Διασυνδεδεμένες Δομές

Δυαδικά Δέντρα

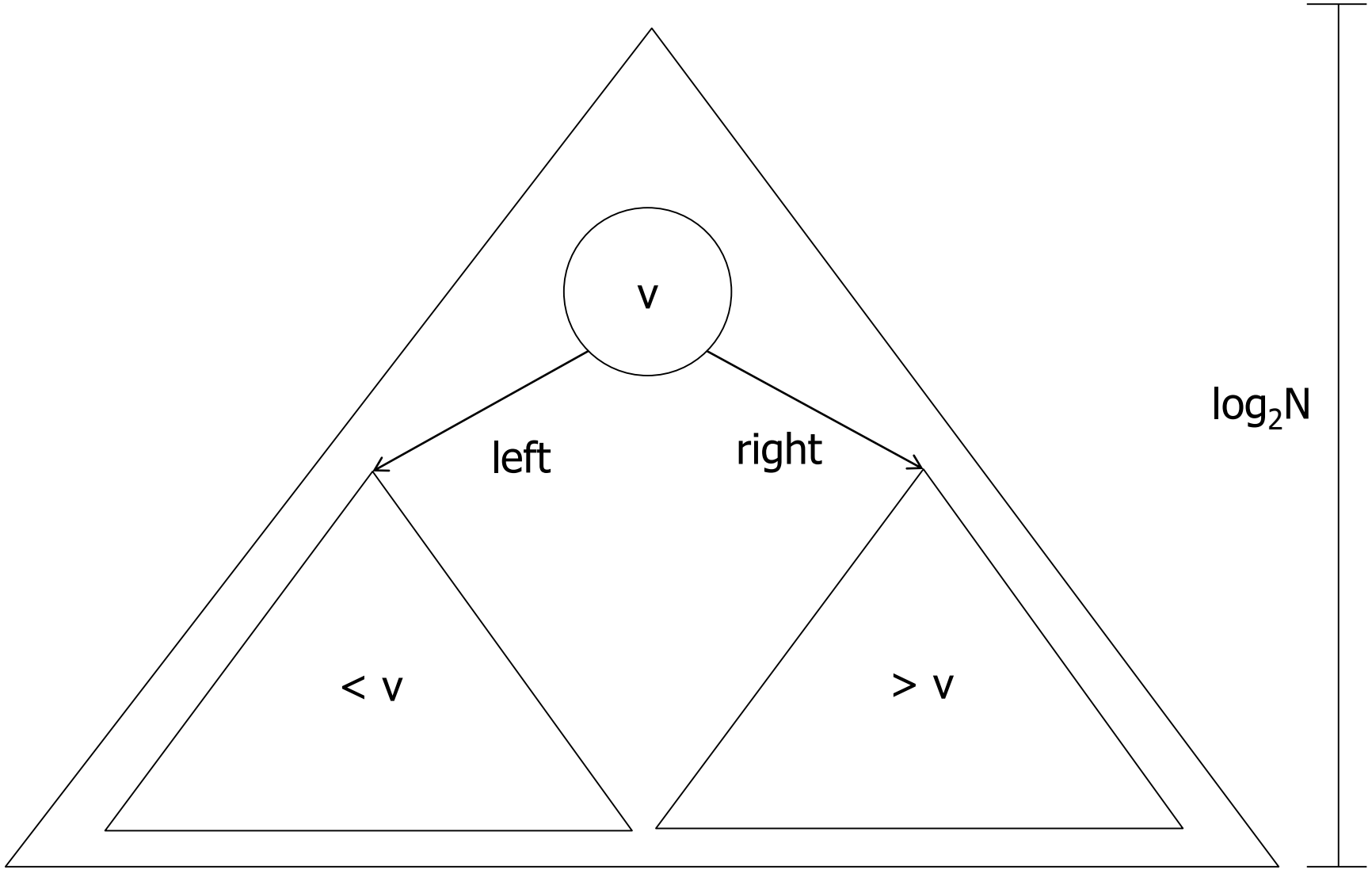
Δέντρα

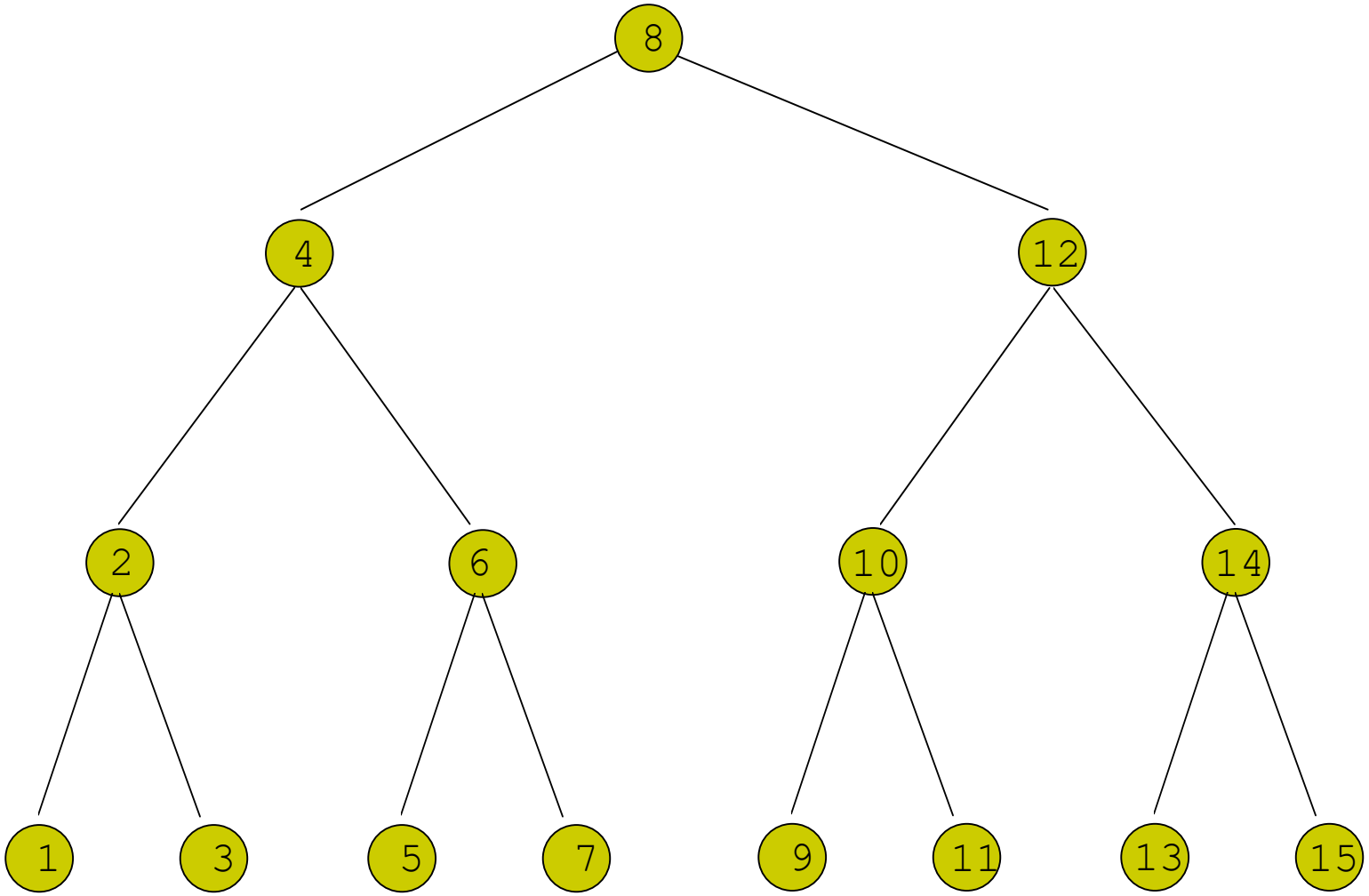
- Τα δέντρα είναι κλασικές **αναδρομικές** δομές
- Ένα δέντρο αποτελείται από υποδέντρα, καθένα από τα οποία μπορεί να θεωρηθεί ως ένα (άλλο) δέντρο, που πάλι αποτελείται από υποδέντρα κλπ.
- Η αναζήτηση σε δέντρα μπορεί να υλοποιηθεί πολύ εύκολα με **αναδρομικό** τρόπο

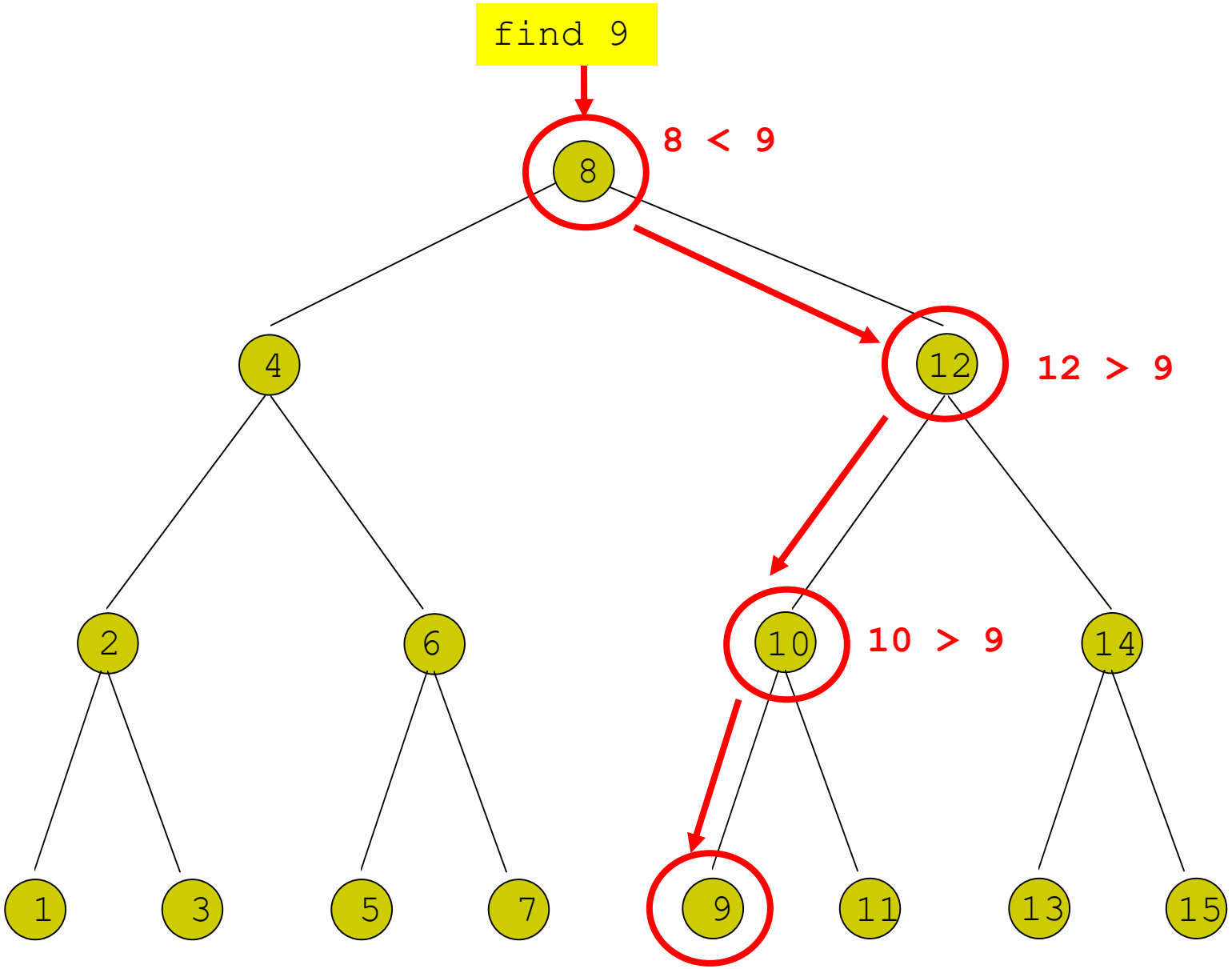
Δυαδικά δέντρα αναζήτησης

- Τα δυαδικά δέντρα είναι ειδική περίπτωση δέντρων
- Κάθε κόμβος έχει (το πολύ) δύο υποδέντρα

- Η κατασκευή μπορεί να γίνει με ειδικό τρόπο, έτσι ώστε να υποστηρίζεται η **γρήγορη αναζήτηση**
- Για κάθε κόμβο με τιμή κλειδιού v όλοι οι κόμβοι του αριστερού υποδέντρου έχουν τιμή **μικρότερη** του v, και όλοι οι κόμβοι του δεξιού υποδέντρου έχουν τιμή **μεγαλύτερη** του v
- Αν το δέντρο είναι **ισορροπημένο** (balanced), η αναζήτηση απαιτεί κατά μ.ο. $(\log_2 N)/2$ βήματα όπου N το πλήθος των κόμβων του δέντρου


v
left
right
< v
> v
log2N


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


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

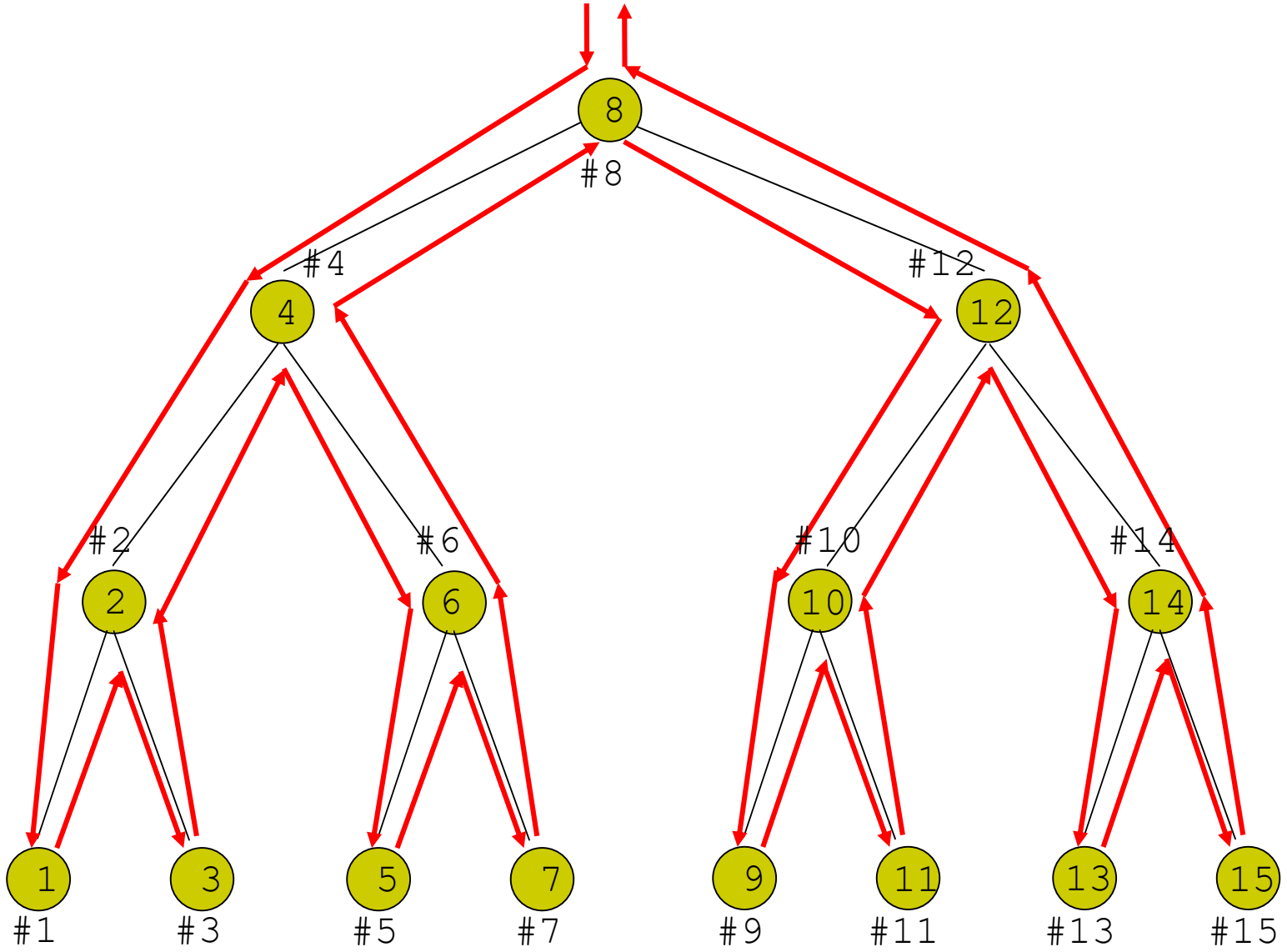
```
struct btree {
  int v;
  struct btree *left,*right;
};
```

```
int btree_find2(struct btree *root, int v) {
  if (root == NULL) {
    return(0);
  }
  else if (root->v == v) {
    return(1);
  }
  else if (root->v > v) {
    return(btree_find2(root->left,v));
  }
  else /* root->v < v */ {
    return(btree_find2(root->right,v));
  }
}
```

