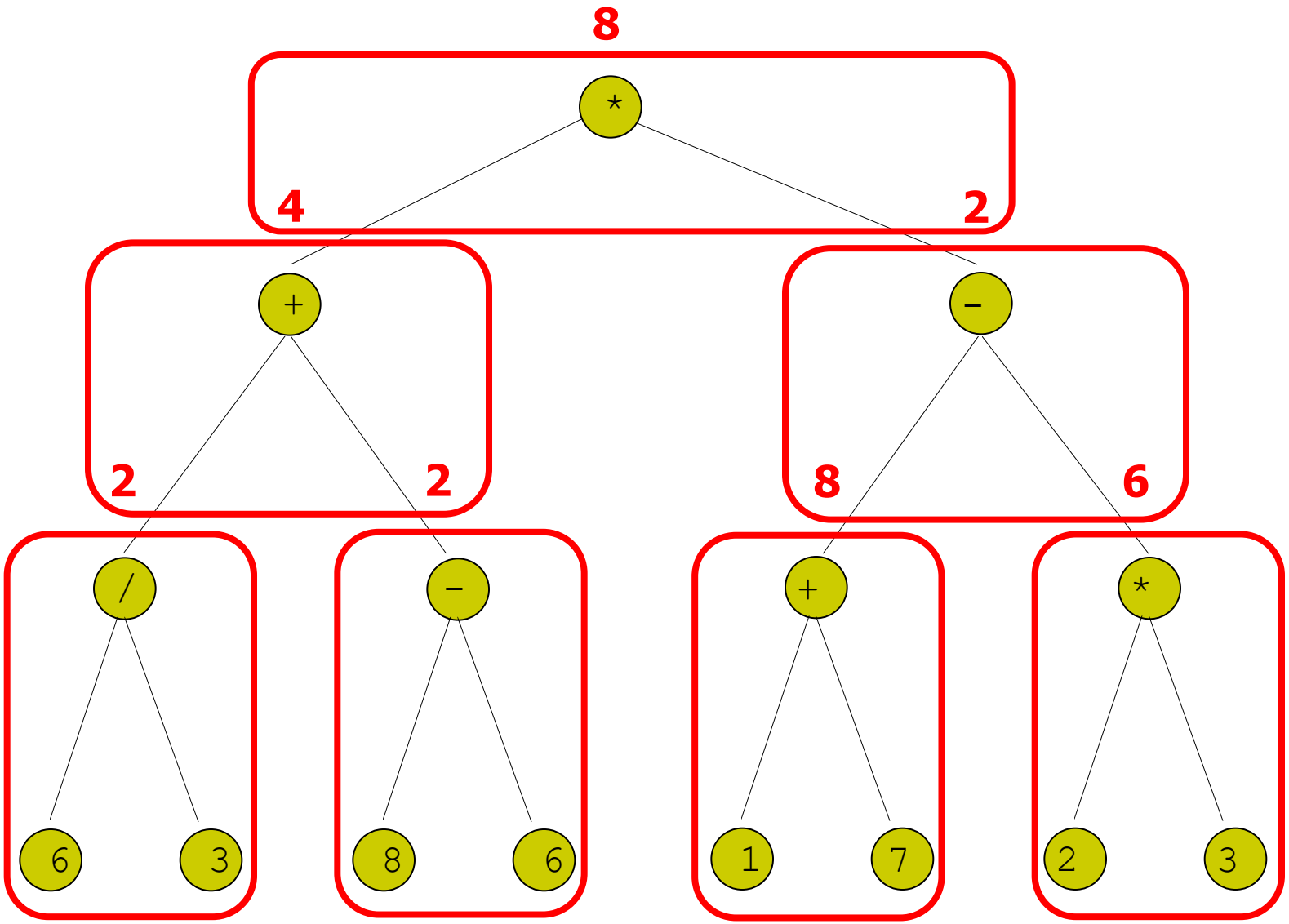
Αναπαράσταση/αποτίμηση εκφράσεων

- Τα δυαδικά δέντρα μπορεί να χρησιμοποιηθούν και για την εσωτερική αναπαράσταση εκφράσεων
  - π.χ. αριθμητικών εκφράσεων σε γλώσσες προγραμματισμού για την υποβοήθηση της σειράς/παραλληλισμού αποτίμησης

- Οι κόμβοι που δεν είναι φύλλα, περιέχουν τελεστές
- Τα ορίσματα (μπορεί να είναι σύνθετες εκφράσεις) βρίσκονται στο αριστερό και στο δεξί υποδέντρο