Διέλευση δυαδικών δέντρων

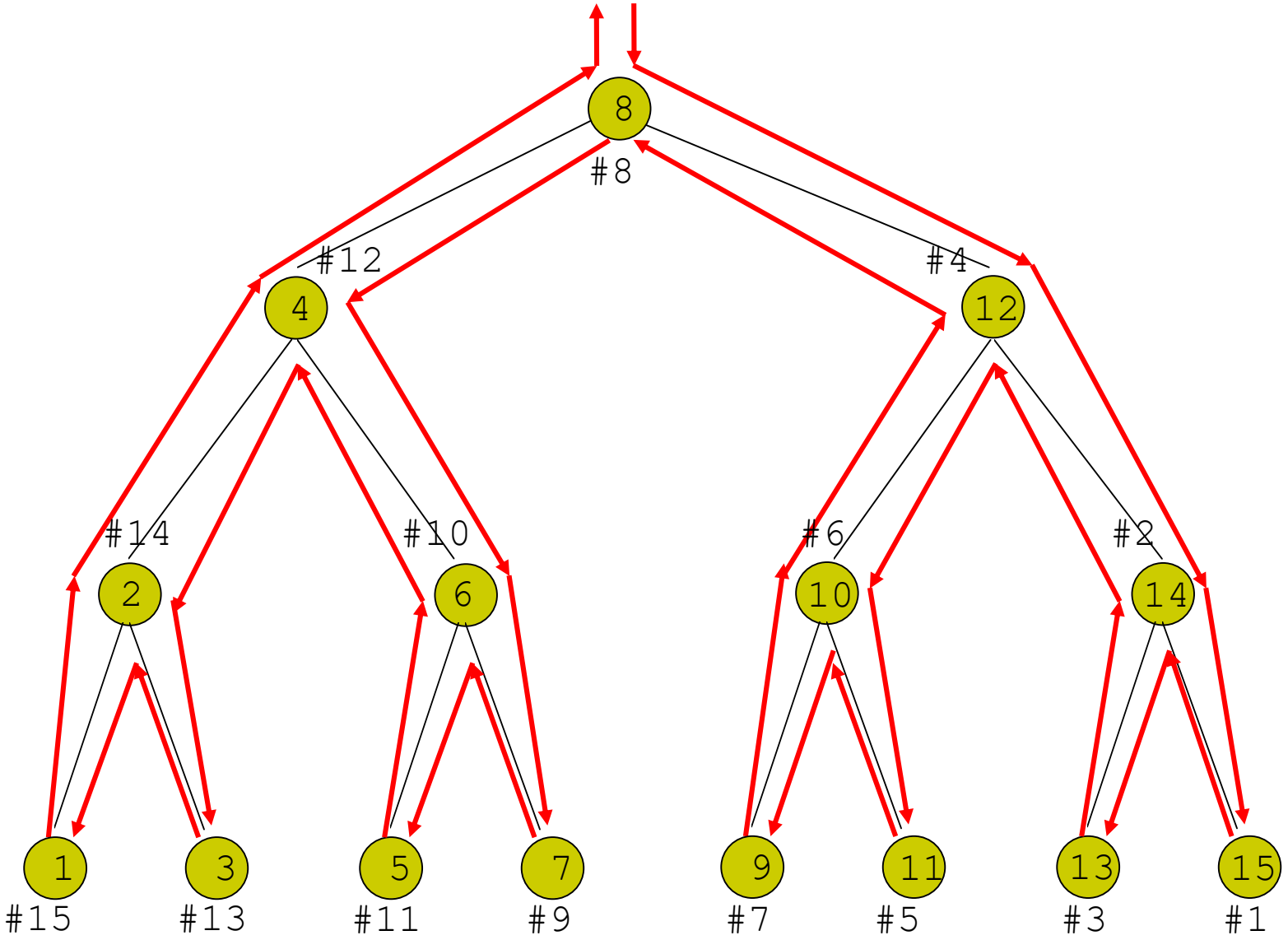
- Η διεύλευση ενός δέντρου μπορεί επίσης να γίνει πολύ εύκολα με **αναδρομικό** τρόπο
- Μπορεί να γίνει επίσκεψη των κόμβων του δέντρου με «αύξουσα» ή με «φθίνουσα» σειρά
- Αναλόγως με το σημείο που βάζουμε την αναδρομή

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



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

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

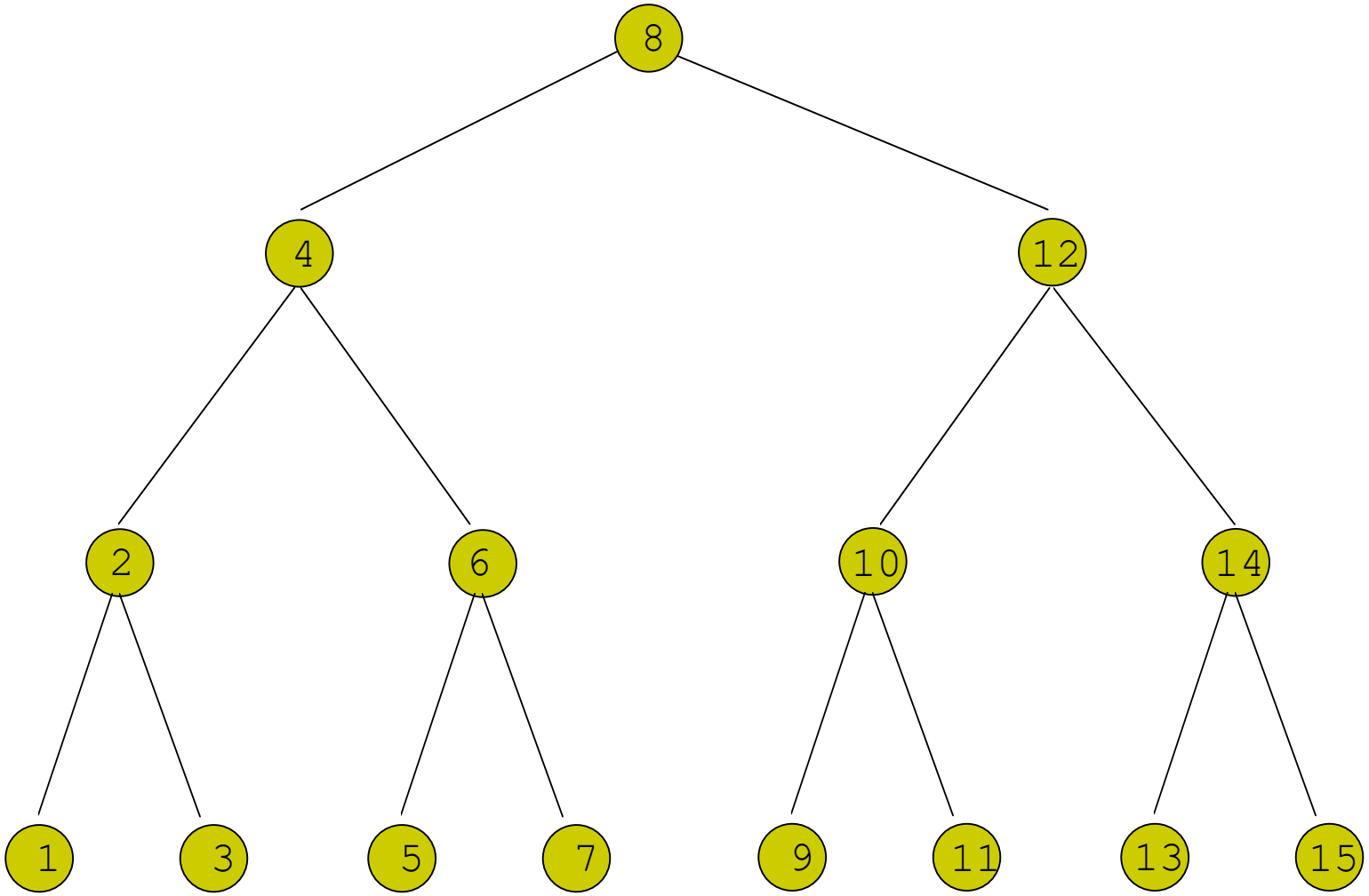


```
void btree_traverse_dec(struct btree *root) {
  if (root != NULL) {
    btree_traverse_dec(root->right);
    printf("%d ",t->v);
    btree_traverse_dec(root->left);
  }
}
```

Προσθήκη κόμβου

- Αναζήτηση με βάση την τιμή-κλειδιού v του κόμβου
- Αν ο κόμβος βρεθεί, γίνεται αντικατάσταση
  - τα δεδομένα αντιγράφονται στον κόμβο που ήδη υπάρχει
- Αν ο κόμβος δεν βρεθεί, τότε ο τελευταίος κόμβος που επισκευτήκαμε είναι ένα φύλλο του δέντρου
- Ο νέος κόμβος εισάγεται κάτω από τον κόμβο-φύλλο, έτσι ώστε να τηρείται η **σύμβαση αναζήτησης**

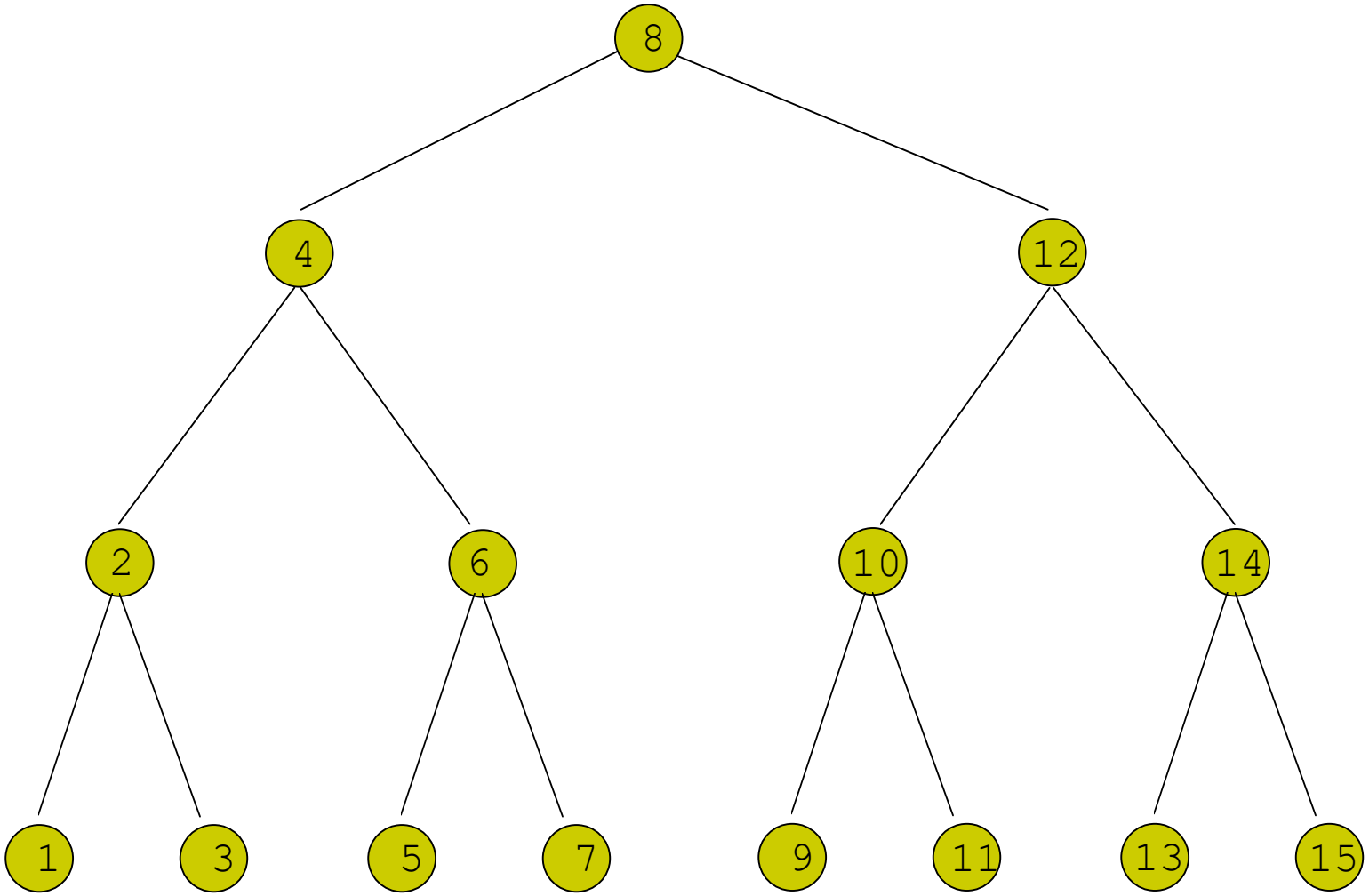
- Έστω ότι η τιμή του κλειδιού στο φύλλο είναι v'
- Αν v < v', ο νέος κόμβος εισάγεται στα αριστερά
- Αν v > v', ο νέος κόμβος εισάγεται στα δεξιά

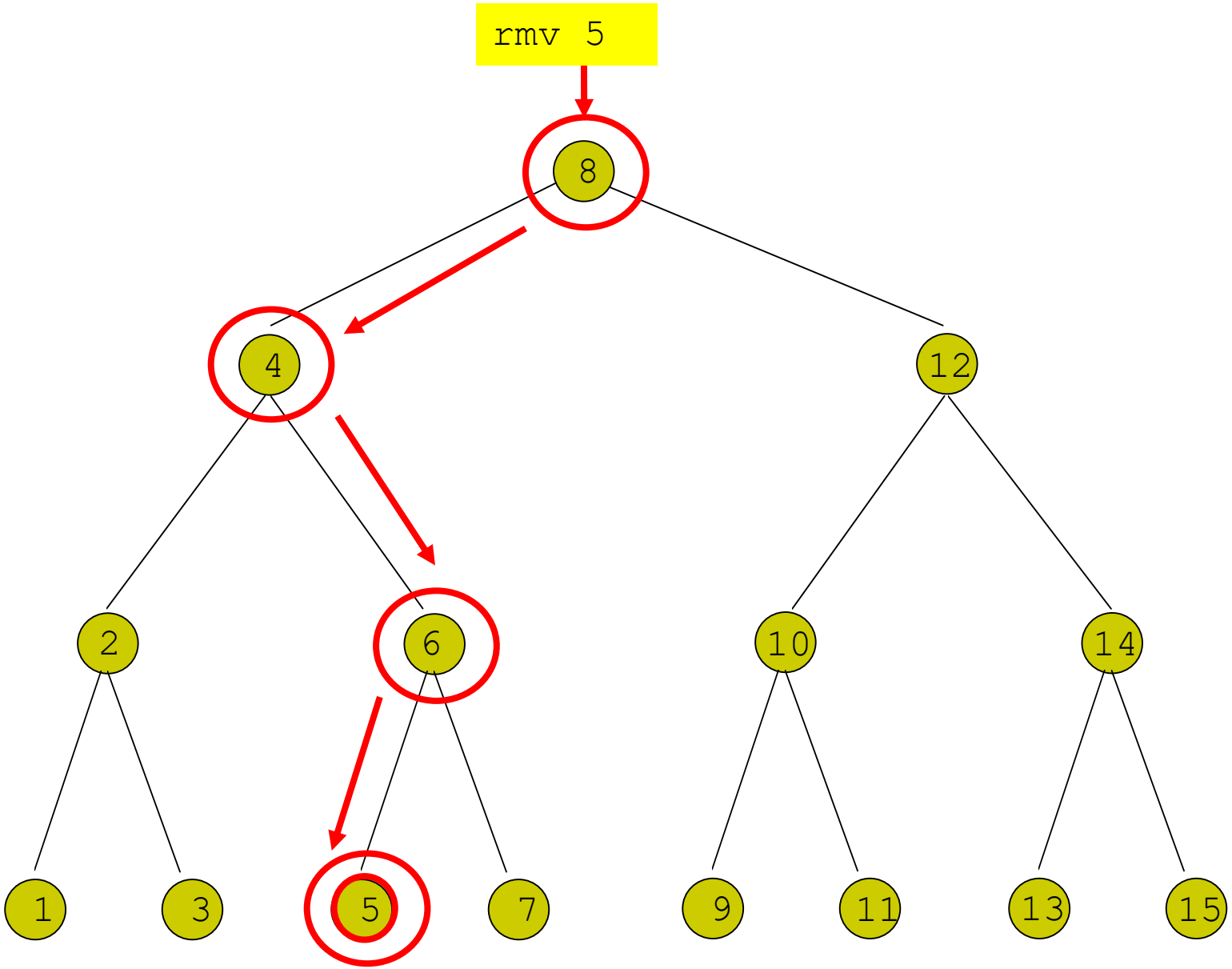

8
4
12
2
6
10
14
1
3
5
7
9
11
13
15

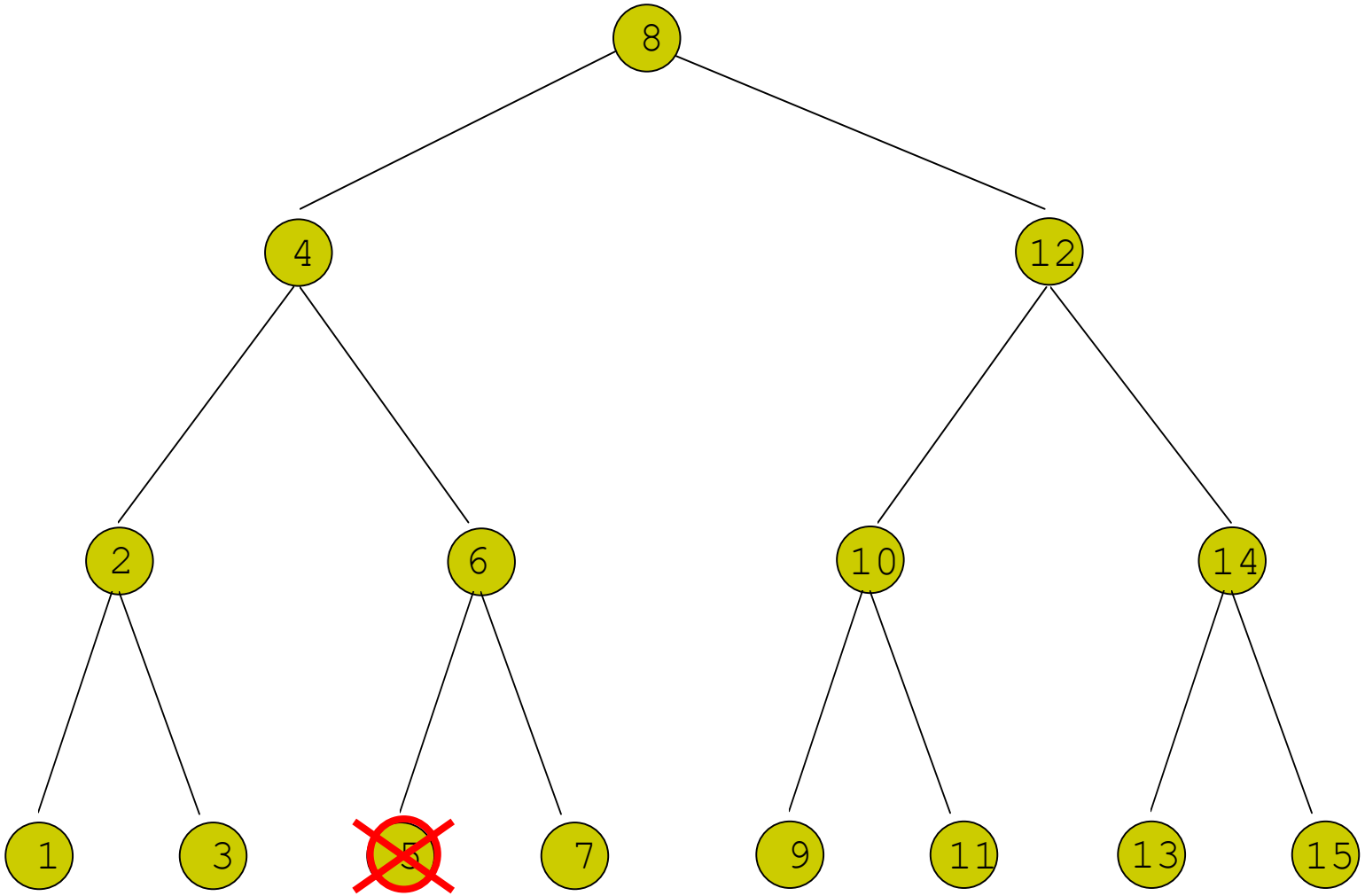
Αφαίρεση κόμβου

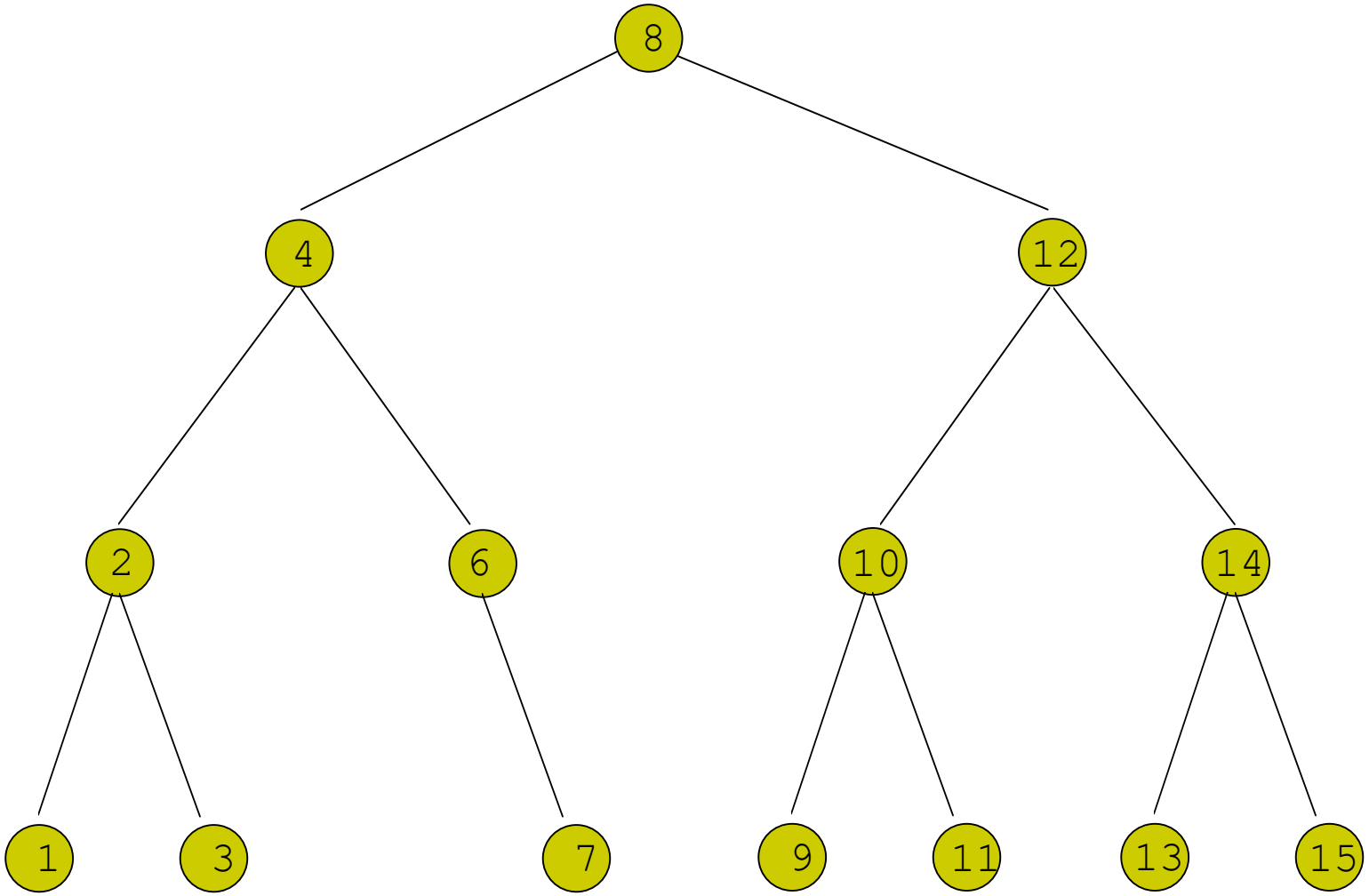
- Αναζήτηση με βάση την τιμή-κλειδιού v του κόμβου
- Αν ο κόμβος δεν βρεθεί, τερματίζουμε
- Αν ο κόμβος βρεθεί, τότε τον αφαιρούμε

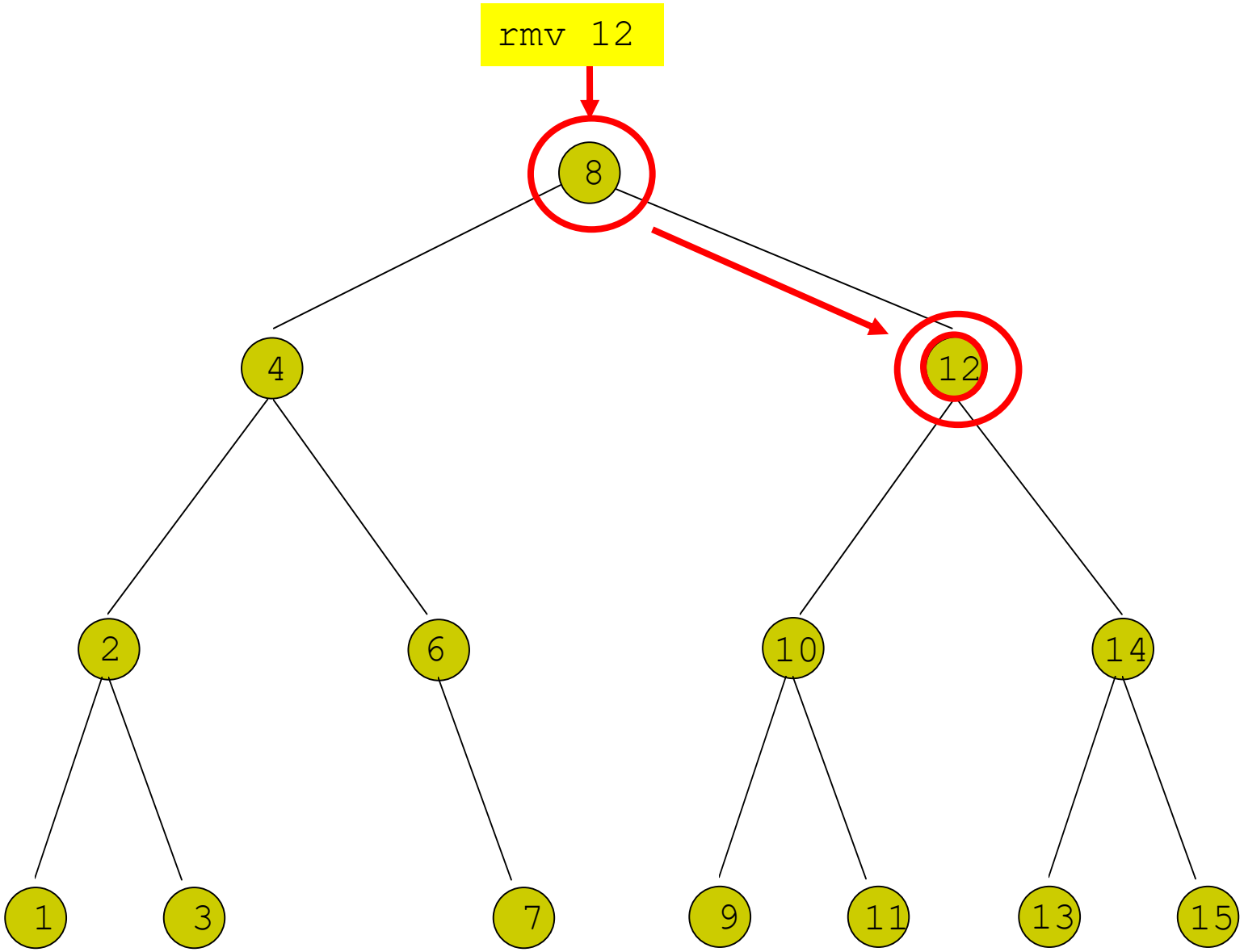
- Αν είναι φύλλο, απλά τον διαγράφουμε
- Αν είναι η **ρίζα του δέντρου**, «προάγουμε» σε ρίζα του δέντρου την ρίζα του αριστερού ή του δεξιού υποδέντρου – αναδρομικά για τα υποδέντρα
- Τα υποδέντρα πρέπει να **αναδιατάσσονται** κατάλληλα για να τηρείται η συνθήκη αναζήτησης