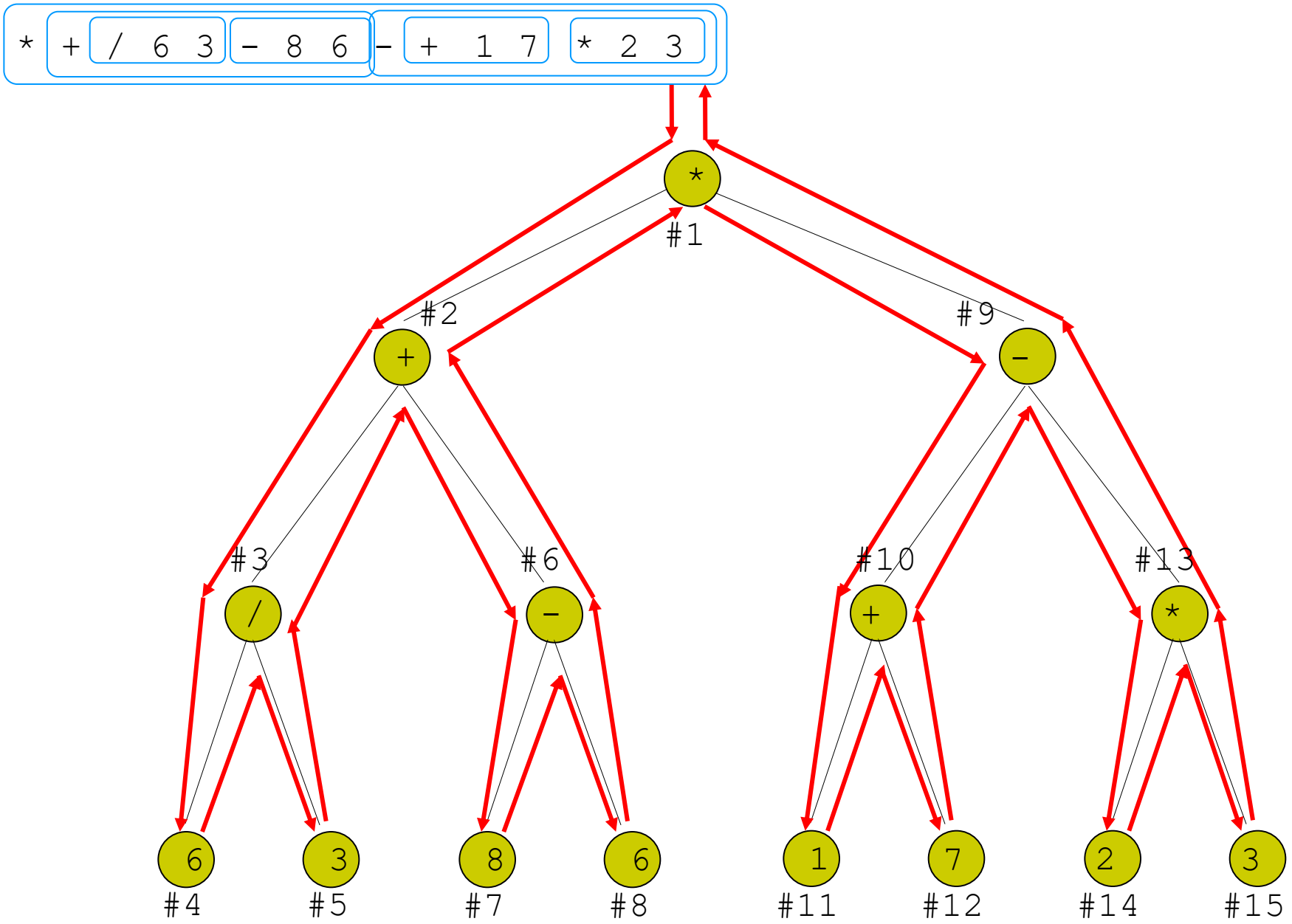
- Τα διάφορα παρακλάδια του δέντρου μπορεί να αποτιμηθούν/εκτελεστούν παράλληλα μεταξύ τους