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

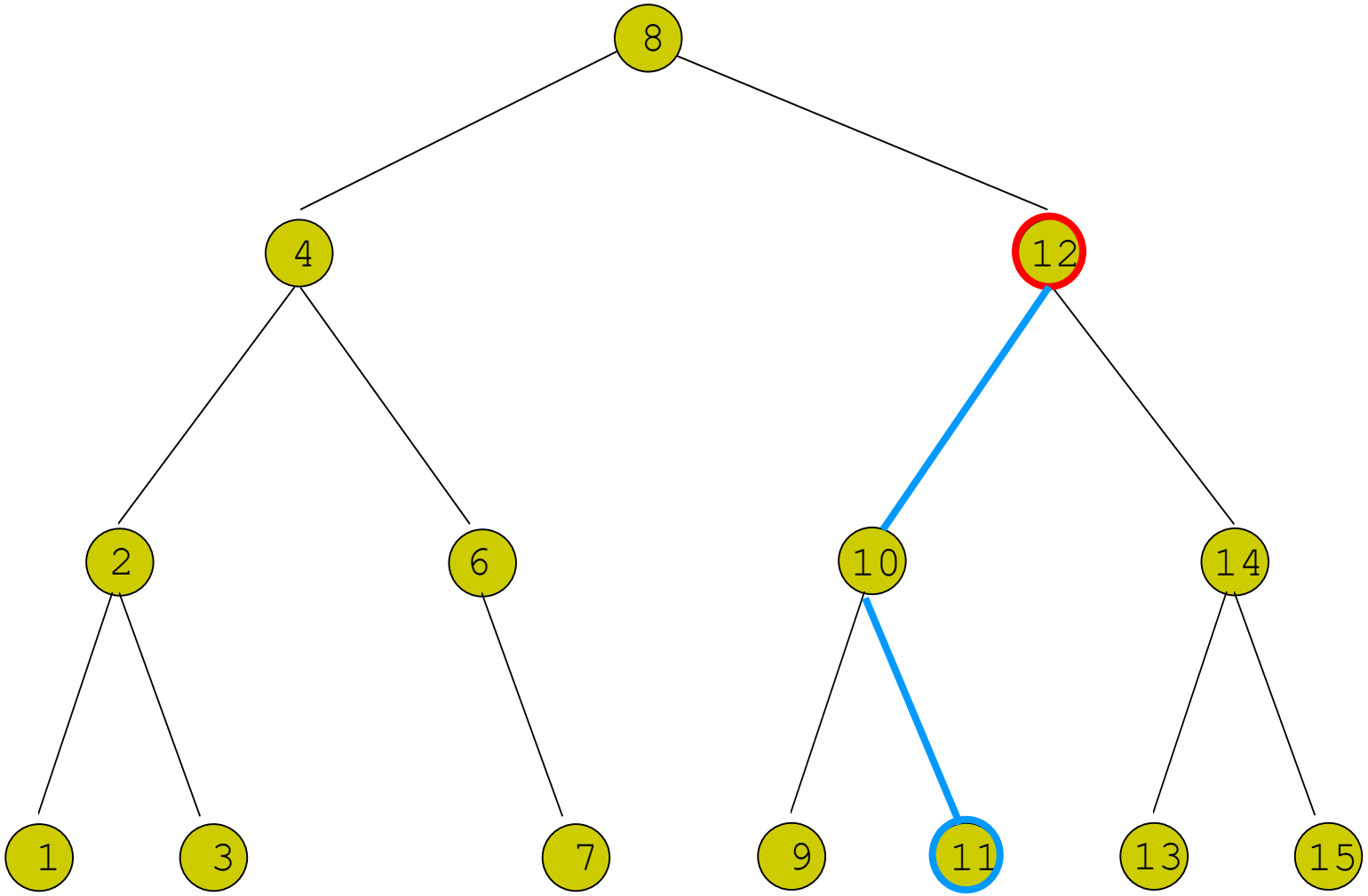

rmv 5
8
4
12
2
6
10
14
1
3
5
7
9
11
13
15


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

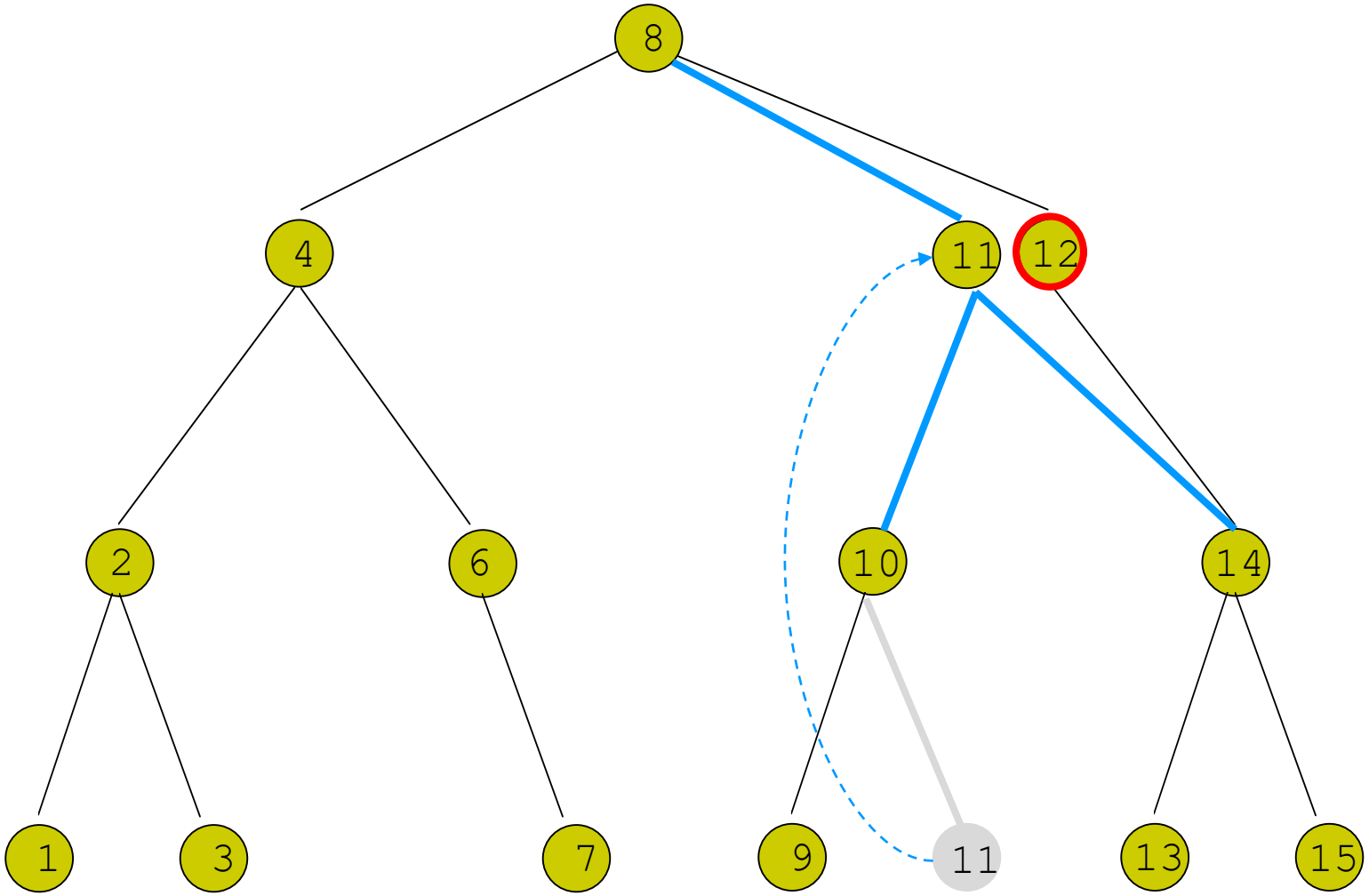

8
4
12
2
6
10
14
1
3
7
9
11
13
15


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

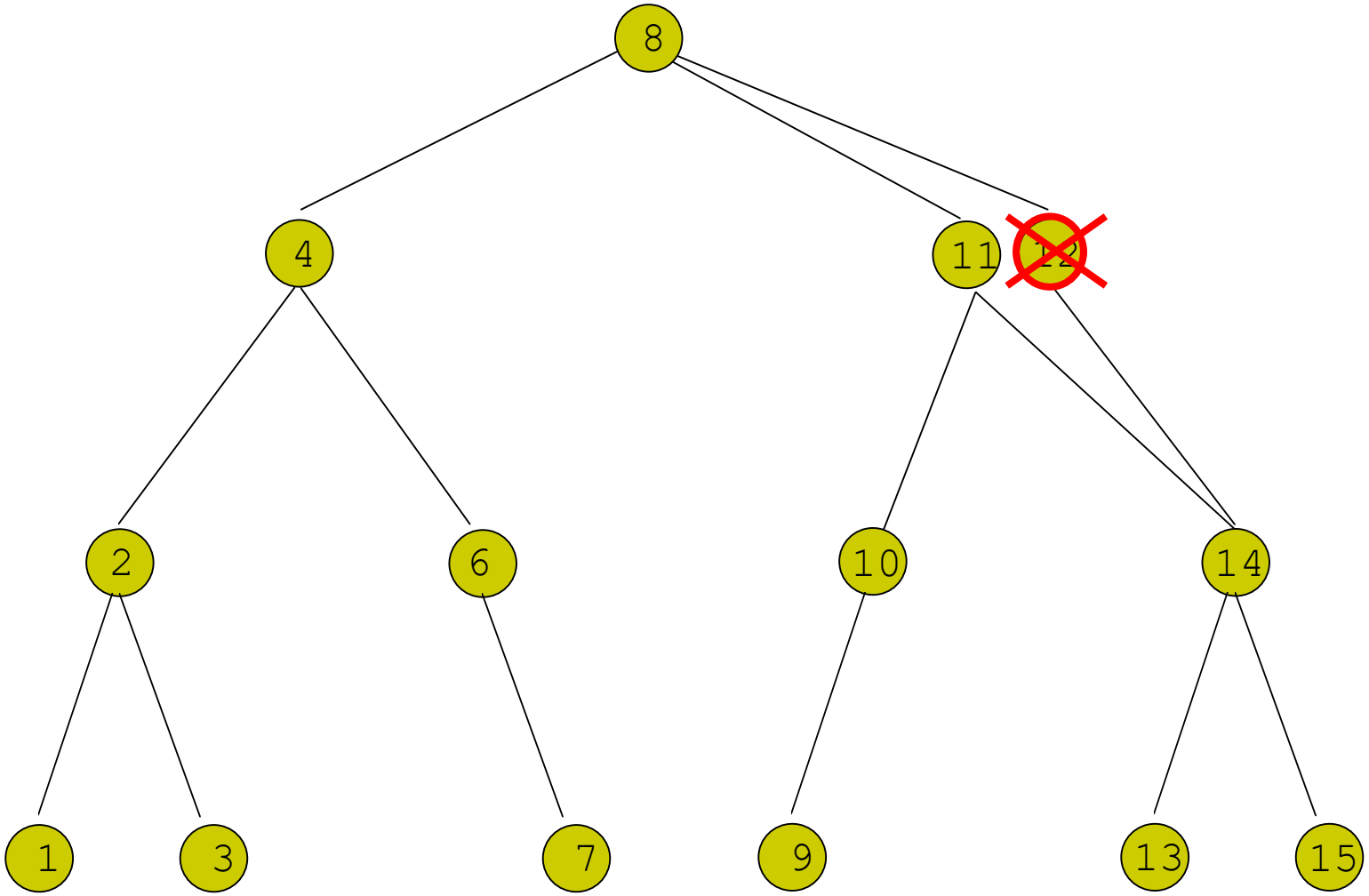
rmv 12

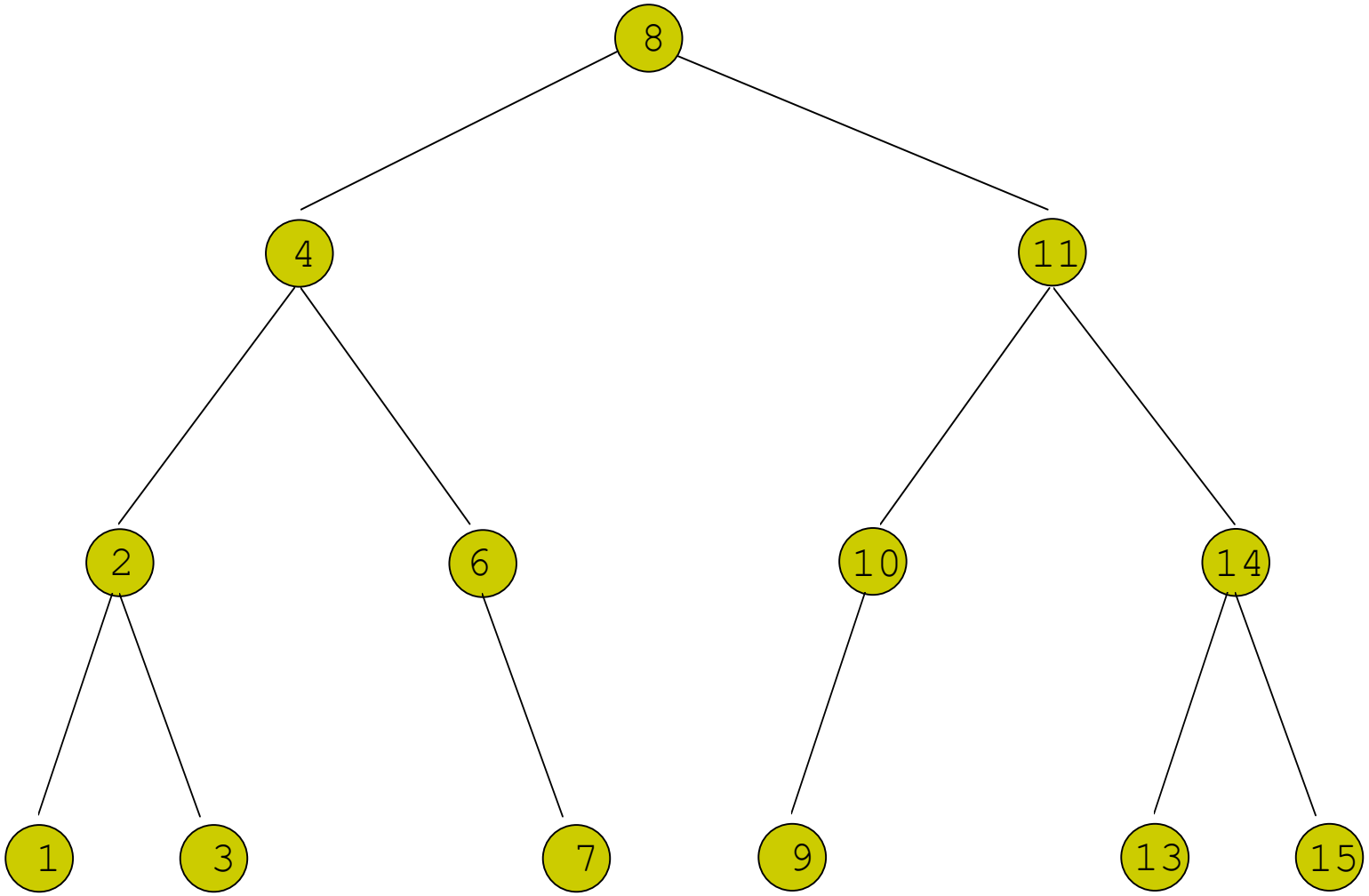


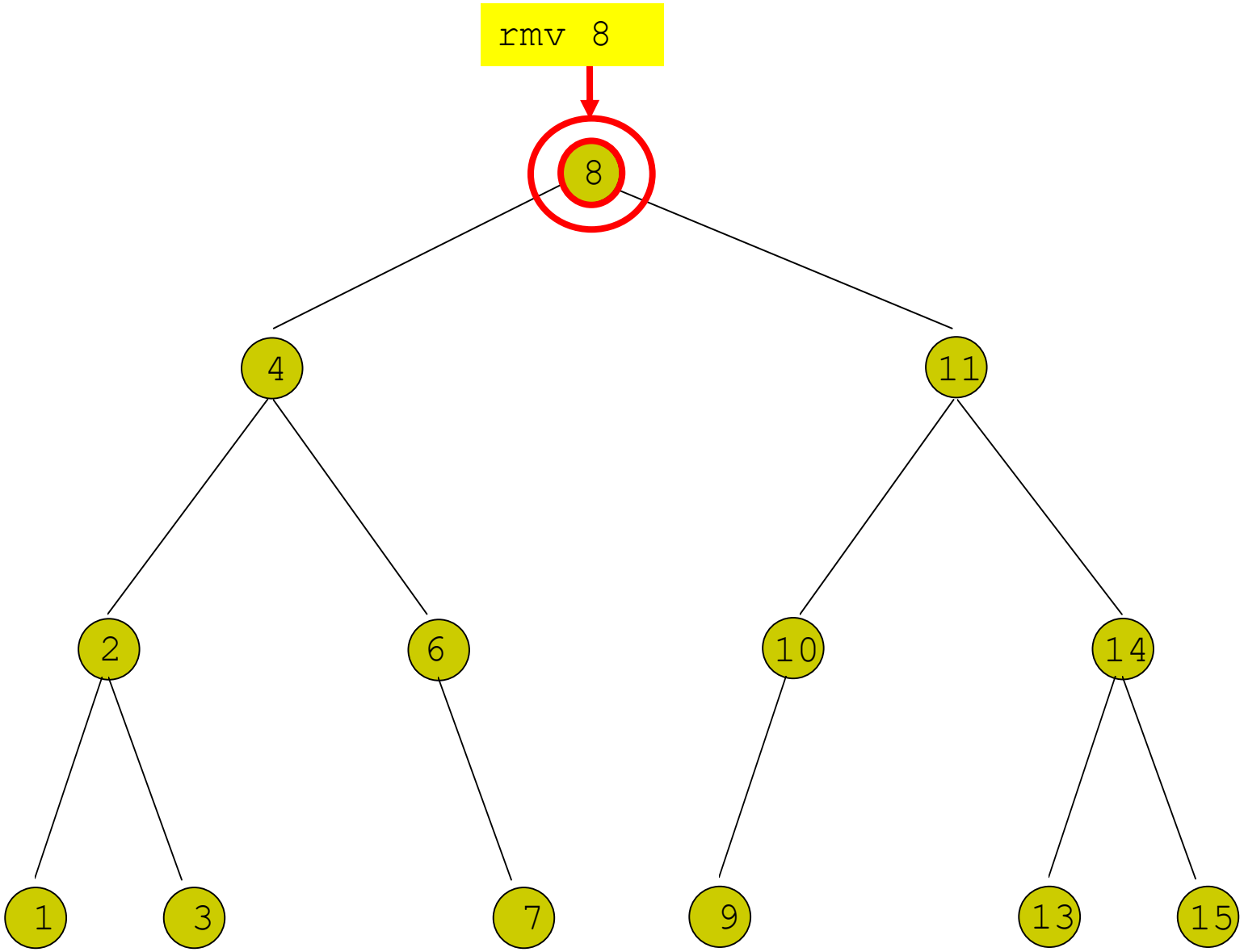
rmv 12

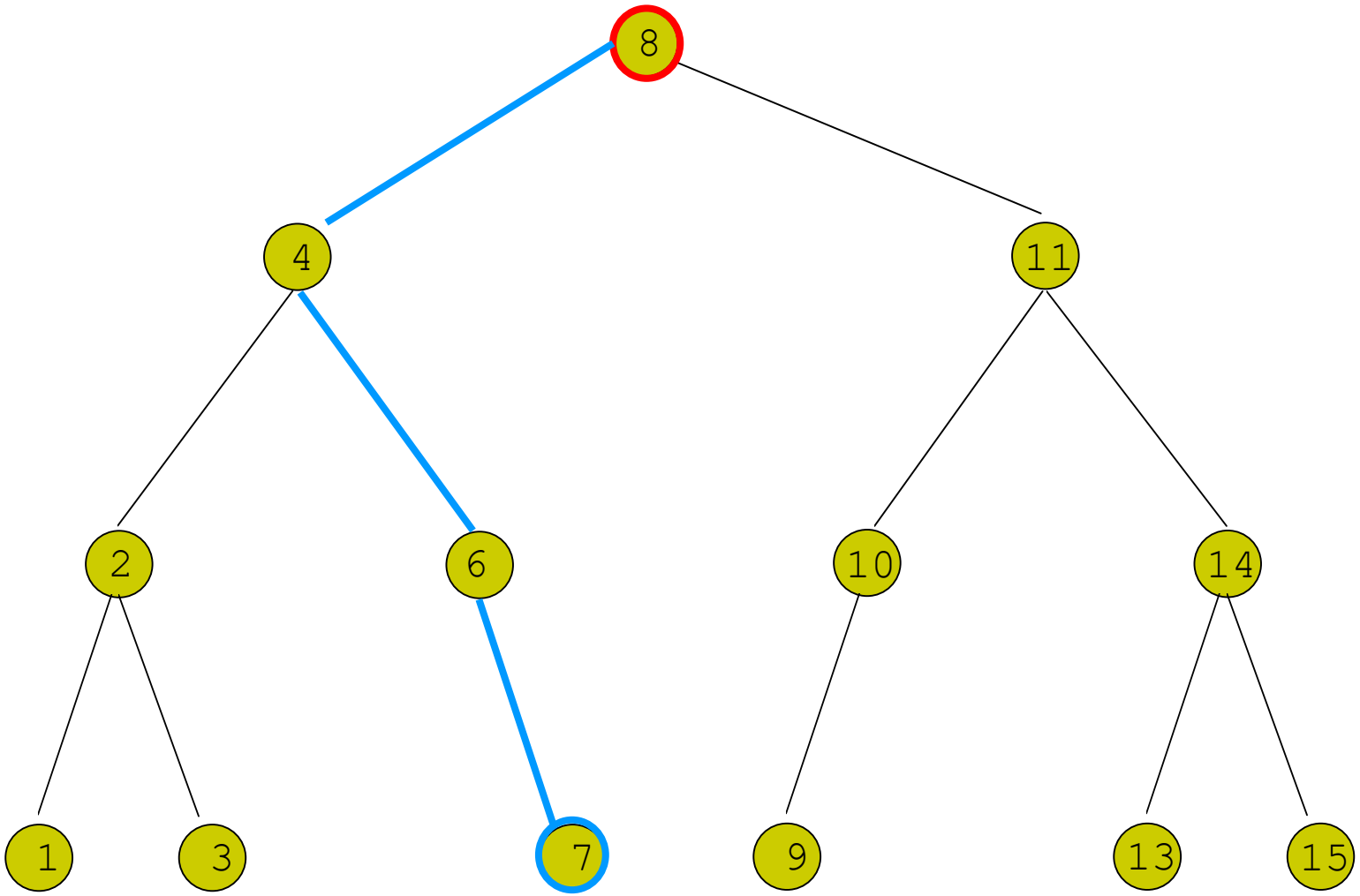


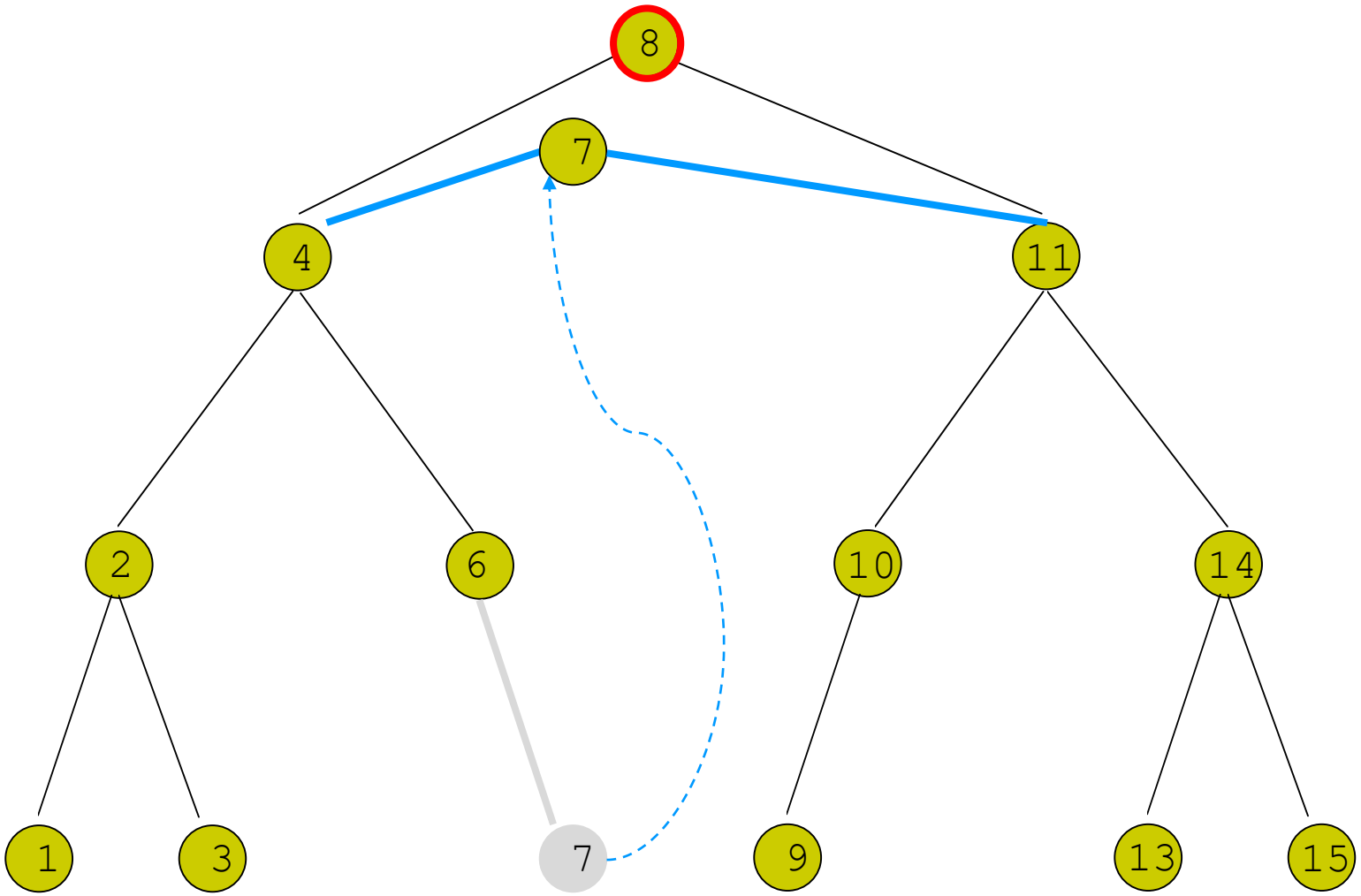
rmv 12

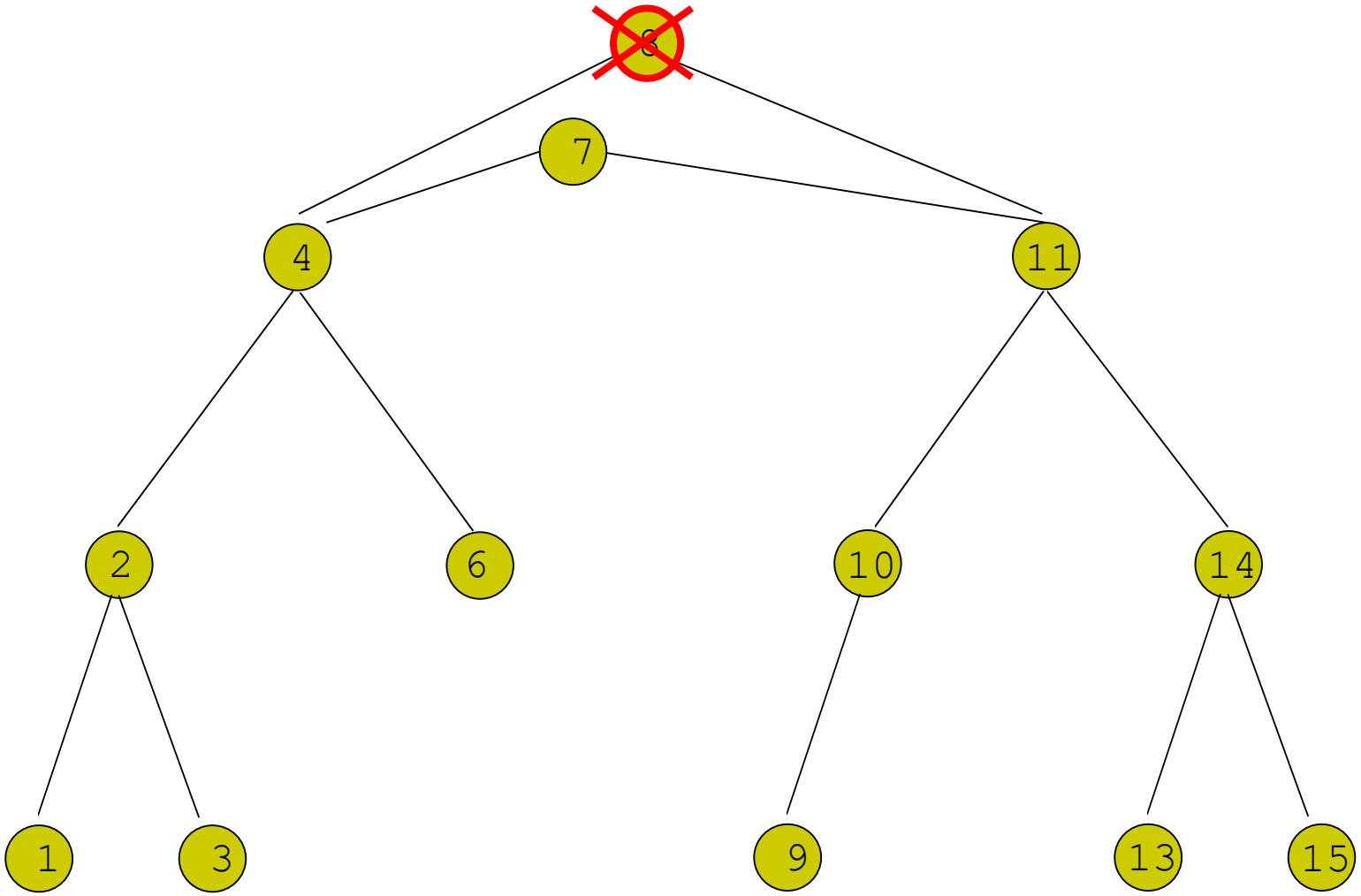


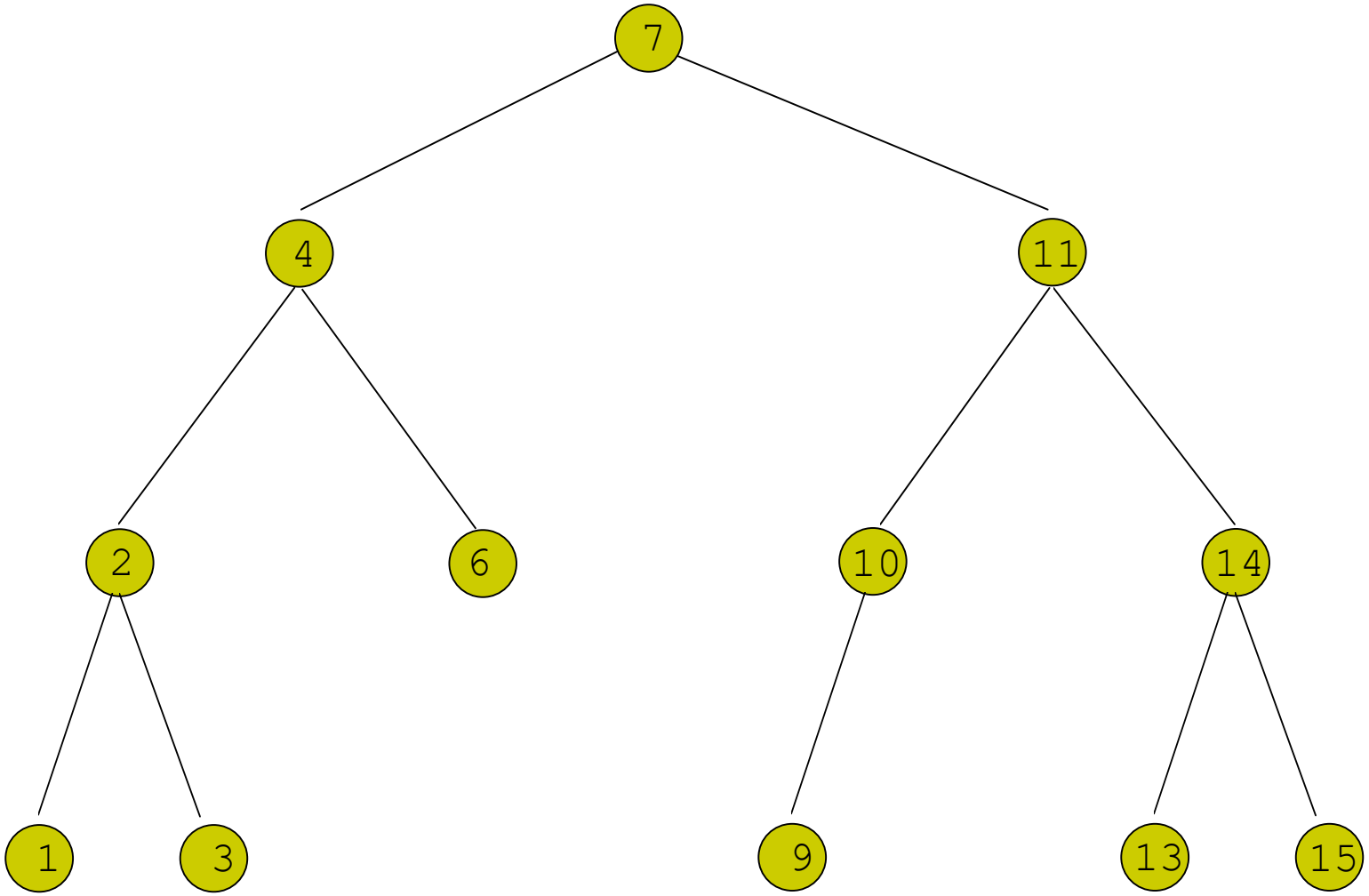

8
4
11
2
6
10
14
1
3
7
9
13
15


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