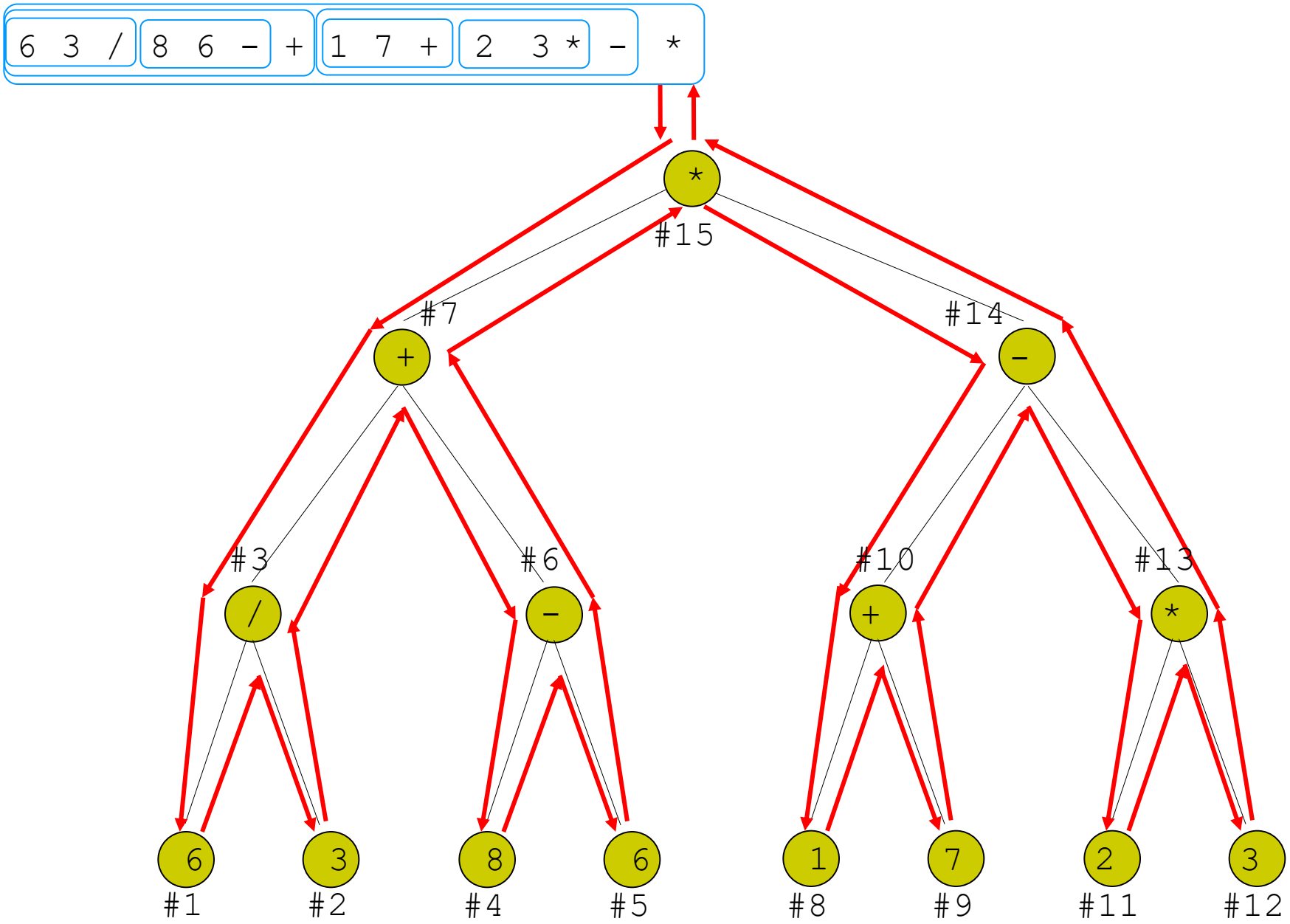
```
((6 / 3) + (8 - 6)) * ((1 + 7) - (2 * 3))
```



Διάσχιση δέντρου έκφρασης

- Μπορεί να γίνει με διαφορετικούς τρόπους
  - κάθε τρόπος αντιστοιχεί σε μια διαφορετική συντακτική αναπαράσταση για την ίδια αριθμητική έκφραση
- **pre-order:** ο τελεστής πριν τα ορίσματα
  - prefix notation
- **post-order:** ο τελεστής μετά τα ορίσματα
  - postfix notation
- **in-order:** ο τελεστής ανάμεσα στα ορίσματα
  - infix notation (αυτό που χρησιμοποιούμε συνήθως)
- **Παρατήρηση:** η infix αναπαράσταση **δεν** έχει ξεκάθαρες προτεραιότητες
  - πρέπει να οριστούν προτεραιότητες για τους τελεστές και σειρά αποτίμησης για τελεστές ίδιας προτεραιότητας


* + / 6 3 - 8 6 - + 1 7 * 2 3
*
#1
+
#2
-
#9
/
#3
-
#6
+
#10
*
#13
6
#4
3
#5
8
#7
6
#8
1
#11
7
#12
2
#14
3
#15


6 3 / 8 6 - + 1 7 + 2 3 * - *
*
#15
#7
+
#14
-
#3
/
#6
-
#10
+
#13
*
6
#1
3
#2
8
#4
6
#5
1
#8
7
#9
2
#11
3
#12