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

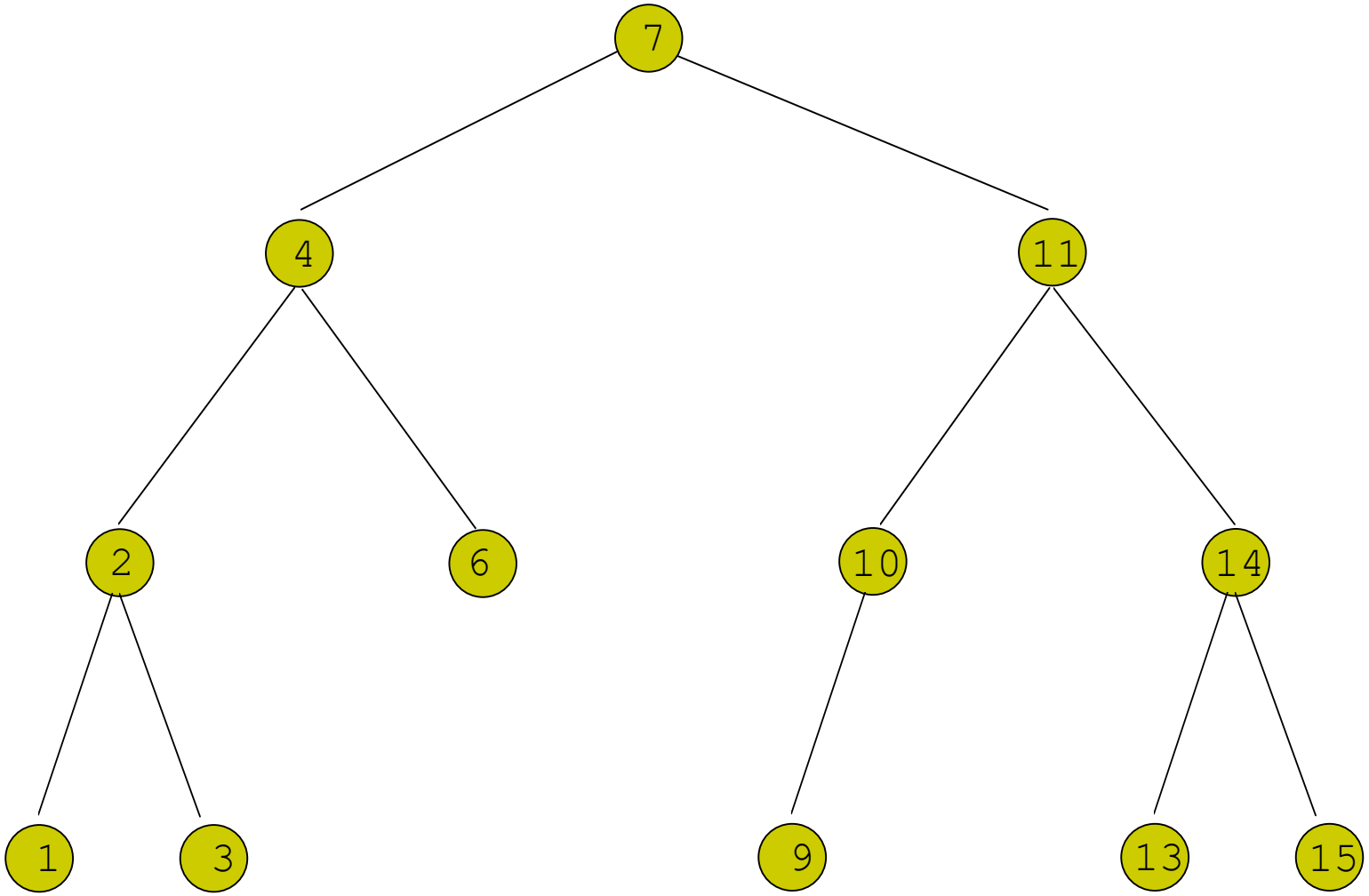

8
7
4
11
2
6
10
14
1
3
7
9
13
15

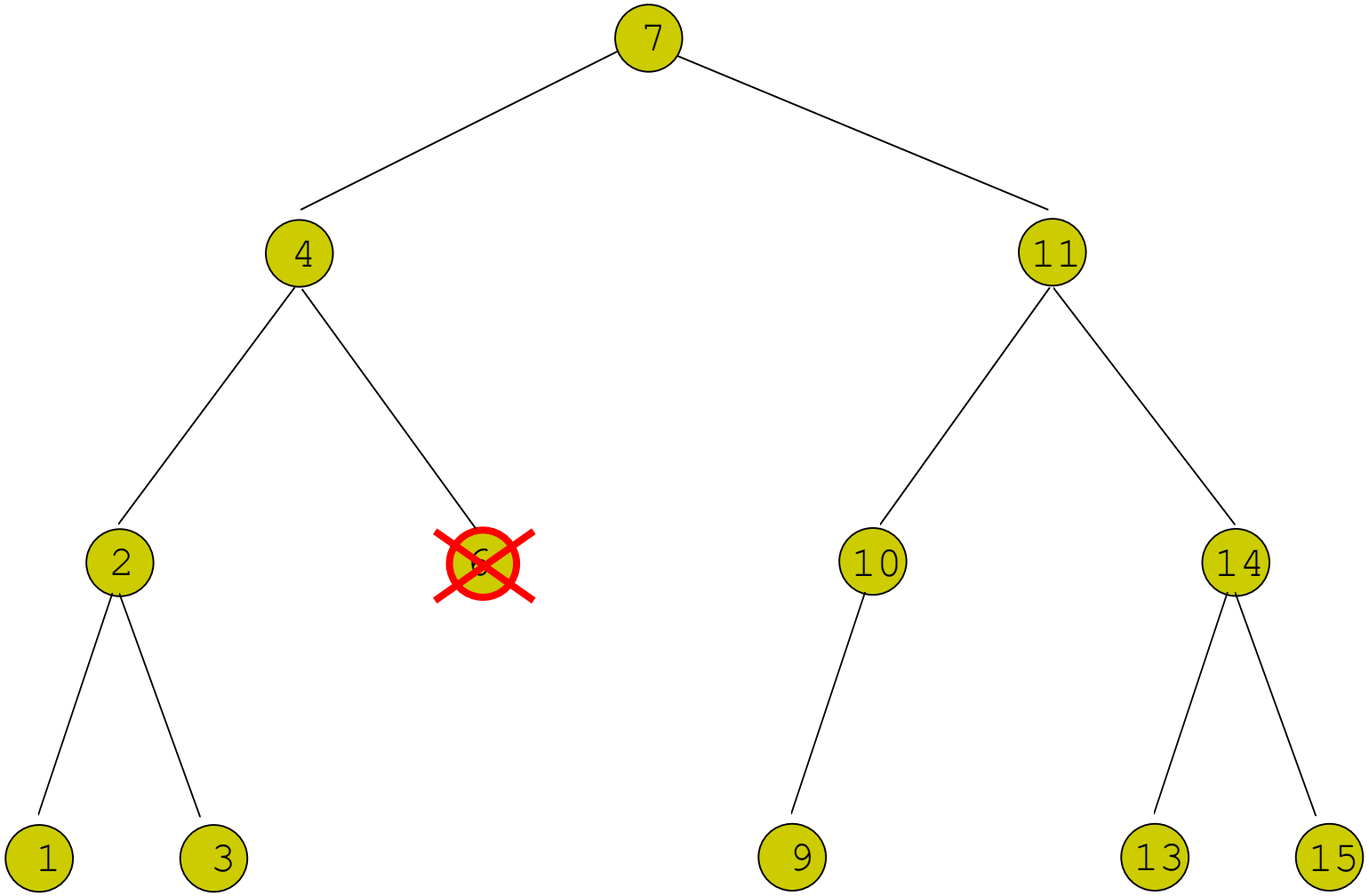

8
7
4
11
2
6
10
14
1
3
9
13
15

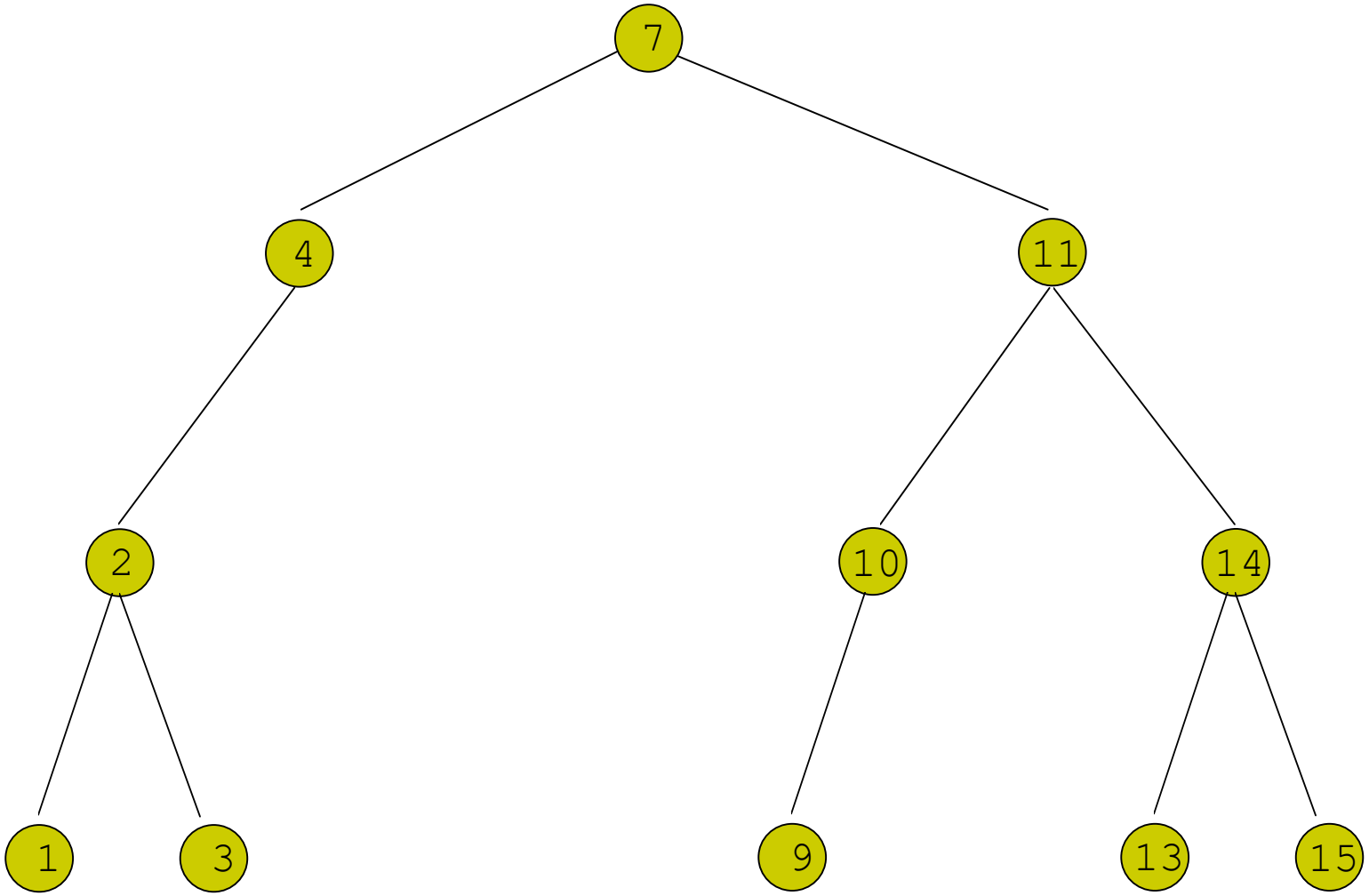

7
4
11
2
6
10
14
1
3
9
13
15

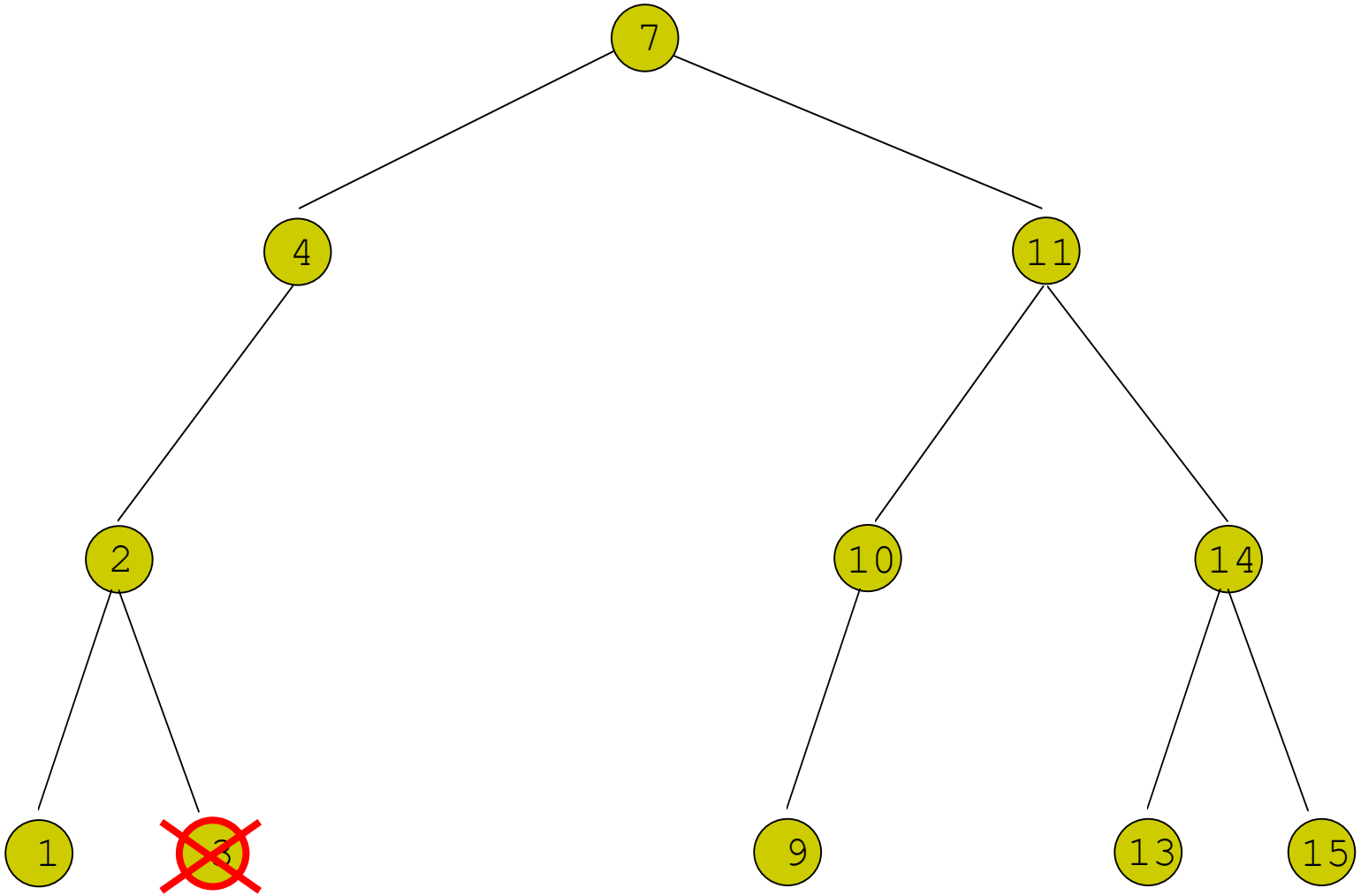
Εξισορρόπηση

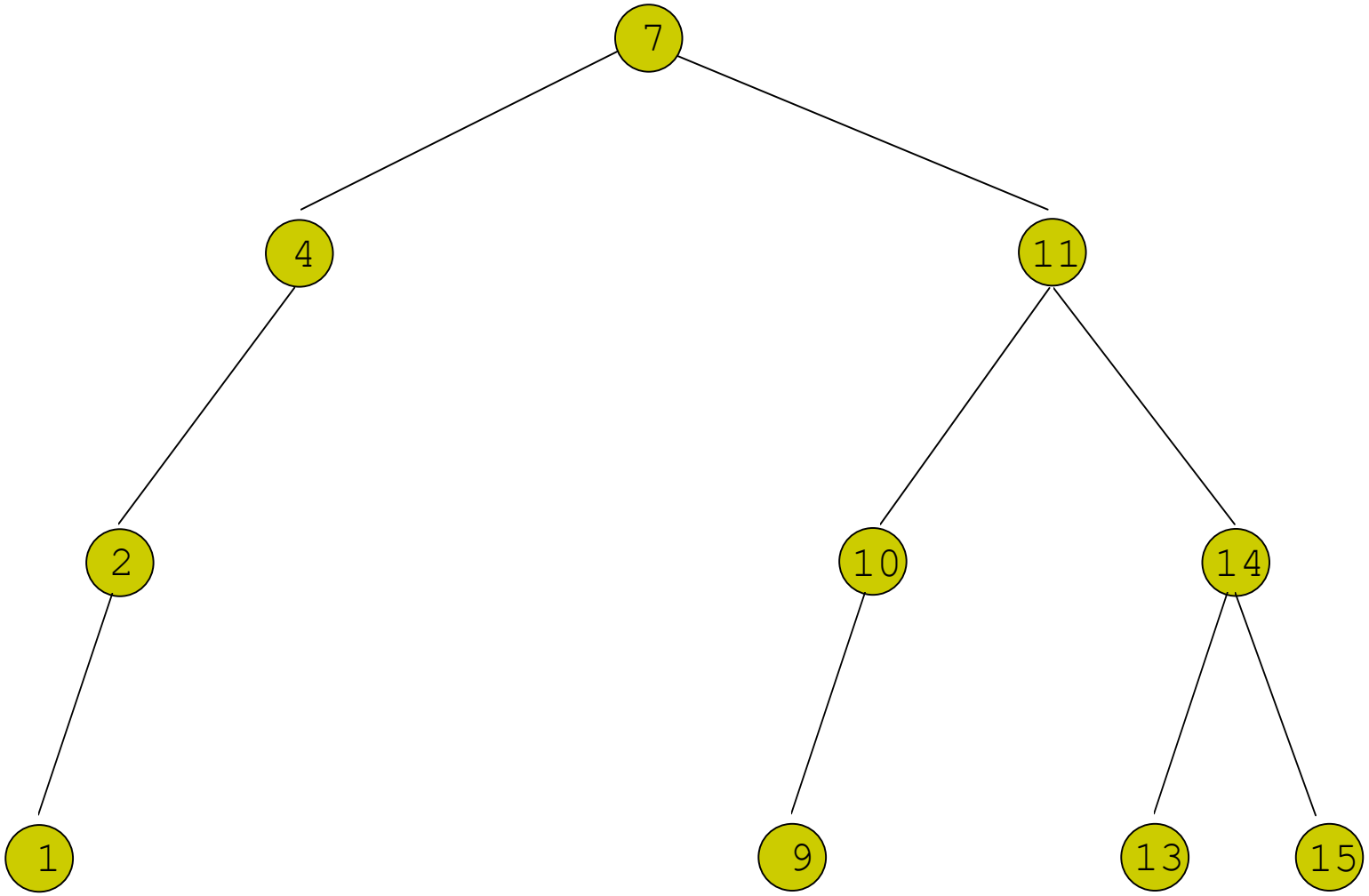
- Το δέντρο μπορεί να χάσει την «ισορροπία» του
- **Ακραία περίπτωση:** το δέντρο **«εκφυλίζεται»** σε μια (ταξινομημένη) λίστα, όπου ο μ.ο. Βημάτων αναζήτησης είναι πλέον N/2 (αντί για $\log_2 N$)
- **Λύση:** κάθε φορά που προστίθεται ή αφαιρείται ένας κόμβος ή όταν εντοπιστεί μεγάλη ανισορροπία, πραγματοποιούνται ενέργειες **εξισορόπησης** (balancing)

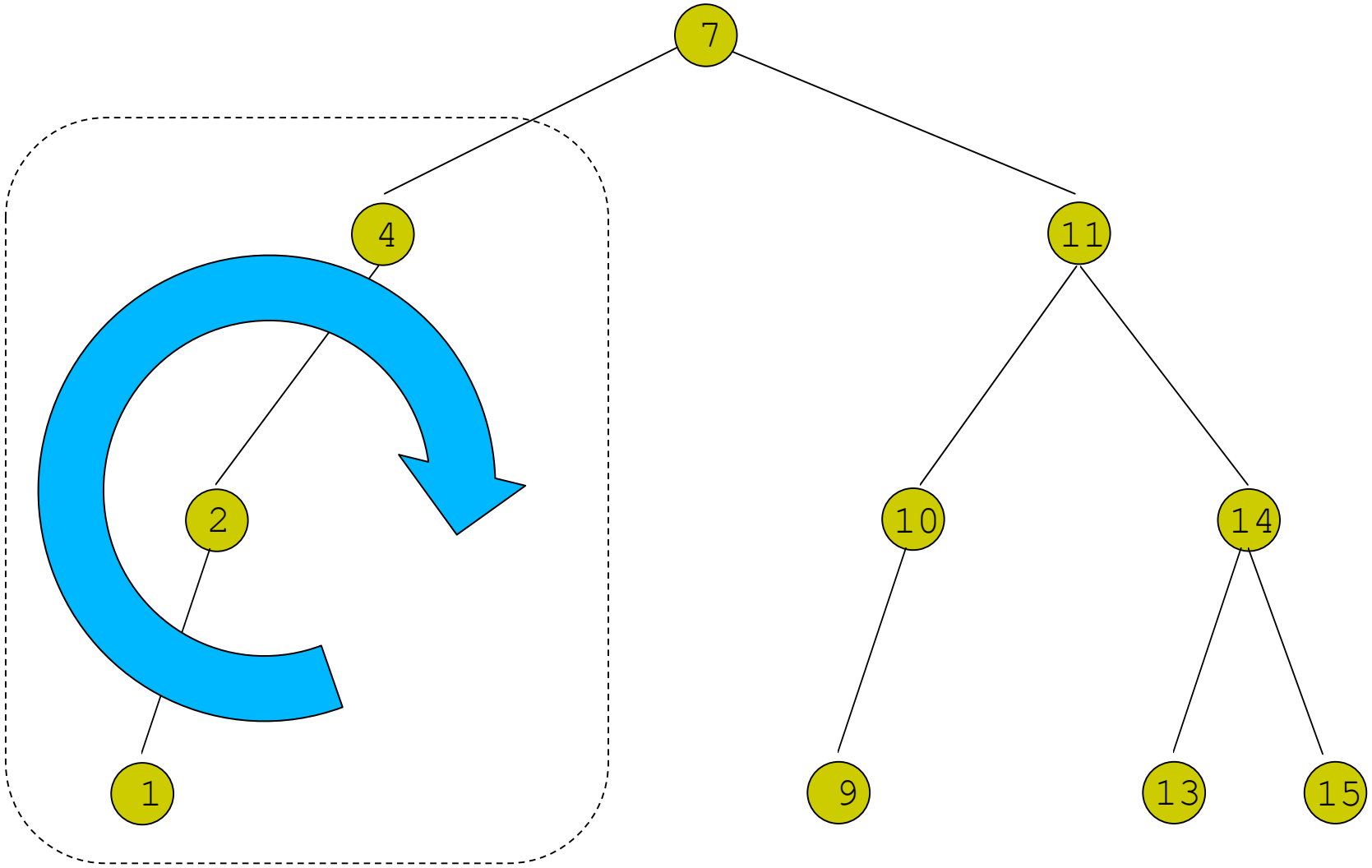

7
4
11
2
6
10
14
1
3
9
13
15

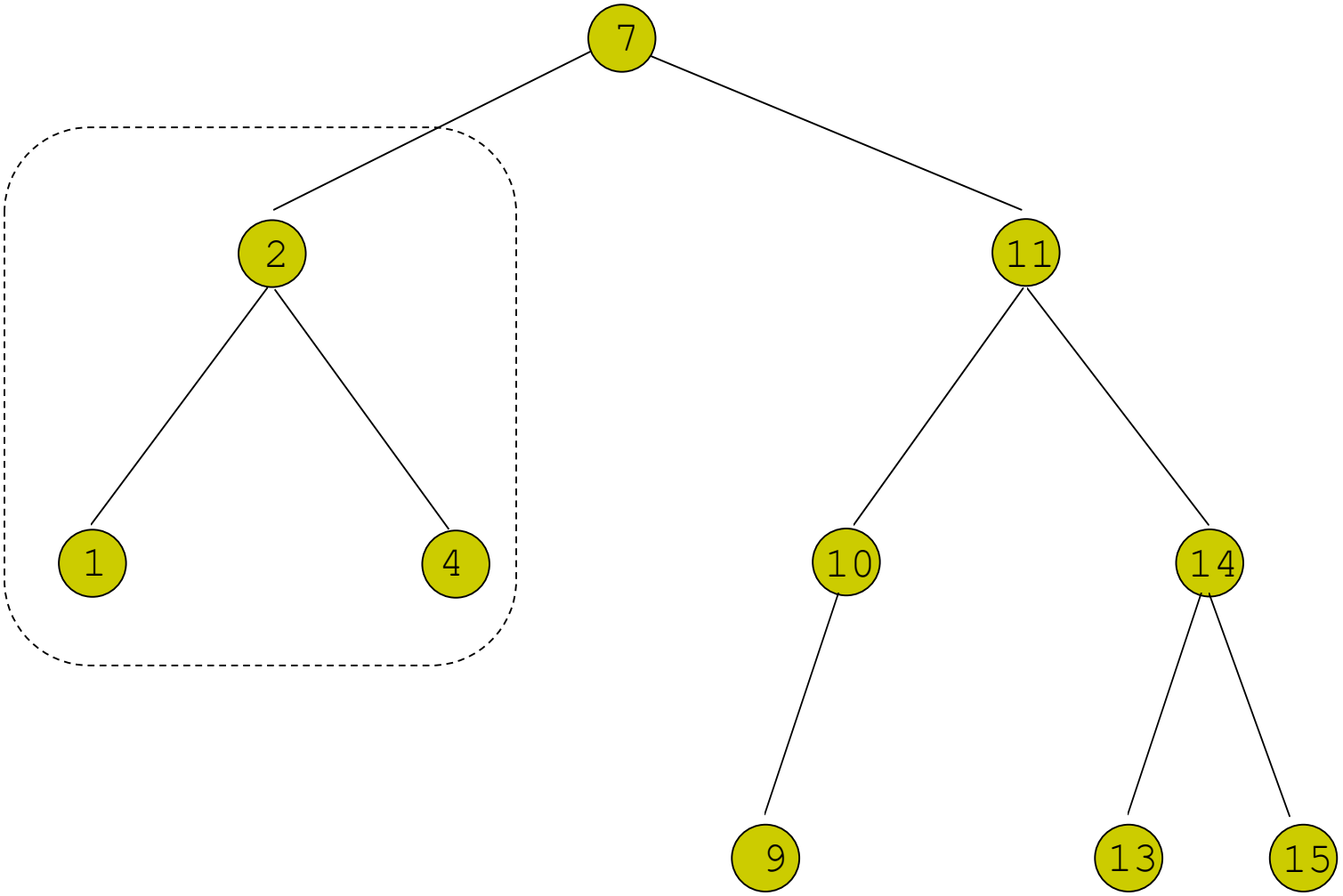

7
4
11
2
6
10
14
1
3
9
13
15


7
4
11
2
10
14
1
3
9
13
15


7
4
11
2
10
14
1
3
9
13
15


7
4
11
2
10
14
1
9
13
15

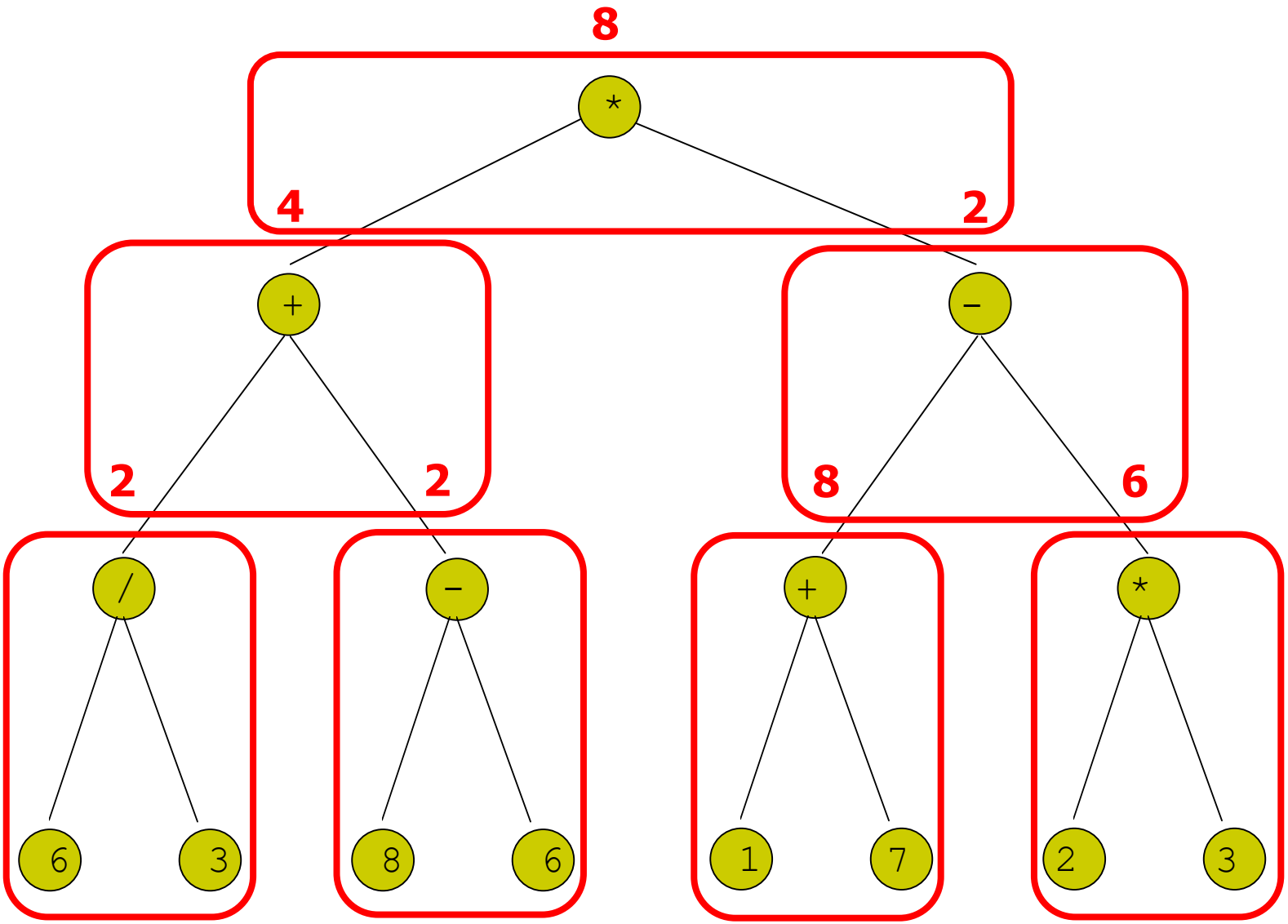

7
4
11
2
10
14
1
9
13
15


7
2
11
1
4
10
14
9
13
15

Αναπαράσταση/αποτίμηση εκφράσεων

- Τα δυαδικά δέντρα μπορεί να χρησιμοποιηθούν και για την εσωτερική αναπαράσταση/κωδικοποίηση εκφράσεων με δυαδικούς τελεστές
  - π.χ. αριθμητικών εκφράσεων
- Οι κόμβοι που δεν είναι φύλλα, περιέχουν τελεστές
- Τα ορίσματα τους (που μπορεί με την σειρά τους να είναι σύνθετες εκφράσεις) βρίσκονται στο αριστερό και στο δεξί υποδέντρο
- Η προτεραιότητα μεταξύ των πράξεων δίνεται από την απόσταση μιας τιμής από τον πιο κοντινό τελεστή
- Πράξεις που βρίσκονται στο ίδιο επίπεδο του δέντρου (έχουν την ίδια προτεραιότητα) μπορεί να εκτελεστούν παράλληλα μεταξύ τους

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

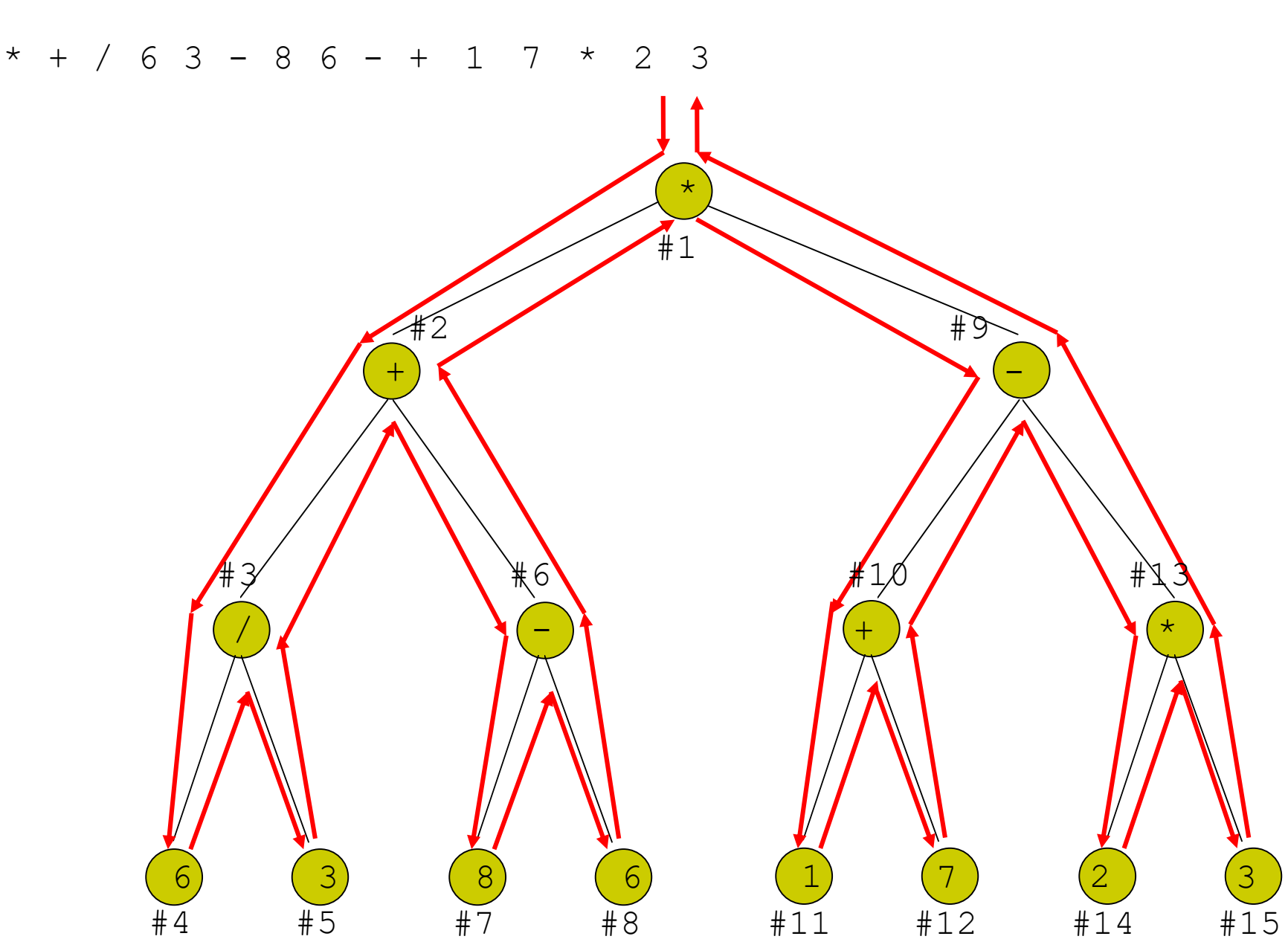


Διέλευση δέντρου έκφρασης

- Η διέλευση του δέντρου μπορεί να γίνει με 3 διαφορετικούς τρόπους

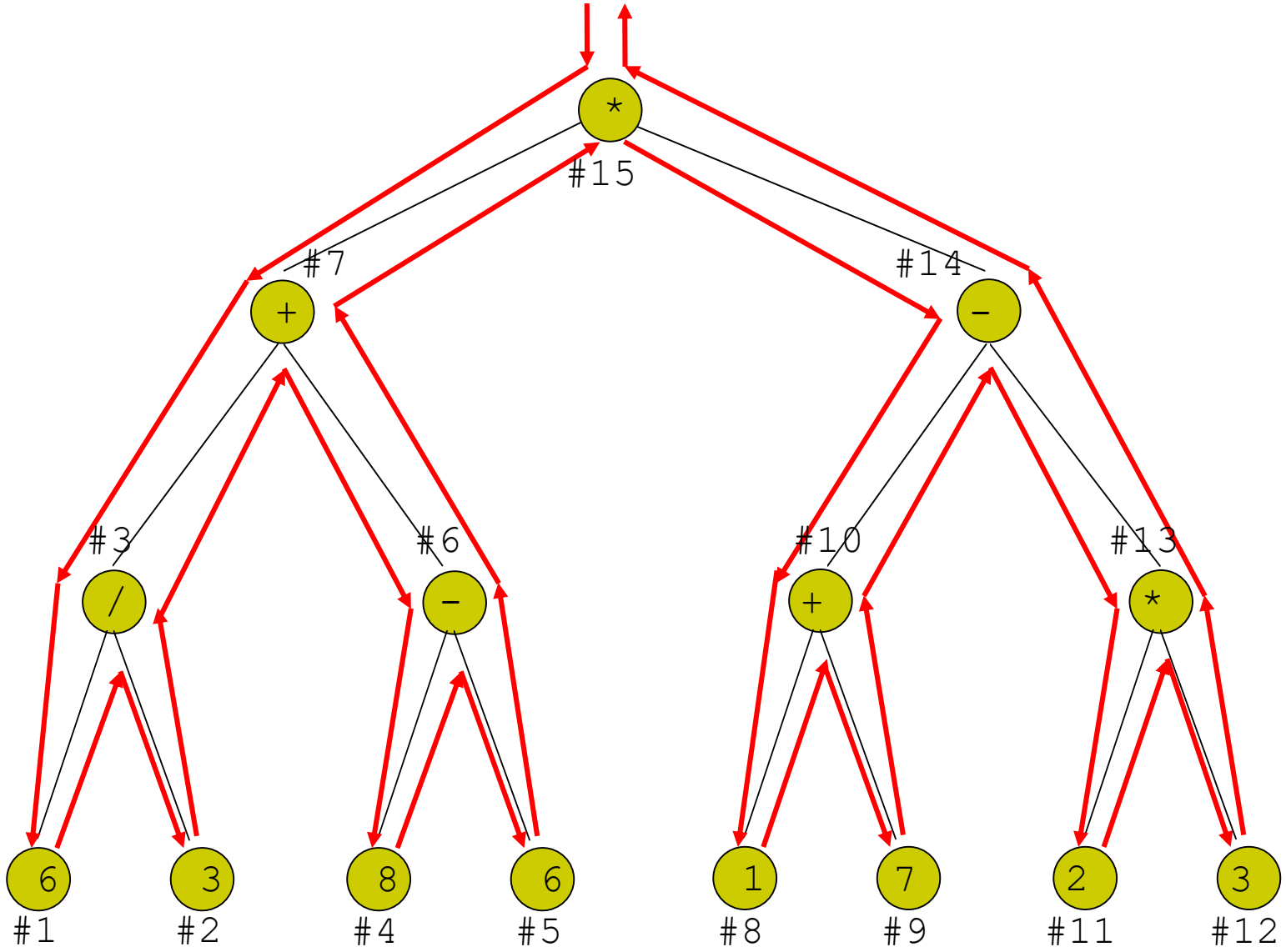
- **Prefix:** ο τελεστής πριν τα ορίσματα
- **Postfix:** ο τελεστής μετά τα ορίσματα
- **Infix:** ο τελεστής ανάμεσα στα ορίσματα

- Με αντίστοιχο τρόπο, μπορεί να γίνει συντακτική ανάλυση τέτοιων εκφράσεων, π.χ. από ένα διερμηνέα ή ένα μεταγλωττιστή, για να κατασκευαστεί μια τέτοια εσωτερική αναπαράσταση που στην συνέχεια μπορεί να χρησιμοποιηθεί για την αποτίμηση


* + / 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

```
void btree_traverse_prefix(struct btree *root) {
  if (root != NULL) {
    printf("%c ",root->v);
    btree_traverse_prefix(root->left);
    btree_traverse_prefix(root->right);
  }
}
```

6 3 / 8 6 - + 1 7 + 2 3 * - *



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